ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ, ΕΡΕΥΝΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ

ΙΝΣΤΙΤΟΥΤΟ ΕΚΠΑΙΔΕΥΤΙΚΗΣ ΠΟΛΙΤΙΚΗΣ

Βραχνός Ε., Κουρέτας Ι., Μακρυγιάννης Π., Παραδείση Α.

ΕΙΔΙΚΑ ΘΕΜΑΤΑ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΥΠΟΛΟΓΙΣΤΩΝ

Γ΄ ΤΑΞΗ ΕΠΑΛ

ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΗΜΕΙΩΣΕΙΣ ΜΑΘΗΤΗ

ΙΝΣΤΙΤΟΥΤΟ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ & ΕΚΔΟΣΕΩΝ
«ΔΙΟΦΑΝΤΟΣ»

ΙΝΣΤΙΤΟΥΤΟ ΕΚΠΑΙΔΕΥΤΙΚΗΣ ΠΟΛΙΤΙΚΗΣ
Πρόεδρος: **Σωτήριος Γκλαβάς**

ΓΡΑΦΕΙΟ ΕΡΕΥΝΑΣ, ΣΧΕΔΙΑΣΜΟΥ ΚΑΙ ΕΦΑΡΜΟΓΩΝ Β΄

Προϊστάμενος: **Μάραντος Παύλος**

ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Επιστημονικός Υπεύθυνος: **Δρ. Τσαπέλας Θεοδόσιος**, Σύμβουλος Β΄ Πληροφορικής ΙΕΠ

ΣΥΓΓΡΑΦΙΚΗ ΟΜΑΔΑ:

Βραχνός Ευριπίδης, καθηγητής πληροφορικής

Κουρέτας Ιωάννης, καθηγητής πληροφορικής

Μακρυγιάννης Παναγιώτης, καθηγητής πληροφορικής

Παραδείση Άρτεμις, καθηγήτρια πληροφορικής

ΕΠΙΜΕΛΕΙΑ ΣΥΝΤΟΝΙΣΜΟΣ ΟΜΑΔΑΣ:

Μακρυγιάννης Παναγιώτης, καθηγητής πληροφορικής

Παραδείση Άρτεμις, καθηγήτρια πληροφορικής

ΕΠΙΤΡΟΠΗ ΚΡΙΣΗΣ:

Βογιατζής Ιωάννης, Επίκουρος Καθηγητής Τ.Ε.Ι. Αθηνών

Μακρυγιάννης Ηλίας, καθηγητής πληροφορικής

Μελετίου Γεώργιος, Μηχανικός Πληροφορικής MSc,

Περιεχόμενα

Εικόνες

ΕΝΟΤΗΤΑ 1

ΕΝΟΤΗΤΑ 2

Πίνακες

ΕΝΟΤΗΤΑ 1

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Αντί Προλόγου

Η Java είναι μια γλώσσα προγραμματισμού η οποία μπορεί να υποστηρίξει την ανάπτυξη οποιασδήποτε εφαρμογής σε οποιαδήποτε πλατφόρμα, από τους προσωπικούς υπολογιστές και τα κινητά τηλέφωνα με Android μέχρι την ψηφιακή τηλεόραση. Ίσως σε κάποιες περιπτώσεις η απόδοση να μην είναι η αναμενόμενη, ωστόσο ο σύντομος χρόνος ανάπτυξης μιας εφαρμογής, ειδικά από μη έμπειρους προγραμματιστές είναι στα πλεονεκτήματα αυτής της γλώσσας. Το σύγγραμμα αυτό παρουσιάζει κάποιες βασικές έννοιες της γλώσσας μαζί με τις πιο βασικές βιβλιοθήκες της με σκοπό να μπορέσει στο τέλος ο μαθητής να αναπτύξει μόνος του μια μικρής έκτασης εφαρμογή, η οποία, αν είναι δυνατόν να συνδυάζει βάσεις δεδομένων, επικοινωνία μέσω δικτύου και αλληλεπίδραση με τον χρήστη μέσω παραθυρικού περιβάλλοντος.

Τα εργαλεία που θα χρησιμοποιήσουμε είναι δυο: Το περιβάλλον προγραμματισμού GreenFoot και το Eclipse. Αρχικά αξιοποιούμε τα εκπαιδευτικά πλεονεκτήματα του GreenFoot για μια πρώτη εισαγωγή στα αντικειμενοστρεφή χαρακτηριστικά της γλώσσας και στον γεγονοστρεφή προγραμματισμό. Η παρουσίαση των αλγοριθμικών δομών επιλογής και επανάληψης είναι σύντομη δεδομένου του ότι οι μαθητές τις έχουν ξανασυναντήσει και σε προηγούμενη τάξη, είτε σε ψευδογλώσσα είτε σε πραγματική γλώσσα προγραμματισμού (Pascal/Python).

Στη συνέχεια παρουσιάζεται το περιβάλλον Eclipse το οποίο είναι ένα εξαιρετικό περιβάλλον για ανάπτυξη εφαρμογών σε Java και έχει ενσωματωμένα πολλά εργαλεία ανάπτυξης εφαρμογών (Ant, javadoc, JUnit). Σε περίπτωση που το περιβάλλον Eclipse θεωρηθεί πολύ βαρύ για τους υπολογιστές του εργαστηρίου πληροφορικής μπορούμε να χρησιμοποιήσουμε έναν πολύ ελαφρύ συντάκτη πηγαίου κώδικα όπως το Notepad++ και μεταγλώττιση και εκτέλεση από τη γραμμή εντολών κάτι που δεν είναι ιδιαίτερα ελκυστικό για τους μαθητές. Ο εκπαιδευτικός, αν θέλει, μπορεί να χρησιμοποιήσει κάποιο άλλο περιβάλλον προγραμματισμού της Java το οποίο του επιτρέπει την επίτευξη των διδακτικών στόχων του Αναλυτικού Προγράμματος Σπουδών (ΑΠΣ) μπορεί να το κάνει.

Το ίδιο προαιρετική είναι και η χρήση της βιβλιοθήκης της Java Task Force (JTF) της ACM η οποία στην αρχή θα μας βοηθήσει να αποφύγουμε τις δυσκολίες που ανακύπτουν κατά τον προγραμματισμό λειτουργιών εισόδου/εξόδου. Η χρήση της βιβλιοθήκης θα μας βοηθήσει να αποκρύψουμε αρκετές επουσιώδεις λεπτομέρειες της γλώσσας που στα πρώτα μαθήματα μπορεί να δυσκολέψουν αρκετά τους μαθητές.

Ο γενικότερος σκοπός είναι να έρθουν οι μαθητές σε επαφή με τη φιλοσοφία ανάπτυξης μιας ολοκληρωμένης εφαρμογής σε Java και όχι να μάθουν τις λεπτομέρειες κάποιου πακέτου ή περιβάλλοντος προγραμματισμού που αύριο μπορεί να θεωρηθεί ξεπερασμένο. Άρα καθώς θα αναπτύσσουν την εφαρμογή τους είναι καλή πρακτική να αναζητούν στο διαδίκτυο βέλτιστες λύσεις σε προβλήματα που θα συναντήσουν μέσω δικτυακών τόπων όπως το *stackoverflow* και να μπορούν να χρησιμοποιούν την τεκμηρίωση των βιβλιοθηκών της Java ώστε να βρίσκουν γρήγορα και εύκολα αυτό που ψάχνουν.

Κεφάλαιο 1

Εισαγωγή στον Αντικειμενοστρεφή Προγραμματισμό

Εισαγωγή στον Αντικειμενοστρεφή Προγραμματισμό

Στόχοι

Οι μαθητές να μπορούν να:

- Περιγράφουν όλα τα στάδια ανάπτυξης ενός προγράμματος
- Αναγνωρίζουν τα συγκριτικά πλεονεκτήματα του αντικειμενοστρεφούς προγραμματισμού
- Υλοποιούν όλες τις φάσεις ανάπτυξης προγράμματος (σύνταξη, μετάφραση, διόρθωση συντακτικών λαθών, εκτέλεση) στο προγραμματιστικό περιβάλλον
- Περιγράφουν τις κλάσεις και τις υποκλάσεις
- Δημιουργούν και να διαγράφουν κλάσεις και αντικείμενα
- Αναγνωρίζουν ότι από μία κλάση μπορούν να δημιουργηθούν πολλά αντικείμενα τα οποία θα έχουν τα χαρακτηριστικά και τις μεθόδους της κλάσης αυτής

Ενότητες Κεφαλαίου

Η γλώσσα προγραμματισμού Java

Το περιβάλλον εκτέλεσης της γλώσσας

Γνωριμία με το περιβάλλον προγραμματισμού Greenfoot

Κλάσεις και Αντικείμενα

Υποκλάσεις και Υπερκλάσεις

Ιδιότητες Αντικειμένων

Ανάπτυξη απλών προγραμμάτων

1.1. Η γλώσσα προγραμματισμού Java

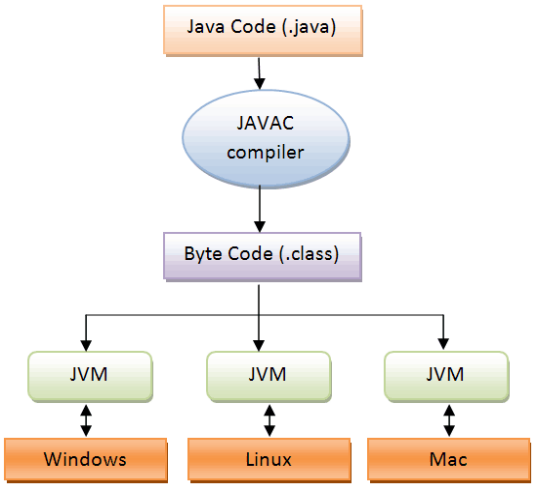
Η Java δημιουργήθηκε το 1995 από τη SUN και θεωρείται αυτή τη στιγμή μία από τις πιο δημοφιλείς γλώσσες στον χώρο της πληροφορικής. Στις 13 Νοεμβρίου του 2006 η Java έγινε πλέον μια γλώσσα ανοιχτού κώδικα (GPL) όσον αφορά το μεταγλωττιστή (javac) και το πακέτο ανάπτυξης (JDK, Java Development Kit). Από το 2010 και μετά η εταιρία λογισμικού Oracle Corporation εξαγόρασε την Sun Microsystems και μαζί με αυτήν όλες τις τεχνολογίες (πνευματικά δικαιώματα/ πατέντες) που η δεύτερη είχε στην κατοχή της ή είχε δημιουργήσει.

Η Java είναι μια γλώσσα που χρησιμοποιείται αρκετά για δικτυακές εφαρμογές ή για εφαρμογές που εκτελούνται μέσω browser στον παγκόσμιο ιστό (applets). Σύμφωνα με τους δημιουργούς της, η Java είναι μια *απλή, αντικειμενοστρεφής, κατανεμημένη, διερμηνευόμενη, ισχυρή, ασφαλής, πολυνηματική και δυναμική γλώσσα.*

Η Java σχεδιάστηκε αρχικά για την ανάπτυξη εφαρμογών ψηφιακής τηλεόρασης με την αρχική ονομασία Oak, όμως η ιδέα αυτή ήταν πολύ προχωρημένη για την εποχή της και εγκαταλείφθηκε. Ωστόσο αυτό που προέκυψε ήταν μια γλώσσα με σύνταξη που έμοιαζε πολύ στις C, C++ και με μια σημαντική για την εποχή εκείνη δυνατότητα. Να μπορεί να εκτελείται σε όλες τις πλατφόρμες. Το αρχικό σύνθημα ήταν “Write Once, Run Anywhere” (WORA). Αυτό σημαίνει ότι τα προγράμματα που είναι γραμμένα σε Java «τρέχουν» ακριβώς το ίδιο σε Windows, Linux, Unix, Macintosh και σύντομα θα «τρέχουν» ακόμα και σε κονσόλες παιχνιδιών, όπως το Playstation, χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση σε γλώσσα μηχανής ή να αλλάξει κάτι στον πηγαίο κώδικα που έγραψε ο προγραμματιστής.

Αυτό οφείλεται στο γεγονός ότι όταν ένα πρόγραμμα σε Java μεταγλωττίζεται, αυτό που παράγεται δεν είναι κώδικας μηχανής για έναν πραγματικό επεξεργαστή, αλλά ένα εξειδικευμένο είδος κώδικα γνωστό ως java bytecode, που «απευθύνεται» σε μια εικονική μηχανή, γνωστή ως Java Virtual Machine. Ο κώδικας αυτός εκτελείται από το περιβάλλον εκτέλεσης της Java (Java Runtime Environment, JRE), που είναι διαφορετικό για κάθε λειτουργικό σύστημα. Το JRE δεν είναι μεταγλωττιστής (compiler), αλλά διερμηνευτής (interpreter).

Άρα για να εκτελέσουμε μια εφαρμογή java στον υπολογιστή μας, πρέπει να έχουμε οπωσδήποτε το περιβάλλον εκτέλεσης της Java, δηλαδή το JRE. Για την ανάπτυξη εφαρμογών χρειαζόμαστε ακόμα τον μεταγλωττιστή της java (javac) για να μετατρέψουμε τον κώδικα που έχουμε γράψει σε java bytecode, και έναν καλό συντάκτη κώδικα (editor) στον οποίο θα γράψουμε το πρόγραμμά μας. Όλα αυτά γίνονται εύκολα και γρήγορα μέσα από ένα ολοκληρωμένο προγραμματιστικό περιβάλλον όπως είναι το Eclipse που διατίθεται δωρεάν από τη διεύθυνση http://www.eclipse.org και το οποίο θα χρησιμοποιήσουμε εδώ.



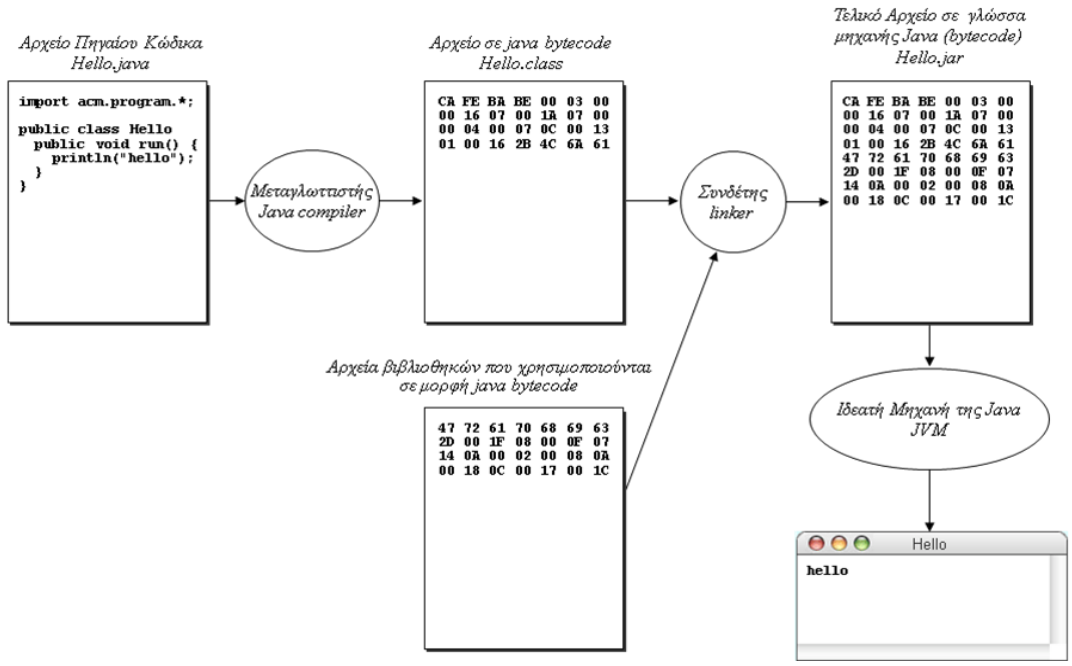
Εικόνα 1.1.1Διαδικασία μεταγλώττισης και εκτέλεσης προγραμμάτων Java

Εκτός από την ανεξαρτησία πλατφόρμας, μέρος της επιτυχίας της Java οφείλεται στο γεγονός ότι είναι η πρώτη γλώσσα που έχει σχεδιαστεί ειδικά για να επωφεληθεί από τη δύναμη του Παγκόσμιου Ιστού (www, 3w), που εμφανίστηκε λίγο πριν από την κυκλοφορία της, και συγκεκριμένα το 1995. Εκτός από τα παραδοσιακά προγράμματα, η Java δίνει τη δυνατότητα ανάπτυξης μικρών διαδραστικών προγραμμάτων των γνωστών **applets**, τα οποία μπορούν να εκτελεστούν υπό τον έλεγχο ενός φυλλομετρητή ιστού (web browser). Ωστόσο η αρχιτεκτονική του Παγκόσμιου Ιστού έχει εξελιχθεί και οι μικροεφαρμογές δεν έχουν πλέον απήχηση, γιατί έχουν αντικατασταθεί από πιο σύγχρονες τεχνολογίες, όπως Flash και HTML5.

1.2. Το περιβάλλον εκτέλεσης της γλώσσας

Κάθε σύστημα υπολογιστή μπορεί να εκτελέσει μια χαμηλού επιπέδου γλώσσα που είναι συγκεκριμένη για το είδος του υλικού. Η γλώσσα αυτή ονομάζεται γλώσσα μηχανής. Κάθε επεξεργαστής έχει τη δική του γλώσσα μηχανής. Επειδή όμως η γλώσσα μηχανής είναι αρκετά δυσνόητη στον άνθρωπο, η ανάπτυξη των εφαρμογών γίνεται στις λεγόμενες γλώσσες υψηλού επιπέδου. Για να εκτελεστεί ένα τέτοιο πρόγραμμα υπάρχουν δυο τρόποι: με μεταγλωττιστή (compiler) ή με διερμηνευτή (interpreter). Ο μεταγλωττιστής ελέγχει τον κώδικα για λάθη και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής για τον συγκεκριμένο τύπο υπολογιστή και το συγκεκριμένο λειτουργικό σύστημα. Ο διερμηνευτής ελέγχει και εκτελεί γραμμή – γραμμή και δεν παράγει εκτελέσιμο πρόγραμμα για συγκεκριμένο τύπο πλατφόρμας. Απλά κάθε πλατφόρμα είναι εφοδιασμένη με τον κατάλληλο διερμηνευτή, ο οποίος δέχεται ένα πρόγραμμα στη γλώσσα υψηλού επιπέδου και αφού ελέγξει μια-μια εντολή για την ορθότητά της στη συνέχεια την εκτελεί.

Η Java χρησιμοποιεί μια άλλη στρατηγική. Τα προγράμματα μεταγλωττίζονται αρχικά σε μια ενδιάμεση γλώσσα γνωστή ως java bytecode. Η γλώσσα αυτή είναι ουσιαστικά η γλώσσα μιας εικονικής μηχανής, γνωστής ως Java Virtual Machine (JVM). Στη συνέχεια το πρόγραμμα που προκύπτει σε μορφή java bytecode εκτελείται από τον διερμηνευτή κάθε πλατφόρμας.



Εικόνα 1.2.1 Διαδικασία ανάπτυξης προγράμματος Java

Στην Εικόνα 1.2.1 φαίνεται η διαδικασία ανάπτυξης ενός απλού προγράμματος σε Java. Αφού το γράψουμε σε κάποιον συντάκτη πηγαίου κώδικα, το πρόγραμμα περνάει από το μεταγλωττιστή (compiler) και παράγεται ο ενδιάμεσος κώδικας για την εικονική μηχανή της Java. Το αρχείο του πηγαίου κώδικα έχει κατάληξη java, π.χ. *Hello.java,* ενώ το αρχείο με τον κώδικα

μηχανής έχει κατάληξη .class και στην προκειμένη περίπτωση *Hello.class*. Κατόπιν γίνεται η σύνδεση με τις βιβλιοθήκες προγραμματισμού που χρησιμοποιούνται (*acm.program*) και παράγεται το τελικό αρχείο για εκτέλεση από την εικονική μηχανή της Java. Στη συνέχεια όποτε θέλουμε να εκτελέσουμε την εφαρμογή, καλούμε τη μηχανή εκτέλεσης της Java με όρισμα το αρχείο jar.

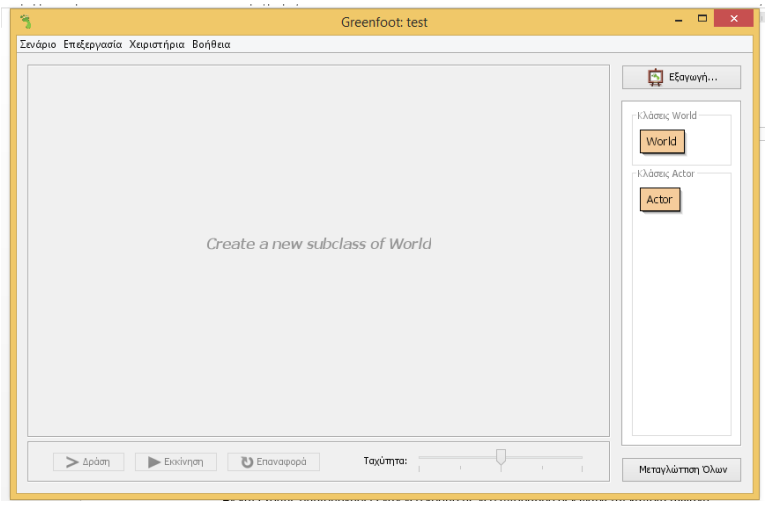
1.3. Γνωριμία με το περιβάλλον προγραμματισμού

1.3.1. Greenfoot

Το Greenfoot είναι ένα Ολοκληρωμένο Περιβάλλον Ανάπτυξης Εφαρμογών (IDE) που έχει ως σκοπό τη διδασκαλία του αντικειμενοστρεφούς προγραμματισμού με τη γλώσσα προγραμματισμού Java. Στο περιβάλλον αυτό δημιουργείται ένας κόσμος μέσα στον οποίο αλληλοεπιδρούν διάφορες μορφές (actors) και μπορούν να υλοποιηθούν προγράμματα που αφορούν σε παιχνίδια, προσομοιώσεις και γραφικά.

Η οθόνη του Greenfoot χωρίζεται σε τρία μέρη:

- Τον **Κόσμο** (World), μέσα στον οποίο ζουν τα αντικείμενα που δημιουργούμε και ο οποίος καταλαμβάνει το μεγαλύτερο μέρος της οθόνης.
- Την ιεραρχική **δομή των κλάσεων** (Class hierarchy), όπου φαίνονται σε ιεραρχική δομή όλοι οι τύποι των αντικειμένων που χρησιμοποιούμε (Οι τύποι αντικειμένων 'World' και 'Actor' εμφανίζονται σε όλα τα σενάρια του Greenfoot). Όλοι οι τύποι των αντικειμένων επεκτείνουν τους αρχικούς τύπους World και Actor και υιοθετούν τις ιδιότητες και την όποια λειτουργικότητά τους.
- Την **κονσόλα χειρισμού,** που βρίσκεται στο κάτω μέρος της οθόνης. Η εκτέλεση ενός σεναρίου γίνεται με το πάτημα του κουμπιού ***Εκκίνηση***. Η ρύθμιση της ταχύτητας εκτέλεσης γίνεται με την μπάρα κύλισης ***Ταχύτητα***. Για να εκτελούμε βήμα - βήμα το σενάριο πατάμε το κουμπί ***Δράση***. Αν θέλουμε να ξεκινήσουμε την εκτέλεση από την αρχή πατάμε το κουμπί ***Επαναφορά***.



Εικόνα 1.3.1 Το βασικό παράθυρο του Greenfoot

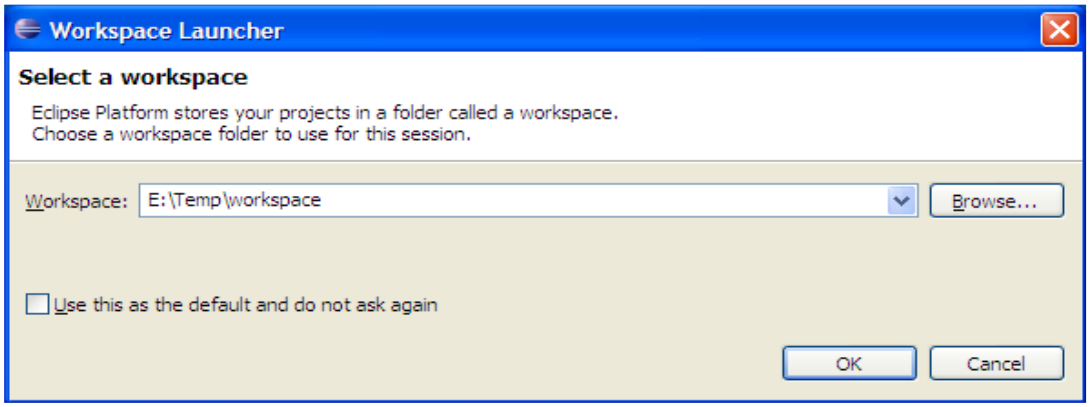
Στο πάνω μέρος της οθόνης υπάρχουν το οριζόντιο μενού: **Σενάριο**, **Επεξεργασία**, **Χειριστήρια** και **Βοήθεια**.

Η αποθήκευση ενός αντίγραφου του σεναρίου, γίνεται με κλικ στην επιλογή **Save as…** στο μενού **Σενάριο**.

1.3.2. Eclipse

Το Eclipse είναι ένα δημοφιλές περιβάλλον προγραμματισμού για τη γλώσσα Java, αλλά χρησιμοποιείται και για τον προγραμματισμό άλλων γλωσσών όπως: Ada, C, C++, Perl, PHP, Python, Ruby κ.α. Το εργαλείο αυτό προσφέρει ευκολίες στην εκτέλεση αυτόνομων προγραμμάτων όπως και μικροεφαρμογών (applets). Τέλος για το Eclipse υπάρχει και το Android Development Tool (ADT) plug in, για τη δημιουργία εφαρμογών Android.

Στο Eclipse ο χρήστης θα πρέπει να επιλέξει ένα φάκελο στο δίσκο για το workspace όπου θα είναι αποθηκευμένα τα project του (Εικόνα 1.3.2.1). Μπορούν να αποθηκευτούν πολλά project στο ίδιο workspace αλλά σε διαφορετικούς υποφακέλους. Το Eclipse υποστηρίζει πολλαπλούς χώρους εργασίας (workspace) και μπορούμε να επιλέξουμε workspace, όταν ξεκινάμε το Eclipse.



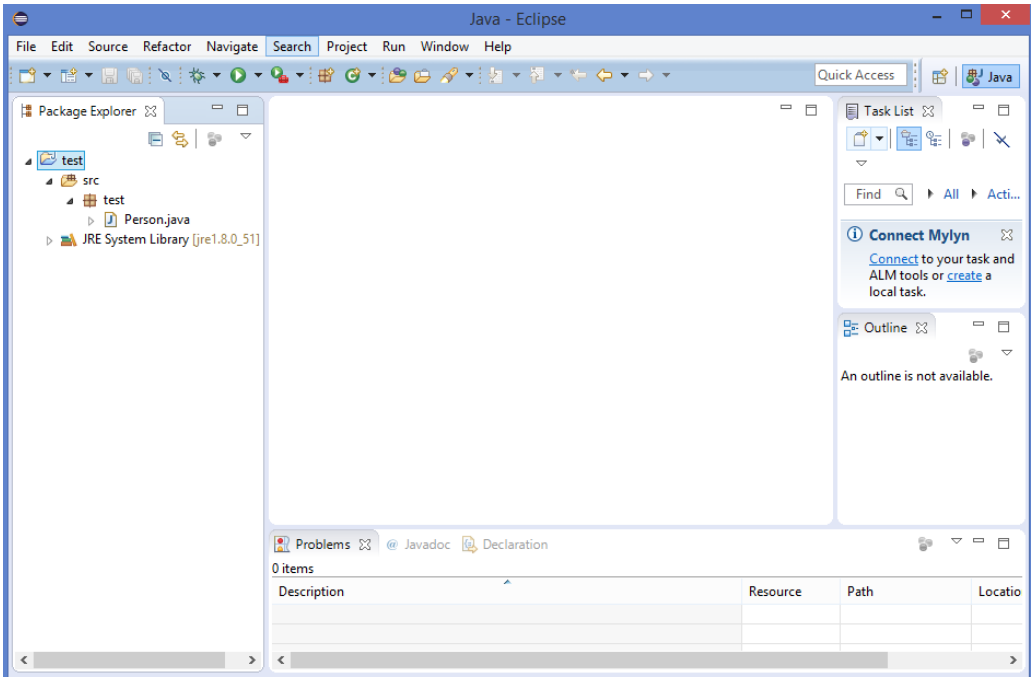
Εικόνα 1.3.2 Επιλογή του χώρου εργασίας (workspace) στο Eclipse

Μόλις «ανοίξει» το Eclipse εμφανίζεται η βασική οθόνη του που αποτελείται από παράθυρα στα οποία αναπτύσσεται μια εφαρμογή (Εικόνα 1.3.2.2). Τα βασικά είναι:

Package Explorer – Ιεραρχικός τρόπος παρουσίασης των εφαρμογών σε μορφή φακέλων και αρχείων.

Problems – Απεικονίζει προειδοποιήσεις (Warnings) και σφάλματα (Errors) κατά τη διάρκεια ανάπτυξης.

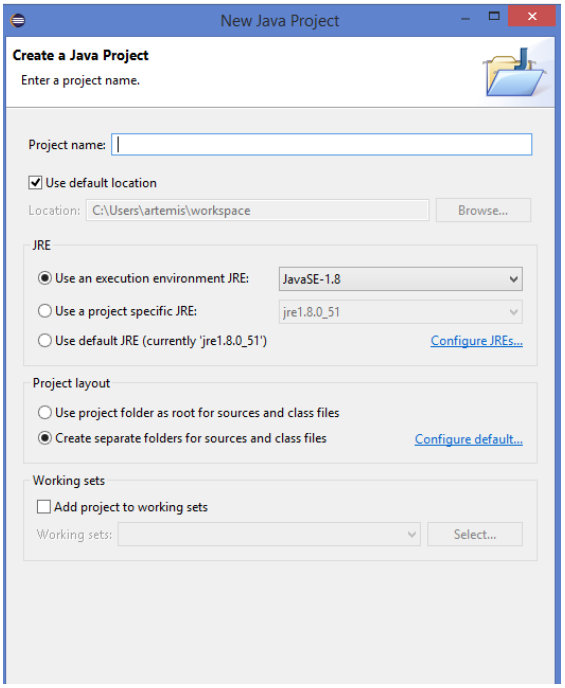
Το κεντρικό και μεγαλύτερο παράθυρο - Το χρησιμοποιούμε για να γράψουμε κώδικα.



Εικόνα 1.3.3 Το περιβάλλον εργασίας του Eclipse

Στο επάνω μέρος της οθόνης βλέπουμε μια μπάρα με διάφορα μενού και από κάτω ακριβώς μια άλλη μπάρα με διάφορα κουμπιά συντομίας ορισμένων βασικών λειτουργιών.

Η δημιουργία του Project γίνεται από το μενού **File→New→Java Project**. Στο παράθυρο διαλόγου που ανοίγει δίνουμε το όνομα του έργου (Εικόνα 1.3.2.3). Αφήνουμε τις άλλες επιλογές του παραθύρου διαλόγου στις προεπιλεγμένες τιμές. Το Eclipse δημιουργεί αυτόματα το φάκελο src και το φάκελο με τις βιβλιοθήκες του συστήματος.



Εικόνα 1.3.4 Δημιουργία ενός νέου project

Στη συνέχεια, θα πρέπει να δημιουργήσουμε τις κλάσεις που αποτελούν τον κορμό του προγράμματος. Η δημιουργία της κλάσης γίνεται από το μενού **File→New→Class**, οπότε εμφανίζεται το παράθυρο διαλόγου που αφορά την κλάση. Επιλέγουμε το είδος της κλάσης και της δίνουμε όνομα. Το πρόγραμμα τη δημιουργεί αυτόματα, όπως επίσης και τη δομή της. Στη συνέχεια μπορούμε να ξεκινήσουμε να γράφουμε τον κώδικα μας.

Για να εκτελέσουμε το πρόγραμμά μας, επιλέγουμε από το μενού **Run→**.

1.4. Κλάσεις και Αντικείμενα

Οι κλάσεις (classes) και τα αντικείμενα (objects or instances) είναι θεμελιώδεις έννοιες στον αντικειμενοστρεφή προγραμματισμό.

Βασική αρχή πάνω στην οποία στηρίζεται ο Αντικειμενοστρεφής Προγραμματισμός είναι η προσέγγιση του πραγματικού κόσμου, ο οποίος αποτελείται από **αντικείμενα** απλά ή σύνθετα. Τα αντικείμενα έχουν ένα σύνολο **χαρακτηριστικών - ιδιοτήτων (attributes)** που προσδιορίζουν τη φυσική τους υπόσταση. Κάθε αντικείμενο επίσης, έχει κανόνες συμπεριφοράς και μπορεί να **εκτελεί συγκεκριμένες ενέργειες (μεθόδους)**, ανάλογα με τα μηνύματα που λαμβάνει από το περιβάλλον στο οποίο βρίσκεται. Επίσης, ένα αντικείμενο μπορεί να επιδρά πάνω σ' ένα άλλο αντικείμενο στέλνοντας του το κατάλληλο μήνυμα.

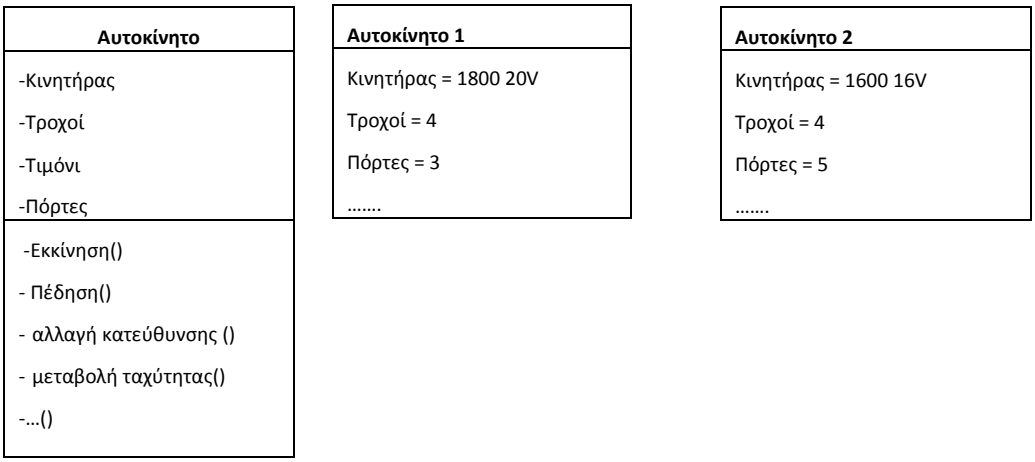
Ως παραδείγματα αντικειμένων από τον πραγματικό κόσμο μπορούμε να αναφέρουμε τον Άνθρωπο, το τηλεχειριστήριο και την τηλεόραση. Κάθε Άνθρωπος έχει κάποια ιδιαίτερα χαρακτηριστικά – ιδιότητες όπως το όνομα του, την ηλικία του, το φύλο του, το χρώμα των ματιών του κ.λπ. Επίσης μπορεί να εκτελεί διάφορες ενέργειες όπως να περπατά, να τρώει, να τρέχει, να γελάει, να πατάει κουμπιά κ.λπ. Το τηλεχειριστήριο έχει χαρακτηριστικά όπως μήκος, χρώμα, πλήκτρα κ.λπ. και εκτελεί την ενέργεια εκπομπής σήματος. Η τηλεόραση έχει χαρακτηριστικά όπως μέγεθος, ευκρίνεια κ.λπ. και εκτελεί διάφορες ενέργειες όπως κλείσιμο, αλλαγή καναλιού, αυξομείωση έντασης ήχου κ.λπ., ανάλογα με τα ερεθίσματα που δέχεται από το περιβάλλον της. Πατώντας ο άνθρωπος το κουμπί αλλαγής τηλεοπτικού σταθμού στο τηλεχειριστήριο, αυτό με τη σειρά του στέλνει κατάλληλο μήνυμα προς την τηλεόραση, η οποία στη συνέχεια, εκτελεί τη συγκεκριμένη ενέργεια αλλάζοντας κανάλι. Πατώντας ο άνθρωπος το κουμπί αύξησης της έντασης του ήχου στο τηλεχειριστήριο, αυτό με τη σειρά του στέλνει διαφορετικό μήνυμα στην τηλεόραση, η οποία στη συνέχεια αυξάνει την ένταση του ήχου. Πως γνωρίζουμε όμως τι ενέργειες είναι σε θέση να εκτελεί το αντικείμενο τηλεόραση; Ο όρος «τηλεόραση» αναπαριστά μια οικογένεια αντικειμένων του ίδιου τύπου (**κλάση**). Οι ενέργειες που μπορεί να εκτελεί ένα αντικείμενο ως απόκριση σε ένα μήνυμα, έχουν καθοριστεί από την κλάση του. Όλα τα αντικείμενα της κλάσης τηλεόραση εκτελούν την ίδια ενέργεια ως απόκριση στον ίδιο τύπο μηνύματος (π.χ. αλλαγή τηλεοπτικού σταθμού).

Συνοψίζοντας θα λέγαμε ότι μία κλάση καθορίζει τις αρχικές ιδιότητες και τις συμπεριφορές των αντικειμένων που προέρχονται από αυτή. Η κλάση είναι το πρότυπο, το σχέδιο, η αφηρημένη περιγραφή του αντικειμένου. Το αντικείμενο είναι η χαρακτηριστική περίπτωση εφαρμογής μιας κλάσης ή όπως λέμε ένα συγκεκριμένο στιγμιότυπο της κλάσης.

Παράδειγμα:

Η κλάση Αυτοκίνητο έχει χαρακτηριστικά (μηχανή, ρόδες, τιμόνι, μηχανισμός πλοήγησης, κ.ά.) και λειτουργίες (εκκίνηση, πέδηση, μεταβολή ταχύτητας, αλλαγή κατεύθυνσης, κ.ά.), οι οποίες καθορίζουν τη συμπεριφορά της κλάσης.

Το αντικείμενο Αυτοκίνητο 1 έχει συγκεκριμένες τιμές στις ιδιότητες που ορίζονται από την κλάση του, ενώ το Αυτοκίνητο 2 έχει διαφορετικές τιμές στις ιδιότητες του. Δηλαδή:

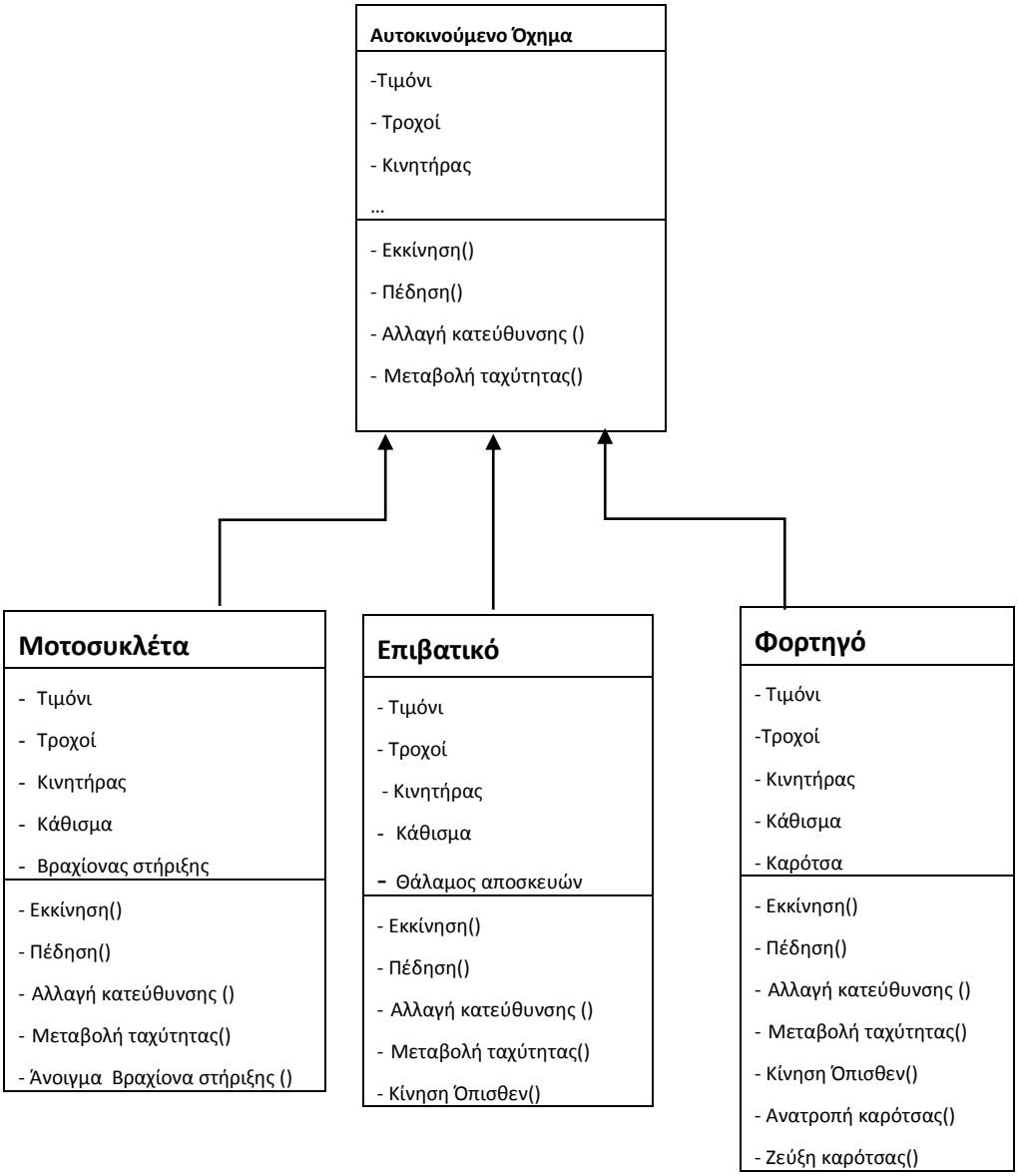


1.5. Υποκλάσεις και Υπερκλάσεις

Στην Java (και γενικότερα στον αντικειμενοστρεφή προγραμματισμό) από μία κλάση μπορεί να δημιουργηθεί μια νέα κλάση απόγονος που κληρονομεί όλα τα χαρακτηριστικά και τις λειτουργίες της μητρικής κλάσης. Μια κλάση που κληρονομεί χαρακτηριστικά και λειτουργίες από μια άλλη κλάση καλείται **υποκλάση (subclass)** και η κλάση από την οποία κληρονομούνται καλείται **υπερκλάση (superclass).**

Η ιδέα είναι, να υπάρχει μια γενική κλάση (**υπερκλάση**), με κάποια χαρακτηριστικά και λειτουργίες, από την οποία μπορούμε να ορίσουμε εξειδικευμένες παραλλαγές της (**υποκλάσεις**). Μία υποκλάση μπορεί να τροποποιεί κάποιες από τις λειτουργίες της υπερκλάσης της ή να ορίζει επιπλέον και τις δικές της μοναδικές ιδιότητες και λειτουργίες. Μια κλάση μπορεί να έχει απεριόριστο αριθμό υποκλάσεων αλλά μόνο μια υπερκλάση. Στον αντικειμενοστρεφή προγραμματισμό η επέκταση των κλάσεων σε πιο εξειδικευμένες κλάσεις είναι γνωστή ως **κληρονομικότητα.**

Στην Εικόνα 1.5.1 η κλάση Αυτοκινούμενο Όχημα είναι η υπερκλάση, ενώ η Μοτοσυκλέτα, το Επιβατικό και το Φορτηγό είναι οι υποκλάσεις της όπου η κάθε μια είναι μια εξειδικευμένη παραλλαγή της.



Εικόνα 1.5.1 Υπερκλάση - Υποκλάσεις

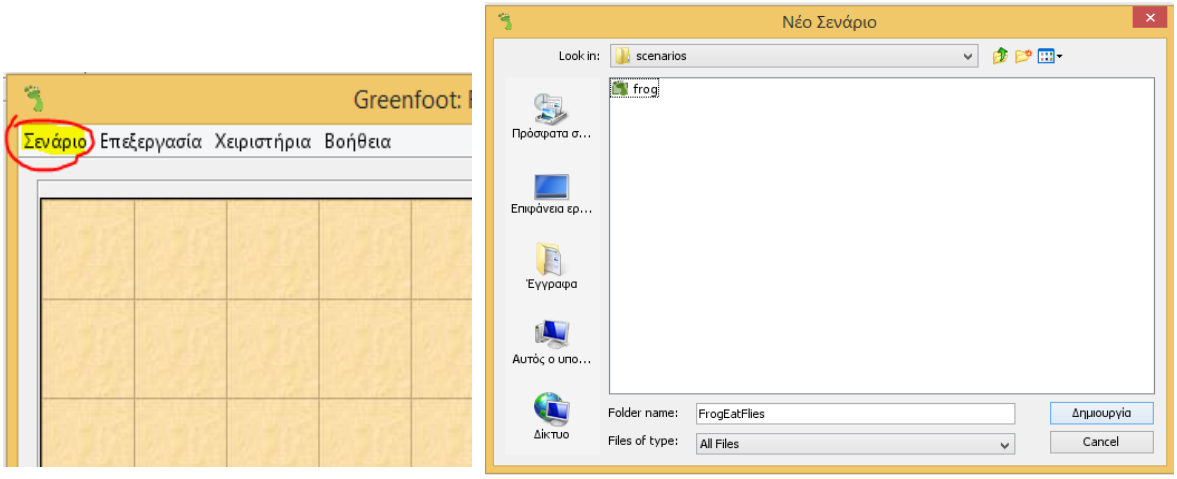
Στο παραπάνω παράδειγμα, η κλάση **Επιβατικό** κληρονομεί όλα τα χαρακτηριστικά και τις λειτουργίες της υπερκλάσης της **Αυτοκινούμενο Όχημα**, όπως τιμόνι, τροχοί, εκκίνηση, πέδηση κ.λπ. και επιπλέον ορίζει νέα χαρακτηριστικά και λειτουργίες όπως το κάθισμα, η Κίνηση Όπισθεν κ.α. Η κλάση **Μοτοσυκλέτα** κληρονομεί όλα τα χαρακτηριστικά και τις λειτουργίες της υπερκλάσης της **Αυτοκινούμενο Όχημα**, όπως τιμόνι, τροχοί, εκκίνηση, πέδηση κ.λπ. και επιπλέον ορίζει νέα χαρακτηριστικά και λειτουργίες όπως το κάθισμα, Βραχίονας στήριξης, Άνοιγμα Βραχίονα στήριξης κ.λπ.

1η εργαστηριακή άσκηση:

Περιγραφή σεναρίου: ένας βάτραχος κινείται μέσα στο πλέγμα του Κόσμου FrogWorld και προσπαθεί να φάει όσες περισσότερες μύγες μπορεί, από αυτές που πιθανόν υπάρχουν μέσα σε αυτό.

Ανοίγουμε το περιβάλλον Greenfoot και δημιουργούμε ένα νέο σενάριο πατώντας με το ποντίκι στο μενού *Σενάριο* και επιλέγοντας *Νέο.* Στην οθόνη που εμφανίζεται δίνουμε στο όνομα

φακέλου τη λέξη FrogEatFlies (ο τίτλος του σεναρίου μας) και πατάμε το κουμπί *Δημιουργία* (Εικόνα 1.5.2).



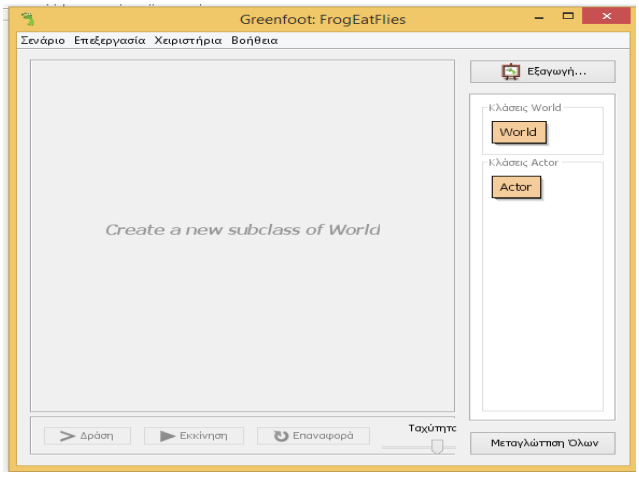
Εικόνα 1.5.2 Δημιουργία νέου σεναρίου στο Greenfoot

Το σενάριο μιας συγκεκριμένης εφαρμογής αποτελείται από μια ομάδα από κλάσεις και το σύνολο του κώδικα που κατασκευάζονται για την υλοποίησή της. Κάθε λειτουργικό σενάριο στο Greenfoot, θα πρέπει να έχει τουλάχιστον έναν Κόσμο, και κάποιες μορφές (Actors).

Στην περίπτωση του σεναρίου μας οι μορφές που θα χρησιμοποιήσουμε είναι δύο, μια μορφή με όνομα Frog (ένας βάτραχος) και μια μορφή με όνομα Fly (μύγα). Ο βάτραχος και η μύγα έχουν ως χαρακτηριστικά: τις συντεταγμένες τους (τετμημένη και τεταγμένη) στο πλέγμα του Κόσμου, τον προσανατολισμό τους σχετικά με τον οριζόντιο άξονα σε μοίρες, τη μορφή που έχουν, τον Κόσμο που ανήκουν και μπορούν να εκτελούν τις ακόλουθες ενέργειες: να κινούνται μέσα στο πλέγμα του Κόσμου FrogWorld και να στρίβουν. Ο βάτραχος έχει επιπλέον ως χαρακτηριστικό τον αριθμό των μυγών που έχει φάει και ως ενέργεια που μπορεί να εκτελεί το να τρώει μύγες. Σε επόμενα μαθήματα θα προσθέτουμε επιπλέον χαρακτηριστικά στην εφαρμογή μας, όπως τον έλεγχο των αντικειμένων από τον χρήστη, ένα μετρητή πόντων κτλ.

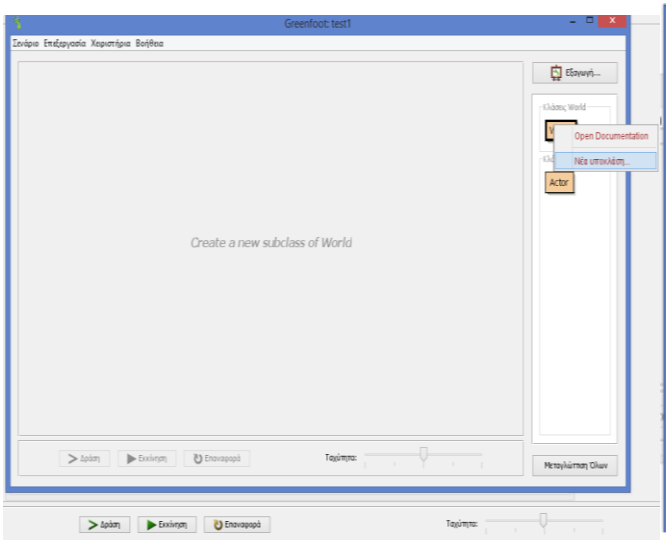
Στην οθόνη μας δεξιά βλέπουμε ότι στην ιεραρχία των κλάσεων του Greenfoot υπάρχουν πάντα δύο κλάσεις:

- Η **World**, που παρέχει την εικόνα του υπόβαθρου για τον Κόσμο του σεναρίου μας και καθορίζει το μέγεθος και την ανάλυση του, και
- Η **Actor**, που παράγει τα αντικείμενα ή στιγμιότυπα που ενεργούν στο σενάριο.

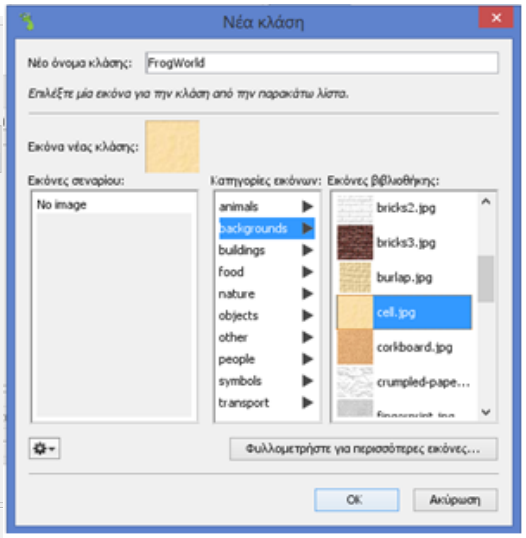


Οι κλάσεις αυτές αποτελούν τις **υπερκλάσεις** όλων των κλάσεων που απεικονίζονται γραφικά στην οθόνη.

Τώρα χρειαζόμαστε να δημιουργήσουμε μία νέα κλάση (έναν νέο Κόσμο), η οποία αποτελεί υποκλάση της κλάσης World. Αυτή η νέα κλάση θα είναι ο Κόσμος μέσα στον οποίο θα κινούνται οι μορφές του σεναρίου μας. Για να δημιουργήσουμε το νέο Κόσμο κάνουμε με το ποντίκι δεξί κλικ πάνω στο εικονίδιο World. Από το μενού που εμφανίζεται επιλέγουμε το πεδίο *Νέα υποκλάση και την* ονομάζουμε FrogWorld, με εικόνα background: cell.jpg (Εικόνες 1.5.3 και 1.5.4).



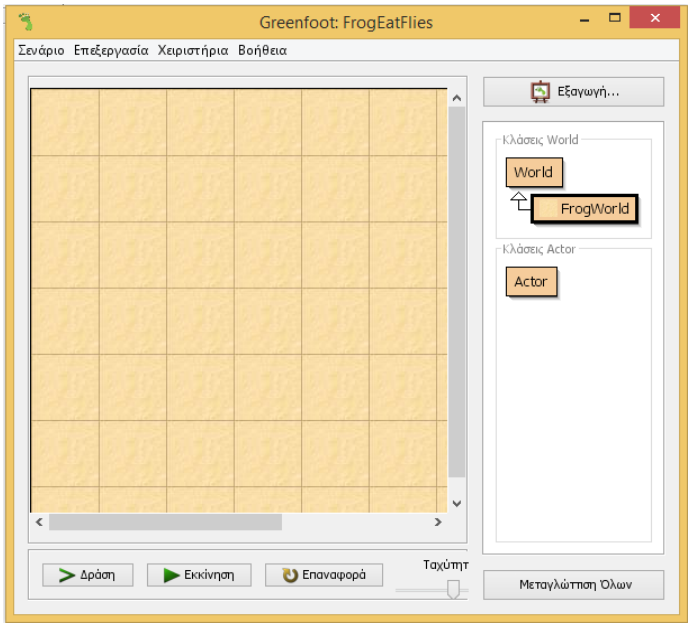
Εικόνα 1.5.3 Δημιουργία υποκλάσης



Εικόνα 1.5.4 Νέο όνομα κλάσης

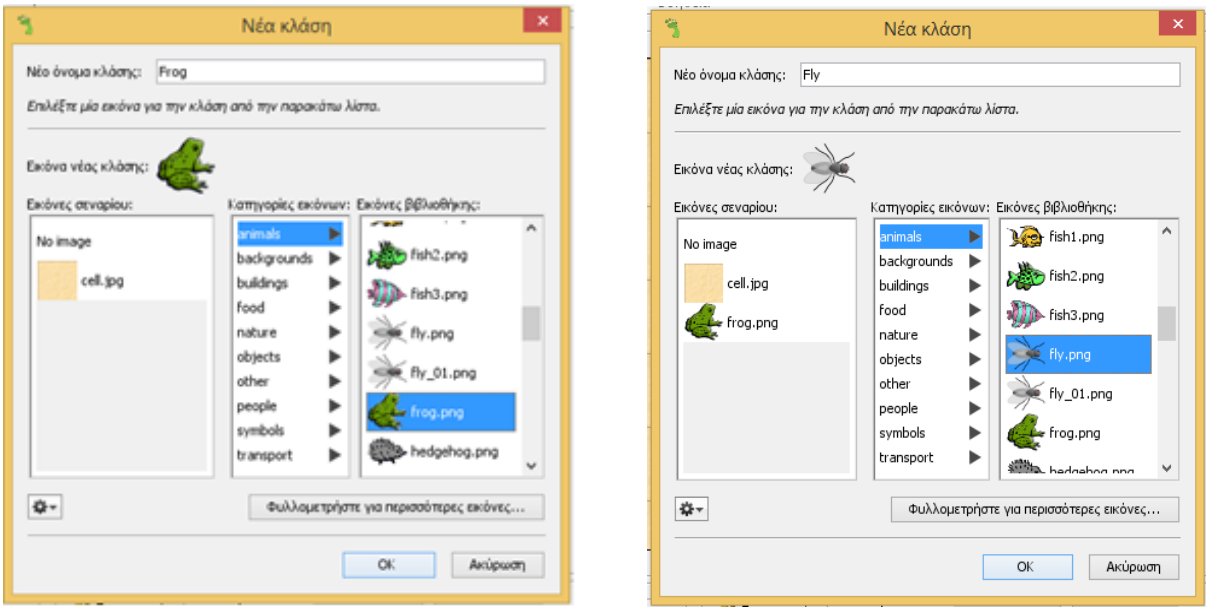
Σύμφωνα με τα όσα είπαμε παραπάνω, η FrogWorld είναι υποκλάση της World, επεκτείνει δηλαδή την υπερκλάση World με νέα χαρακτηριστικά με τα οποία και εξειδικεύεται (π.χ. υπόβαθρο, διαστάσεις κ.λπ.). Η σχέση υπερκλάσης-υποκλάσης απεικονίζεται γραφικά στο περιβάλλον του Greenfoot με το βέλος που συνδέει την κλάση FrogWorld με την κλάση World (Εικόνα 1.5.5).

Οι αλλαγές που πραγματοποιήσαμε θα εμφανισθούν στο περιβάλλον του Greenfoot αφού πρώτα μεταγλωττίσουμε το πρόγραμμά μας, πατώντας το κουμπί ***Μεταγλώττιση όλων***.

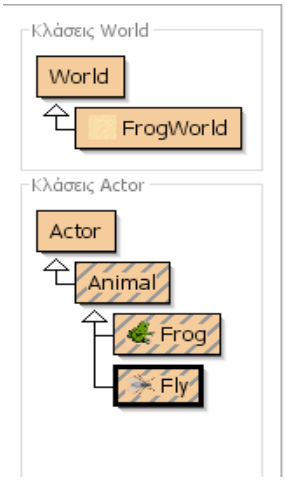


Εικόνα 1.5.5 Σχέση μεταξύ των κλάσεων

Το επόμενο βήμα είναι να δημιουργήσουμε τις κλάσεις για τα αντικείμενα που θα μας χρειαστούν στο σενάριο μας, δηλαδή το βάτραχο και τη μύγα. Μια καλή σχεδίαση του Κόσμου είναι να δημιουργήσουμε μια υποκλάση της Actor, την κλάση Animal (ζώα) και δύο υποκλάσεις της, τη Frog και Fly. Η κλάση Animal δεν θα έχει κάποια εικόνα, ενώ στις κλάσεις Frog και Fly θα δώσουμε τις εικόνες frog.png και fly.png αντίστοιχα, που θα τις βρούμε μέσα στην κατηγορία εικόνων «animals» (Εικόνα 1.5.6).



Εικόνα 1.5.6 Οι νέες κλάσεις Frog και Fly

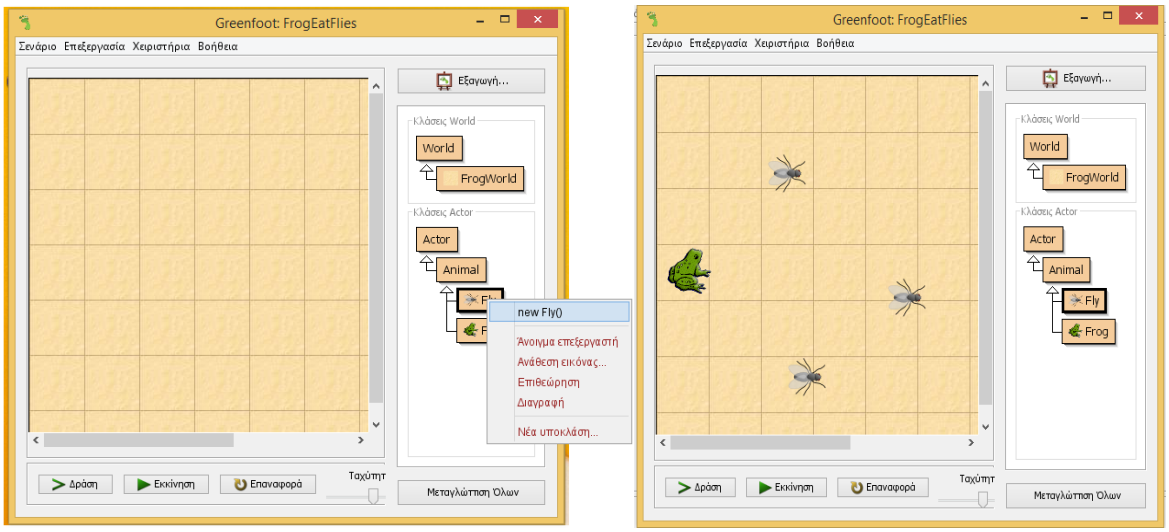


Όταν κάνουμε μια αλλαγή στον πηγαίο κώδικα ή στην δομή μιας κλάσης, το αντίστοιχο εικονίδιο εμφανίζεται με ρίγες, κάτι που σημαίνει ότι οι αλλαγές δεν έχουν περάσει σε γλώσσα μηχανής.

Στην διπλανή εικόνα, όλες οι κλάσεις εκτός της Actor, έχουν τροποποιηθεί και χρειάζονται μεταγλώττιση (για να ενσωματωθούν οι αλλαγές και σε γλώσσα μηχανής) ώστε, όταν εκτελέσουμε το πρόγραμμα, να εκτελεστεί η τελευταία ενημερωμένη έκδοση.

Για να γίνει αυτό θα πρέπει να μεταγλωττίσουμε όλη τη δομή που έχουμε φτιάξει μέχρι εκείνη τη στιγμή, πατώντας το πλήκτρο *Μεταγλώττιση Όλων* του Greenfoot. Έτσι θα μεταφραστεί ξανά ο πηγαίος κώδικας σε κώδικα java bytecode, αυτόν δηλαδή που καταλαβαίνει η εικονική μηχανή της Java (Java Virtual Machine).

Στο παράδειγμα μας, όπως βλέπουμε και στην εικόνα 1.5.5, ο Κόσμος FrogWorld είναι κενός, καθώς δεν έχει δημιουργηθεί ακόμα τίποτα. Θα πρέπει να δημιουργήσουμε κάποια αντικείμενα των κλάσεων που έχουμε φτιάξει και να «δώσουμε ζωή» στα αντικείμενα αυτά. Η δημιουργία των αντικειμένων γίνεται με δεξί κλικ πάνω στην αντίστοιχη κλάση και στη συνέχεια επιλογή της ενέργειας **new** Frog(), **new** Fly() αντίστοιχα (Εικόνα 1.5.7). Τα αντικείμενα που δημιουργούνται μπορούμε να τα σύρουμε με το ποντίκι σε όποιο σημείο της επιφάνειας εργασίας (του Κόσμου) θέλουμε. Με αυτό τον τρόπο μπορούμε να δημιουργήσουμε, σε κάθε κλάση, όσα αντικείμενα επιθυμούμε.



Εικόνα 1.5.7 Δημιουργία αντικειμένων κλάσης

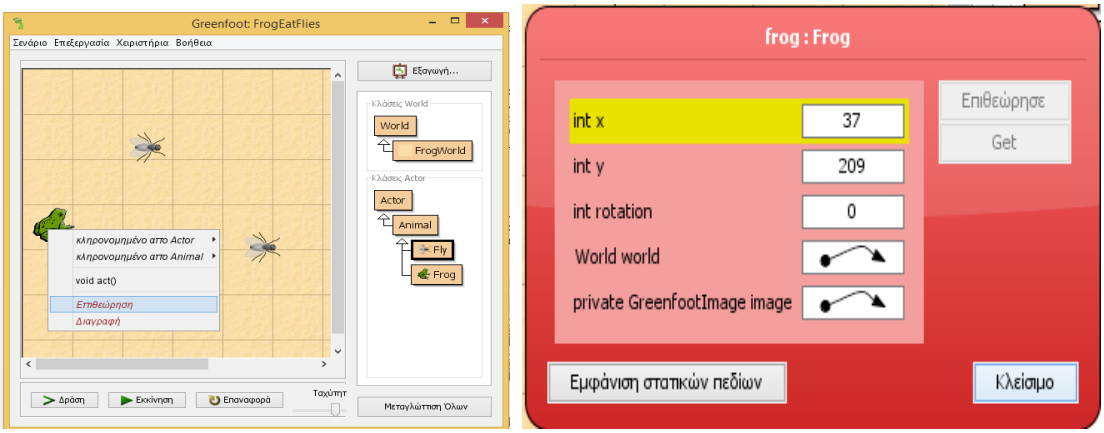
1.6. Ιδιότητες Αντικειμένων (attributes)

Όπως αναφέραμε παραπάνω, τα αντικείμενα στον προγραμματισμό έχουν διάφορα χαρακτηριστικά και συμπεριφορές, όπως έχουν και τα αντικείμενα στον πραγματικό κόσμο. Τα χαρακτηριστικά, που συχνά λέγονται και **ιδιότητες** (attributes), αποθηκεύουν πληροφορίες για τα αντικείμενα. Π.χ. οι ιδιότητες ενός αντικειμένου στο περιβάλλον Greenfoot είναι μεταβλητές που αποθηκεύουν πληροφορίες γι' αυτό, όπως τη θέση του στον Κόσμο, το μέγεθος του, το χρώμα του.

2η εργαστηριακή άσκηση:

Επιστρέφουμε στο σενάριο FrogEatFlies της προηγούμενης άσκησης όπου δημιουργήσαμε μερικά αντικείμενα των κλάσεων που έχουμε ορίσει. Μπορούμε να δούμε τις ιδιότητες των αντικειμένων αυτών κάνοντας δεξί κλικ επάνω τους και επιλέγοντας από το μενού που εμφανίζεται το ***Επιθεώρηση***. Η επιλογή αυτή μας δείχνει όλες τις πληροφορίες για τις μεταβλητές που συνθέτουν το αντικείμενο.

Κάνοντας δεξί κλικ πάνω στο αντικείμενο βάτραχος και επιλέγοντας το ***Επιθεώρηση*** βλέπουμε τα χαρακτηριστικά του. Κάθε χαρακτηριστικό καταλαμβάνει μια γραμμή και περιλαμβάνει τον τύπο του, το όνομα του και την τιμή του. Τα χαρακτηριστικά που βλέπουμε είναι: οι συντεταγμένες του x και y (int x=37 και int y=209) και ο προσανατολισμός του σχετικά με τον οριζόντιο άξονα σε μοίρες (int rotation=0). Στις δύο τελευταίες γραμμές υπάρχουν δύο ακόμα χαρακτηριστικά που είναι η εικόνα που έχει ως μορφή (GreenfootImage image) και ο Κόσμος που ανήκει (World world) (Εικόνα 1.6.1). Αυτά δεν έχουν κάποια τιμή, αλλά είναι αναφορές σε κάποια άλλα αντικείμενα. Θα μιλήσουμε γι' αυτού του είδους τα χαρακτηριστικά αργότερα.



Εικόνα 1.6.1 Επιθεώρηση ιδιοτήτων αντικειμένου

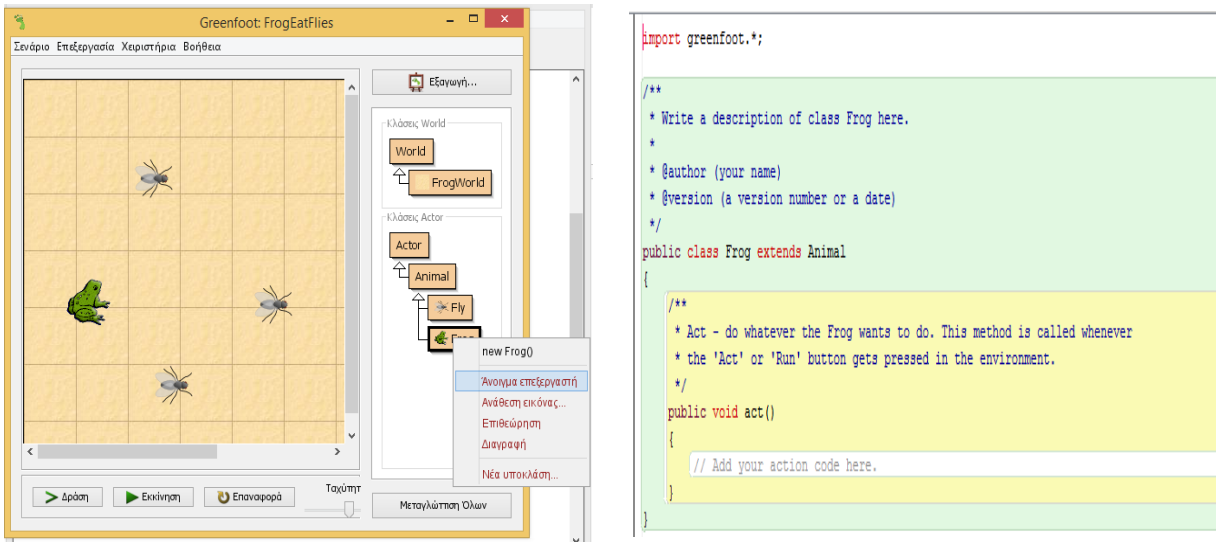
Κάνοντας δεξί κλικ οπουδήποτε πάνω στον Κόσμο μπορούμε να δούμε επίσης τις ιδιότητες του αντικειμένου FrogWorld. Είναι οι ίδιες μ' αυτές του αντικειμένου Frog; Αν διαφέρουν, μπορείτε να εξηγήσετε γιατί;

1.7. Ανάπτυξη απλών προγραμμάτων

3η εργαστηριακή άσκηση:

Το DNA δίνει στους ανθρώπους ορισμένα χαρακτηριστικά, όπως η εμφάνιση, η κινητικότητα και η επικοινωνία. Όπως το DNA, έτσι και ο πηγαίος κώδικας μιας κλάσης ορίζει τις δυνατότητες και τη συμπεριφορά των αντικειμένων της κλάσης.

Για να δούμε τον πηγαίο κώδικα για την κλάση Frog μπορούμε να κάνουμε δεξί κλικ πάνω στην κλάση και να επιλέξουμε ***Άνοιγμα επεξεργαστή***. Εμφανίζεται ο πηγαίος κώδικας της κλάσης γραμμένος στη γλώσσα java.



Εικόνα 1.7.1 Ο επεξεργαστής της κλάσης

Στο κίτρινο πλαίσιο (Εικόνα 1.7.1) βλέπουμε την ενέργεια-λειτουργία (μέθοδο) **act()** (τον κώδικα της δηλαδή σε Java). Αυτή η μέθοδος υπάρχει στον πηγαίο κώδικα όλων των αντικειμένων του Greenfoot και καλείται (εκτελείται) για όλα τα αντικείμενα του Κόσμου κάθε φορά που κάνουμε κλικ στο κουμπί εκτέλεσης '***Δράση***' ή '***Εκκίνηση***' (για τις μεθόδους θ' αναφερθούμε στη συνέχεια*).*

Μέσα στο σώμα της μεθόδου act(), που βρίσκεται ανάμεσα στα δύο άγκιστρα { }, γράφονται οι εντολές οι οποίες καθορίζουν τις δυνατότητες (συμπεριφορές), δηλαδή τις ενέργειες, που το αντικείμενο μπορεί να πραγματοποιήσει όταν εκτελεστεί η act().

Στον επεξεργαστή κώδικα μπορούμε να:

- Γράφουμε κώδικα για να προγραμματίζουμε τις ενέργειες που θα εκτελούν τα αντικείμενα των κλάσεων.

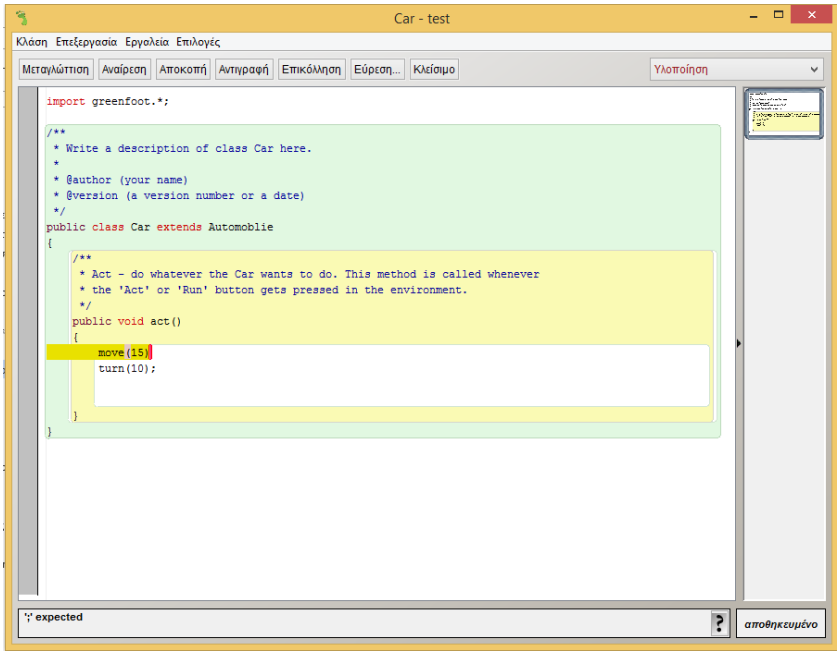
Τροποποιούμε τον κώδικα για ν' αλλάζουμε την συμπεριφορά ενός αντικειμένου

Ας γράψουμε τον παρακάτω κώδικα ανάμεσα στα δύο άγκιστρα { } της μεθόδου act() του αντικειμένου Frog:

```
move(15);
turn(10);
```

Κλείνουμε τον επεξεργαστή, πατάμε ***Μεταγλώττιση Όλων*** και δημιουργούμε ένα νέο αντικείμενο Frog. Πατάμε το κουμπί ***Εκκίνηση*** και βλέπουμε το βάτραχο να διαγράφει μικρούς κύκλους.

Κατά τη διαδικασία της μεταγλώττισης υπάρχει πιθανότητα να εντοπίσουμε κάποιο λάθος που έχουμε κάνει κατά τη συγγραφή του προγράμματος. Στην παρακάτω εικόνα έχουμε ξεχάσει το ερωτηματικό στο τέλος της εντολής move (Εικόνα 1.7.2).



Εικόνα 1.7.2 Συντακτικό λάθος της Java

Η ύπαρξη λαθών, κατά τη συγγραφή προγραμμάτων, αποτελεί σύνηθες φαινόμενο. Ορισμένα από αυτά μπορεί να εμποδίζουν τη μεταγλώττιση τους. Το λάθος που είδαμε παραπάνω λέγεται **συντακτικό** και ανιχνεύεται κατά τη μεταγλώττιση. Ο προγραμματιστής μπορεί να διαβάζει τα τυχόν μηνύματα λάθους που του εμφανίζει το περιβάλλον του Greenfoot, να τα διορθώνει και να μεταγλωττίζει ξανά το πρόγραμμα του. Ένας χαρακτήρας που λείπει ή έχει γραφεί λάθος, η παράλειψη π.χ. του συμβόλου του ερωτηματικού (;) στο τέλος μιας εντολής, μπορεί να προκαλέσει την αποτυχία εκτέλεσης του προγράμματος μας.

Μερικές χρήσιμες οδηγίες για την αποφυγή λαθών είναι οι ακόλουθες:

- Για κάθε αγκύλη αρχής { να υπάρχει η αντίστοιχη αγκύλη τέλους }
- Για κάθε αριστερή παρένθεση (να υπάρχει η αντίστοιχη δεξιά) που την κλείνει.
- Να τελειώνουν με ελληνικό ερωτηματικό (;) όλες οι εντολές του κώδικα.

Μερικές καλές πρακτικές που συνήθως ακολουθούνται στη συγγραφή κώδικα σε Java είναι:

- Τα ονόματα των κλάσεων ν' αρχίζουν με κεφαλαίο γράμμα π.χ. Frog, ενώ αν αποτελούνται από παραπάνω λέξεις το πρώτο γράμμα κάθε λέξης να γράφεται με κεφαλαίο π.χ. FrogWorld.
- Τα ονόματα των μεταβλητών να ξεκινούν με μικρό γράμμα π.χ. age.

1.8. Δραστηριότητες

Δραστηριότητα 1

Να σχεδιάσετε το διάγραμμα κλάσεων στο Greenfoot για τις παρακάτω κατηγορίες αριθμών: Ακέραιοι, Φυσικοί, Ρητοί, Άρρητοι, Πραγματικοί. Στη συνέχεια να μεταγλωττίσετε το πρόγραμμά σας και να δημιουργήσετε 2 αντικείμενα από κάθε κατηγορία.

Δραστηριότητα 2

Να κατεβάσετε από την διεύθυνση http://www.greenfoot.org/scenarios δυο σενάρια του Greenfoot της επιλογής σας, να τα ανοίξετε, να τα μεταγλωττίσετε και να τα εκτελέσετε.

Δραστηριότητα 3

Ταξινομήστε τις παρακάτω οντότητες σε μια ιεραρχία, χρησιμοποιώντας το μηχανισμό της κληρονομικότητας:

Φάλαινα, Ψάρι, Παθολόγος, Θηλαστικό, Φώκια, Ελάφι, Τσιπούρα, Άνθρωπος, Εργαζόμενος, Γιατρός, Δικηγόρος, Ζωντανοί οργανισμοί, Οφθαλμίατρος.

Κεφάλαιο 2

Βασικά στοιχεία της γλώσσας

2. Βασικά Στοιχεία της Γλώσσας

Εισαγωγή

Σε αυτή την ενότητα θα παρουσιάσουμε τους βασικούς τύπους των δεδομένων της Java, τους αντίστοιχους τελεστές (αριθμητικοί, συγκριτικοί, λογικοί), τη δομή επιλογής και τη δομή επανάληψης.

Η Java χαρακτηρίζεται από ένα αρκετά καλά οργανωμένο σύνολο εντολών κι ένα μέρος της επιτυχίας της οφείλεται στα διάφορα APIs (βιβλιοθήκες) που έρχονται μαζί με τη γλώσσα. Έτσι στις περισσότερες περιπτώσεις υπάρχουν πολλές από τις λειτουργίες που μπορεί να χρειαστεί κάποιος έτοιμες. Σε αυτή την ενότητα θα δείξουμε πώς μπορούμε να χρησιμοποιήσουμε μια βιβλιοθήκη για την επικοινωνία της εφαρμογής με τον χρήστη μέσω παραθύρων διαλόγου.

Η λογική της Java συνοψίζεται στην εξής ιδέα της ελάσσονος προσπάθειας:

Δεν επανεφευρίσκουμε τον τροχό. Όταν θέλουμε να αναπτύξουμε μια εφαρμογή ψάχνουμε πρώτα στις βιβλιοθήκες της γλώσσας μήπως αυτή η εφαρμογή ή τμήματά της έχουν ήδη υλοποιηθεί από άλλον προγραμματιστή.

Αν από την άλλη πλευρά αναπτύξουμε μια βιβλιοθήκη με τόσο εξειδικευμένες λειτουργίες που δεν έχουν αναπτυχθεί ακόμα, καλό θα ήταν να συνεισφέρουμε και εμείς στην κοινότητα των προγραμματιστών θέτοντάς την στην διαθεσιμότητα και άλλων προγραμματιστών.

Στόχοι

Οι μαθητές να μπορούν να:

- Χρησιμοποιούν τις δομές ελέγχου για την επίλυση προβλημάτων
- Χρησιμοποιούν διάφορα είδη παραθυρικών διαγνωστικών μηνυμάτων
- Βρίσκουν και να χρησιμοποιούν τις έτοιμες συναρτήσεις των προκαθορισμένων βιβλιοθηκών

Ενότητες κεφαλαίου

- Μεταβλητές, Τύποι, Τελεστές και Εκφράσεις
- Η δομή επιλογής if … else
- Οι δομές επανάληψης for και while
- Βασικές Συναρτήσεις
- Τεκμηρίωση Λογισμικού
- Διαγνωστικά Μηνύματα

2.1. Μεταβλητές, Τύποι, Τελεστές και Εκφράσεις

Η *Java* είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού. Τα πάντα στην Java είναι αντικείμενα κάποιων κλάσεων εκτός από κάποιες ελάχιστες εξαιρέσεις, όπως είναι οι βασικοί τύποι δεδομένων.

2.1.1. Βασικοί τύποι

Οι βασικοί πρωτογενείς **τύποι δεδομένων**, οι οποίοι δεν είναι αντικείμενα κάποια κλάσης, είναι οι παρακάτω:

Τύπος	Περιγραφή	Μέγεθος	Εύρος
byte	Byte	1 byte	-128 έως 127
short	Ακέραιος περιορισμένου εύρους	2 bytes	-32768 έως 32767
int	Ακέραιος	4 bytes	-2^{31} έως $2^{31}-1$
long	Εκτεταμένος Ακέραιος	8 bytes	-2^{63} έως $2^{63}-1$
float	Πραγματικός (κινητής υποδιαστολής)	4 bytes	$1.4x10^{-45}$ έως $3.4x10^{38}$
double	Πραγματικός διπλής ακρίβειας	8 bytes	$4.9x10^{-324}$ έως $1.8x10^{308}$
boolean	Λογική τιμή	1 bytes	true / false
char	Χαρακτήρας	2 bytes	οποιοδήποτε γράμμα ή ψηφίο ή άλλο σύμβολο του κώδικα *unicode*

Πίνακας 1 Οι βασικοί τύποι δεδομένων

Οι τύποι δεδομένων *byte, short, int, long* χρησιμοποιούνται ως ακέραιοι αριθμοί ανάλογα με το εύρος που μας ενδιαφέρει, ενώ οι τύποι *float, double* για χρήση αριθμών κινητής υποδιαστολής, δηλαδή δεκαδικών αριθμών. Ο τύπος boolean παίρνει μόνο δυο τιμές true / false (Αληθής / Ψευδής) ενώ ο τύπος char έχει μέγεθος 2 bytes γιατί έχει σχεδιαστεί να περιέχει χαρακτήρες Unicode (UTF-16).

2.1.2. Τελεστές

Στη Java ορίζονται οι παρακάτω τελεστές:

Τελεστής ανάθεσης (=), ο οποίος θέτει μια τιμή σε μια μεταβλητή, π.χ. a = 496; θέτει την τιμή 496 στη μεταβλητή a.

Αριθμητικοί Τελεστές			
Τελεστής	**Περιγραφή**	**Παράδειγμα**	**Αποτέλεσμα**
+	Πρόσθεση	18 + 10	28
–	Αφαίρεση	18 – 10	8
*	Πολλαπλασιασμός	18*10	180
/	Πηλίκο Διαίρεσης	18/10	1
		18.0/10	1.8
%	Υπόλοιπο ακέραιας Διαίρεσης	18 % 10	8

Πίνακας 2 Αριθμητικοί τελεστές

Στη Java η λειτουργία των τελεστών σε κάποιες περιπτώσεις εξαρτάται από τους τύπους δεδομένων που τους πλαισιώνουν. Ένα τέτοιο παράδειγμα είναι ο τελεστής (/) της διαίρεσης. Όταν

και οι δυο αριθμοί είναι ακέραιοι, τότε το αποτέλεσμα είναι ακέραιος, ενώ όταν ένας τουλάχιστον είναι πραγματικός το αποτέλεσμα είναι πραγματικός αριθμός.

Σχεσιακοί Τελεστές			
Τελεστής	**Περιγραφή**	**Παράδειγμα**	**Αποτέλεσμα**
==	Ισότητα	18 == 10	false
<	Μικρότερο	18 < 10	false
>	Μεγαλύτερο	18 > 10	true
<=	Μικρότερο ή ίσο	18 <= 10	false
>=	Μεγαλύτερο ή ίσο	18 >= 10	true
!=	Διαφορετικό	18 != 10	true

Πίνακας 3 Σχεσιακοί τελεστές

Λογικοί Τελεστές	
Τελεστής	**Περιγραφή**
!	όχι (not)
&&	και (and)
||	ή (or)
^	αποκλειστικό ή (xor)

Πίνακας 4 Λογικοί τελεστές

Έστω δυο λογικές μεταβλητές A και B, τύπου ***boolean*** στη Java. Παρακάτω δίνεται ο πίνακας αλήθειας για όλους τους λογικούς τελεστές :

A	**B**	**!A**	**A & B**	**A | B**	**A ^ B**
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	False	false

Πίνακας 5 Πίνακας αλήθειας λογικών τελεστών

Οι λογικοί τελεστές δεν υπολογίζουν όλη την λογική παράσταση αν αυτό δεν είναι απαραίτητο. Για παράδειγμα η παράσταση **false && οτιδήποτε**, είναι πάντα false. Δεν χρειάζεται να συνεχίσουμε τους υπολογισμούς. Αυτή η πρακτική είναι γνωστή ως ***short-circuit evaluation***.

2.1.3. Δήλωση και ορισμός μεταβλητών

Συχνά, τα αντικείμενα μας πρέπει να «θυμούνται» πληροφορίες. Στις γλώσσες προγραμματισμού αυτό γίνεται όταν αποθηκεύεται πληροφορία σε μια μεταβλητή.

Στην Java όπως και στη C++ και την Pascal, πρέπει να δηλώνουμε τον τύπο των μεταβλητών που θα χρησιμοποιήσουμε (σε αντίθεση με την Python όπου, όλα ορίζονται δυναμικά και δεν χρειάζεται να δηλώνουμε άμεσα τον τύπο κάθε μεταβλητής). Η δήλωση μιας μεταβλητής γίνεται συνήθως στην αρχή του προγράμματος, μπορεί όμως να γίνει και σε άλλες θέσεις μέσα στο πρόγραμμα, αρκεί φυσικά να είναι πριν από την πρώτη εντολή στην οποία χρησιμοποιείται η μεταβλητή (δηλαδή, πριν χρησιμοποιήσουμε οποιαδήποτε μεταβλητή πρέπει πρώτα να δηλωθεί).

Ακολουθούν διάφορες δηλώσεις μεταβλητών:

```
int number;   // Η μεταβλητή number είναι ακέραιος.

int counter = 6;   // Δήλωση και ανάθεση τιμής στην ίδια εντολή
```

```
boolean areEqual = (number == counter);   // Μια λογική μεταβλητή
```

Τα δεδομένα ενός αντικειμένου που ονομάζουμε πεδία (attributes), αποθηκεύονται σε μεταβλητές οι οποίες είναι μέρη του αντικειμένου και είναι γνωστές ως μεταβλητές στιγμιοτύπου, (instances variables).

Η μεταβλητή αντικειμένου ανήκει στο αντικείμενο (στιγμιότυπο κλάσης) και καταλαμβάνει ένα μέρος στη μνήμη. Δηλώνεται γράφοντας συνήθως στην αρχή τη λέξη **private** (θα εξηγηθεί αργότερα) ακολουθούμενη από το τύπο της μεταβλητής και το όνομα της.

Ο τύπος της μεταβλητής ορίζει το είδος της πληροφορίας που θα αποθηκευτεί στη μνήμη. Το όνομα της μεταβλητής πρέπει να είναι τέτοιο που να περιγράφει το σκοπό για τον οποίο θα χρησιμοποιηθεί η μεταβλητή.

Γενικά, η σύνταξη για τη δήλωση μιας μεταβλητής αντικειμένου (instance variable) ακολουθεί τον κανόνα:

«λέξη public ή private» «τύπος μεταβλητής» «όνομα μεταβλητής».

Τέλος, πρέπει να θυμίσουμε ότι το όνομα μιας μεταβλητής συνηθίζεται να ξεκινάει με μικρό γράμμα, σε αντίθεση με το όνομα μιας κλάσης που συνηθίζεται να ξεκινάει με κεφαλαίο. Είναι καλό τα ονόματα που δίνουμε σε μεταβλητές και κλάσεις να είναι περιγραφικά της έννοιας που θέλουν να αναπαραστήσουν. Μη δίνετε ελληνικά ονόματα στις μεταβλητές σας. Σε περίπτωση που θέλετε να χρησιμοποιήσετε ελληνικά ονόματα για τις μεταβλητές σας είναι προτιμότερο να χρησιμοποιήσετε greeklish. Τα ονόματα μεταβλητών δεν αρχίζουν ποτέ με αριθμό και δεν μπορούν να περιέχουν κενά.

2.2. Η δομή επιλογής if … else

Η δομή επιλογής ή εντολή διακλάδωσης στην Java δεν διαφέρει ιδιαίτερα από τις άλλες γλώσσες προγραμματισμού. Η εντολή if χρησιμοποιείται όταν θέλουμε να εκτελέσουμε ορισμένες εντολές ανάλογα με την ικανοποίηση ή όχι κάποιας συνθήκης. Η σύνταξή της είναι η εξής:

```
if (συνθήκη)
{
   Εντολές
}
else
{
   Εντολές
}
```

Οι εντολές που περιέχονται ανάμεσα στα άγκιστρα μεταξύ του if και του else, εκτελούνται μόνο όταν η συνθήκη ισχύει (είναι αληθής), ενώ οι εντολές που περιέχονται στα άγκιστρα μετά το else, εκτελούνται μόνο όταν δεν ισχύει η συνθήκη (είναι ψευδής).

Όταν έχουμε πολλές περιπτώσεις μπορούμε να συνδυάσουμε πολλές εντολές if μεταξύ τους όπως φαίνεται παρακάτω:

```
if (number < 0)
    digits = -1;
else if (number < 10)
    digits = 1;
else if (number < 100)
    digits = 2;
else if (number < 1000)
    digits = 3;
else
    digits = 4;
```

Σε περίπτωση που έχουμε μόνο μια εντολή, δεν χρειάζονται τα άγκιστρα (δεν είναι λάθος όμως αν τα βάλουμε). Επίσης μπορούμε να έχουμε εμφωλευμένες δομές επιλογής όπως φαίνεται στο παρακάτω παράδειγμα:

```
if (a != 0) {
    if (b > 0) {
        if (c < 0) {
            f = 1;
        }
        else {
            f = 2;
            if (b == 100)
                g = 9;
        }
    }
}
```

2.3. Οι δομές επανάληψης *for* και *while*

Στην Java έχουμε τις δομές επανάληψης for και while, οι οποίες έχουν την εξής σύνταξη:

<αρχικοποίηση>
while (συνθήκη) {
  <εντολές>
  <εντολές_βήματος>
}
	for (αρχικοποίηση; συνθήκη; εντολές_βήματος) {
	  <εντολές>
	}

Οι παραπάνω δομές είναι απολύτως ισοδύναμες. Στο παρακάτω παράδειγμα υπολογίζουμε το άθροισμα όλων των αριθμών από το 1 έως και το 1000. Δίπλα δίνουμε και τον αντίστοιχο αλγόριθμο στην ψευδογλώσσα του μαθήματος "Εισαγωγή στην Επιστήμη των Υπολογιστών" της Β' Λυκείου:

sum = 0;
for (i = 1; i <= 1000; i=i+1) {
  sum = sum + i;
}
	sum ← 0
	i ← 1
	Όσο i <= 1000 **Επανάλαβε**
	  sum = sum + i;
	  i = i + 1;
	Τέλος Επανάληψης

Στην Java αντί να γράφουμε την εντολή i = i + 1 έχει επικρατήσει ο γνωστός τελεστής μεταύξησης **i++** , ομοίως υπάρχει και αντίστοιχος τελεστής μείωσης κατά 1, **i--** .

Εδώ θα πρέπει να εξηγήσουμε τη διαφορά που παρουσιάζει στο αποτέλεσμα η χρήση των διπλών συμβόλων αύξησης/μείωσης αριθμών (++ / - -) όταν τοποθετούνται πριν από τη μεταβλητή και μετά από τη μεταβλητή. Όταν τοποθετούνται πριν από τη μεταβλητή γίνεται πρώτα η αύξηση (ή μείωση αντίστοιχα) της τιμής της και μετά εκτελείται η εντολή που την περιέχει, λαμβάνοντας υπόψη τη νέα της τιμή. Αντίθετα, όταν τοποθετούνται μετά από τη μεταβλητή εκτελείται πρώτα η εντολή που την περιέχει και μετά γίνεται η αύξηση (ή μείωση αντίστοιχα) της τιμής της.

Ας δούμε ένα πιο συγκεκριμένο παράδειγμα για να καταλάβουμε τη διαφορά τους:

```
int var=3;     //Ορίζουμε μια μεταβλητή var με αρχική τιμή 3
System.out.println(++var);  // αύξησε την var κατά 1 και εμφάνισε την τιμή της var
```

```
System.out.println(var);     //  εμφάνισε την τιμή της var
```

Το αποτέλεσμα που θα εμφανιστεί στην οθόνη μας είναι ο αριθμός 4 δύο φορές. Αυτό οφείλεται στο γεγονός ότι στην εντολή System.out.println(++var); η μεταβλητή var πρώτα αυξάνει την τιμή της από 3 σε 4 και μετά εκτελείται η εντολή εκτύπωσης.

Αντίθετα, με τη χρήση του τελεστή μετά τη μεταβλητή θα έχουμε:

```
int var=3;
System.out.println(var++);  // εμφάνισε και μετά αύξησε την var κατά 1
System.out.println(var);     // εμφάνισε την τιμή της var
```

Το αποτέλεσμα που θα εμφανιστεί στην οθόνη μας είναι ο αριθμός 3 και μετά ο αριθμός 4.

Αυτό οφείλεται στο γεγονός ότι, στην εντολή System.out.println(var++); πρώτα εκτελείται η εντολή εκτύπωσης και εμφανίζεται στην οθόνη η τρέχουσα τιμή της μεταβλητής var, δηλαδή ο αριθμός 3 και μετά η μεταβλητή var αυξάνει την τιμή της από 3 σε 4. Στην αμέσως επόμενη γραμμή ζητάμε να δούμε την τρέχουσα τιμή της μεταβλητής που φυσικά τώρα είναι ο αριθμός 4.

2.4. Βασικές Συναρτήσεις - Μέθοδοι

Οι συναρτήσεις, είναι επαναχρησιμοποιήσιμα τμήματα κώδικα, στα οποία έχει δοθεί ένα όνομα, με απλή αναφορά του οποίου και εκτελούνται. Η λειτουργία αυτή είναι γνωστή ως **κλήση συνάρτησης**. Οι συναρτήσεις στην Java ανήκουν πάντα σε μια κλάση και λέγονται **μέθοδοι** της κλάσης (στην ευρύτερη έννοια της μεθόδου συμπεριλαμβάνονται και τα γνωστά μας υποπρογράμματα). Στο προηγούμενο κεφάλαιο είδαμε ότι η συμπεριφορά των αντικειμένων καθορίζεται μέσα στην κλάση που ανήκουν. Οι μέθοδοι είναι τα τμήματα κώδικα που καθορίζουν αυτή τη συμπεριφορά. Με άλλα λόγια, οι μέθοδοι είναι οι ενέργειες που μπορεί να εκτελεί ένα αντικείμενο. Ας πάρουμε το παράδειγμα ενός αντικειμένου που δημιουργήσαμε από την κλάση Αυτοκινούμενο όχημα. Οι μέθοδοι που συνδέονται με αυτή την κλάση είναι π.χ. η Εκκίνηση(), η Πέδηση(), η Μεταβολή ταχύτητας().

Η τυπική βιβλιοθήκη κλάσεων της Java παρέχει ένα σύνολο APIs (Application Programming Interface) κλάσεων, οι οποίες είναι διαθέσιμες σε κάθε πρόγραμμα Java. Η βιβλιοθήκη κλάσεων οργανώνεται σε πακέτα. Στη JDK version 1.8 υπάρχουν περίπου 1.000 κλάσεις σε 60 πακέτα (packages). Ένα **πακέτο** είναι μια συλλογή από κλάσεις, που σχετίζονται μεταξύ τους και κάθε κλάση περιέχει αρκετές μεθόδους. Τα ονόματα των πακέτων απαρτίζονται από μια ή περισσότερες λέξεις χωρισμένες με τελείες.

Παρακάτω δίνουμε κάποια από τα πακέτα που θα χρησιμοποιήσουμε:

java.lang	Υποστήριξη των βασικών χαρακτηριστικών της γλώσσας.
java.util	Χρήσιμες εργασίες σε δεδομένα.
java.sql	Java Data Base Connectivity (JDBC) για διαχείριση βάσης δεδομένων.

Το πακέτο java.lang περιέχει τις βασικές κλάσεις της γλώσσας (Object, String, System) και κάποιες ειδικές κλάσεις για αναπαράσταση των βασικών τύπων δεδομένων με τη μορφή αντικειμένων (Integer, Long, Double). Οι κλάσεις αυτές είναι πάντα διαθέσιμες στα προγράμματά μας, καθώς το πακέτο java.lang φορτώνεται αυτόματα. Για να χρησιμοποιήσουμε τις κλάσεις από κάποιο άλλο πακέτο πρέπει να τις ενσωματώσουμε στον κώδικά μας χρησιμοποιώντας τη δεσμευμένη λέξη **import**. Ακολουθεί το όνομα του πακέτου και στη συνέχεια το όνομα της κλάσης. Όταν θέλουμε να εισάγουμε όλες τις κλάσεις ενός πακέτου βάζουμε *,

για παράδειγμα η εντολή : **import** acm.program.*;

ενημερώνει τον μεταγλωττιστή ότι θα χρησιμοποιήσουμε όλες τις κλάσεις που βρίσκονται στο πακέτο acm.program.

Η δήλωση import γίνεται στην αρχή του αρχείου της κλάσης πριν ξεκινήσει ο ορισμός της. Αν απαιτείται η εισαγωγή περισσοτέρων πακέτων, μπορούμε να χρησιμοποιήσουμε περισσότερες δηλώσεις import, πάντα στην αρχή του αρχείου.

Μια κλάση που θα χρησιμοποιούμε συχνά είναι η Math, του πακέτου java.lang, η οποία περιέχει όλες τις μαθηματικές μεθόδους που θα χρειαστούμε.

Παρακάτω δίνουμε μερικά παραδείγματα χρήσης μεθόδων της κλάσης Math.

```
x = Math.max(a, b) ;        // Επιστρέφει στην x το μέγιστο των a, b
root = Math.sqrt(x) ;       // Επιστρέφει την τετραγωνική ρίζα του x
```

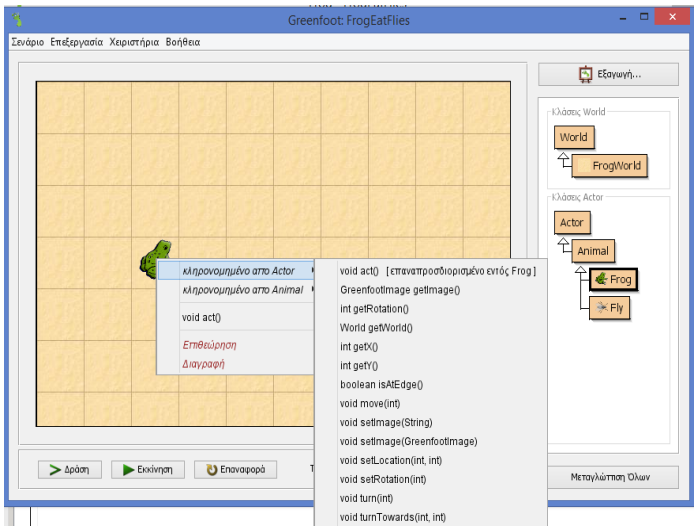
Πολύ συχνά χρησιμοποιούμε τη **μέθοδο println**, η οποία εμφανίζει στην οθόνη ένα μήνυμα. Η println είναι μια μέθοδος του **αντικειμένου out** που αναπαριστά το ρεύμα εξόδου, το οποίο συνήθως κατευθύνεται στην οθόνη. Το αντικείμενο out ανήκει στην **κλάση System**. Έτσι αν θέλουμε να εμφανίσουμε το μήνυμα : *Καλημέρα γαλαξία* γράφουμε:

```
System.out.println("Καλημέρα γαλαξία");
```

Υπάρχει μεγάλο πλήθος βιβλιοθηκών της java που περιέχουν πάρα πολλές μεθόδους κλάσεων. Αυτό είναι ένα από τα βασικά πλεονεκτήματα της Java, καθώς μπορούμε να υλοποιήσουμε πολύπλοκες εφαρμογές, χρησιμοποιώντας έτοιμο κώδικα που αναζητούμε στην πληθώρα των βιβλιοθηκών της, γλυτώνοντας την εκ νέου συγγραφή του. Δεν είμαστε υποχρεωμένοι και ούτε έχει νόημα να γνωρίζουμε όλες τις κλάσεις και τις μεθόδους ενός πακέτου. Συνήθως, όπως θα δούμε στη συνέχεια, αναζητούμε τη μέθοδο που ψάχνουμε στην **τεκμηρίωση** της κλάσης της.

4η εργαστηριακή άσκηση:

Από το περιβάλλον του Greenfoot, ανοίγουμε το σενάριο FrogEatFlies. Αφού δημιουργήσουμε μερικά αντικείμενα των κλάσεων, που έχουμε ορίσει ως τώρα, μπορούμε να τους δώσουμε εντολές για να εκτελέσουν κάποιες ενέργειες. Αρκεί να κάνουμε δεξί κλικ επάνω τους, όπως φαίνεται στην Εικόνα 2.4.1 για το αντικείμενο της κλάσης Frog. Στο μενού που εμφανίζεται βλέπουμε την επιλογή «κληρονομημένο από Actor» και αν την επιλέξουμε με το ποντίκι εμφανίζονται όλες οι μέθοδοι που έχουν κληρονομηθεί από την κλάση Actor. Η κλάση Frog είναι υποκλάση της Actor και σύμφωνα με τον ορισμό της κληρονομικότητας που δώσαμε στο προηγούμενο κεφάλαιο, κληρονομεί όλες τις μεθόδους της.



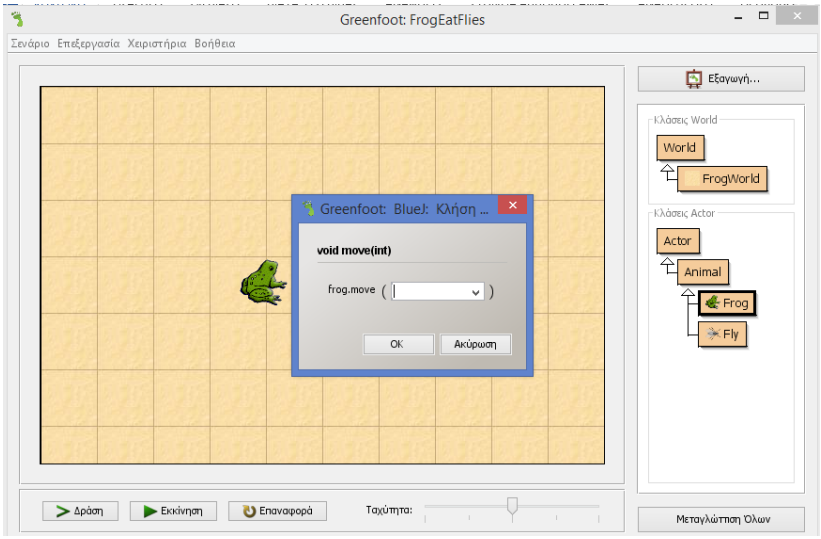
Εικόνα 2.4.1 Οι μέθοδοι του αντικειμένου Frog

Μερικές από τις μεθόδους, που κληρονομεί το αντικείμενο Frog δίνουν, όταν εκτελούνται, εντολή στο αντικείμενο να κάνει κάποια ενέργεια, όπως βλέπουμε στον Πίνακα που ακολουθεί.

move(int)	Κινεί το αντικείμενο προς την κατεύθυνση που βρίσκεται κατά συγκεκριμένα βήματα
turn(int)	Στρίβει το αντικείμενο κατά συγκεκριμένες μοίρες
act()	Το αντικείμενο κάνει μια ενέργεια
setLocation(int, int)	Ορίζει νέα θέση του αντικειμένου
setRotation(int)	Ορίζει την περιστροφή του αντικειμένου

Πίνακας 6 Μέθοδοι που δίνουν εντολή για να ενεργήσει το αντικείμενο

Παρατηρούμε ότι κάποιες μέθοδοι έχουν τη λέξη int μεταξύ των παρενθέσεων π.χ. η μέθοδος move(int). Η λέξη int μέσα στην παρένθεση ονομάζεται **παράμετρος** και σημαίνει ότι θα πρέπει να ορίσουμε κάποια τιμή, όταν επικαλούμαστε αυτή τη μέθοδο. Ο όρος 'int' υποδηλώνει ότι ένας ακέραιος αριθμός αναμένεται να δοθεί που θα προσδιορίσει πόσα βήματα θα κινηθεί ο βάτραχος όταν επικαλεστούμε τη συγκεκριμένη μέθοδο. Όταν λοιπόν την επιλέξουμε θα εμφανιστεί ένα παράθυρο διαλόγου που αναμένει να εισάγουμε μια τιμή για αυτήν την παράμετρο (Εικόνα 2.4.2). Πληκτρολογήστε έναν αριθμό (π.χ.: 20) και πατήστε Ok. Παρατηρούμε ότι ο βάτραχος θα κινηθεί 20 βήματα.



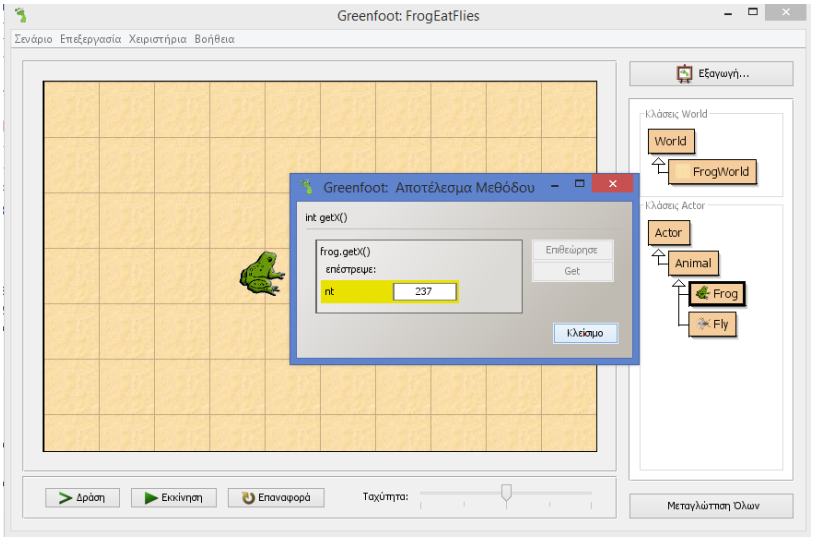
Εικόνα 2.4.2 Κλήση μεθόδου

Μερικές από τις μεθόδους, που κληρονομεί το αντικείμενο Frog δίνουν, όταν εκτελούνται, πληροφορίες για το αντικείμενο. Αυτές είναι οι:

getRotation()	Δίνει την τρέχουσα περιστροφή του αντικειμένου
getWorld()	Δίνει το όνομα του Κόσμου που βρίσκεται το αντικείμενο
getX()	Δίνει την συντεταγμένη χ της τρέχουσας θέσης του αντικειμένου
getY()	Δίνει την συντεταγμένη y της τρέχουσας θέσης του αντικειμένου

Πίνακας 7 Μέθοδοι που δίνουν πληροφορίες για το αντικείμενο

Στις μεθόδους αυτές, που δεν δέχονται παραμέτρους, η παρένθεση δεν περιέχει τίποτα. Όταν επιλέξουμε μια από αυτές θα εμφανιστεί στην οθόνη μας ένα παράθυρο διαλόγου που μας δίνει μια συγκεκριμένη πληροφορία για το αντικείμενο. Π.χ. αν επιλέξουμε τη μέθοδο getX() θα δούμε την τρέχουσα θέση του αντικειμένου στον άξονα του x (Εικόνα 2.4.3).



Εικόνα 2.4.3 Αποτέλεσμα μεθόδου

Εκτελέστε την μέθοδο setLocation(int, int) στο αντικείμενο Frog δίνοντας στο παράθυρο διαλόγου που εμφανίζεται τις συντεταγμένες των αξόνων x και y της νέας θέσης που επιθυμείτε να μετακινηθεί το αντικείμενο. Κατόπιν εκτελέστε τη μέθοδο getX() και επιβεβαιώστε ότι η τιμή που θα εμφανιστεί είναι η συντεταγμένη του άξονα x που δώσατε στην προηγούμενη μέθοδο setLocation. Δοκιμάστε την μέθοδο setLocation με διαφορετικές τιμές. Τι συμβαίνει αν πληκτρολογείτε στις παραμέτρους μια λέξη η έναν δεκαδικό αριθμό;

2.5. Τεκμηρίωση Λογισμικού

Η τεκμηρίωση μιας κλάσης είναι πολύ σημαντική γιατί αποτελεί την εικόνα της προς τον έξω κόσμο. Όποιος θέλει να χρησιμοποιήσει μεθόδους της κλάσης διαβάζει την τεκμηρίωση (***Documentation***) για να πληροφορηθεί σχετικά με τη λειτουργία των μεθόδων και τον τρόπο κλήσης τους. Η τεκμηρίωση των βιβλιοθηκών που έρχονται μαζί με τη γλώσσα Java είναι διαθέσιμη μέσω του παγκόσμιου ιστού. Είναι επίσης ενσωματωμένη μέσα στα περισσότερα ολοκληρωμένα προγραμματιστικά περιβάλλοντα (IDEs).

5η εργαστηριακή άσκηση:

Ας γυρίσουμε πίσω στο σενάριο μας. Με δεξί κλικ πάνω στην κλάση Actor επιλέγουμε ***Open Documentation*** για να μας ανοίξει στο φυλλομετρητή, την τεκμηρίωση της κλάσης Actor (Εικόνα 2.5.1).

Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS

SUMMARY: NESTED | FIELD | CONSTR | METHOD

greenfoot

Class Actor

```
java.lang.Object
  └ greenfoot.Actor
```

```
public abstract class Actor
extends java.lang.Object
```

An Actor is an object that exists in the Greenfoot world. Every Actor has a location in the world, and an appearance (that is: an icon).

An Actor is not normally instantiated, but instead used as a superclass to more specific objects in the world. Every object that is intended to appe

One of the most important aspects of this class is the 'act' method. This method is called when the 'Act' or 'Run' buttons are activated in the Green

Version:
2.5
Author:
Poul Henriksen

Constructor Summary

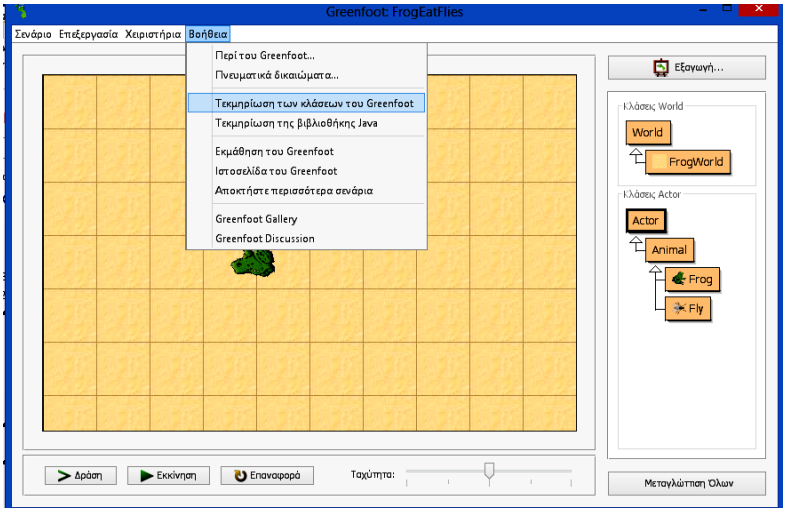
Actor()	Construct an Actor.

Method Summary

void	act() The act method is called by the greenfoot framework to give actors a chance to perform some action.
protected void	addedToWorld(World world) This method is called by the Greenfoot system when this actor has been inserted into the world.
protected java.util.List	getIntersectingObjects(java.lang.Class cls) Return all the objects that intersect this object.
protected java.util.List	getNeighbours(int distance, boolean diagonal, java.lang.Class cls) Return the neighbours to this object within a given distance.
protected java.util.List	getObjectsAtOffset(int dx, int dy, java.lang.Class cls) Return all objects that intersect the center of the given location (relative to this object's location).
protected java.util.List	getObjectsInRange(int radius, java.lang.Class cls) Return all objects within range 'radius' around this object.
protected Actor	getOneIntersectingObject(java.lang.Class cls) Return an object that intersects this object.
protected Actor	getOneObjectAtOffset(int dx, int dy, java.lang.Class cls) Return one object that is located at the specified cell (relative to this objects location).
int	getRotation() Return the current rotation of this actor.
World	getWorld() Return the world that this actor lives in.
int	getX() Return the x-coordinate of the actor's current location.
int	getY() Return the y-coordinate of the object's current location.
protected boolean	intersects(Actor other) Check whether this object intersects with another given object.
boolean	isAtEdge() Detect whether the actor has reached the edge of the world.
protected boolean	isTouching(java.lang.Class cls) Checks whether this actor is touching any other objects of the given class.
void	move(int distance) Move this actor the specified distance in the direction it is currently facing.
protected void	removeTouching(java.lang.Class cls) Removes one object of the given class that this actor is currently touching (if any exist).

Εικόνα 2.5.1 Τεκμηρίωση της κλάσης Actor

Για να δούμε την τεκμηρίωση όλων των κλάσεων του Greenfoot πηγαίνουμε στο μενού Βοήθεια και επιλέγουμε ***Τεκμηρίωση των κλάσεων του Greenfoot***, όπως φαίνεται στην Εικόνα 2.5.2.



Εικόνα 2.5.2 Το μενού Βοήθεια

Στην ιστοσελίδα που ανοίγει εμφανίζονται οι κλάσεις του πακέτου greenfoot με μια μικρή περίληψη του τι κάνει η κάθε μια από αυτές, όπως φαίνεται στην Εικόνα 2.5.3.

Class Summary	
Actor	Ο ηθοποιός (Actor) είναι ένα αντικείμενο το οποίο υπάρχει μέσα στον κόσμο του Greenfoot.
Greenfoot	Αυτή η κλάση είναι γενικής χρήσης και παρέχει μεθόδους για τον έλεγχο της εξομοίωσης και την αλληλεπίδραση με το σύστημα.
GreenfootImage	Μία εικόνα που θα εμφανίζεται στην οθόνη.
MouseInfo	Αυτή η κλάση περιέχει πληροφορίες για την κατάσταση του ποντικιού.
World	World είναι ο κόσμος μέσα στον οποίο ζουν οι ηθοποιοί (Actors).

Εικόνα 2.5.3 Κλάσεις του πακέτου greenfoot

Για να ρίξουμε μια αναλυτικότερη ματιά σε κάποια από τις κλάσεις, πατάμε με το ποντίκι πάνω στο όνομα της κλάσης και θα οδηγηθούμε στην ιστοσελίδα της, όπου θα δούμε ποιες μεθόδους παρέχει.

Περιηγηθείτε και στις πέντε κλάσεις του πακέτου greenfoot και προσπαθήστε να καταλάβετε τι λειτουργίες (μεθόδους) υποστηρίζει η κάθε μια.

Από το μενού Βοήθεια μπορούμε να δούμε επίσης, την τεκμηρίωση της βιβλιοθήκης Java που περιλαμβάνει όλα τα πακέτα και τις κλάσεις της βιβλιοθήκης.

2.5.1. Σχόλια

Ένα σημαντικό στοιχείο που πρέπει ν' αναφερθεί είναι η δυνατότητα να προσθέτουμε σχόλια στο πρόγραμμα μας.

Τα σχόλια χρησιμοποιούνται για να:

- Τεκμηριώνουμε: τον σκοπό και τους στόχους του προγράμματος μας, τον συγγραφέα του προγράμματος, τις εκδόσεις του κώδικά μας (revision history) κτλ.
- Περιγράφουμε: πεδία, μεθόδους, κλάσεις και κατασκευαστές.

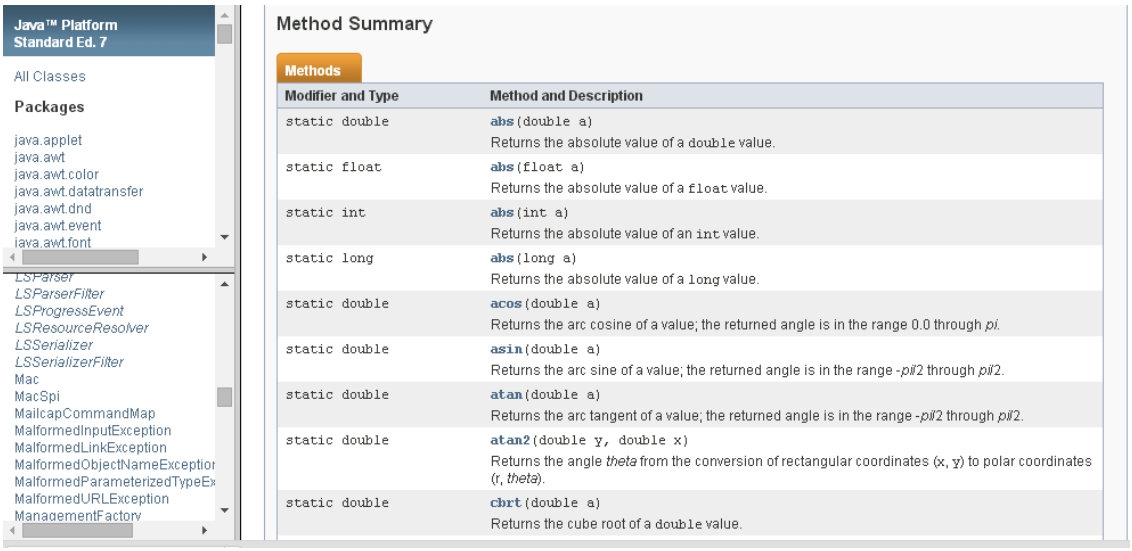
Τα σχόλια που γράφουμε στον πηγαίο κώδικα δεν εμφανίζονται στην εκτέλεση του προγράμματος μας.

Ένα σχόλιο πολλών γραμμών γράφεται ανάμεσα στα σύμβολα /* και */.

Ένα σχόλιο μιας γραμμής γράφεται μετά από το σύμβολο // μέχρι το τέλος της γραμμής.

Η δημιουργία τεκμηρίωσης για τις δικές μας κλάσεις στην Java είναι πολύ απλή και γίνεται εντελώς αυτοματοποιημένα, με χρήση του εργαλείου **javadoc** της γλώσσας. Το javadoc είναι ένα πρόγραμμα για αυτόματη κατασκευή τεκμηρίωσης σε μορφή HTM. Είναι αρκετά χρήσιμο στην κατασκευή βοηθημάτων και τεχνικών αναφορών για εφαρμογές οποιουδήποτε μεγέθους. Παρέχεται ως τμήμα του καθιερωμένου Πακέτου Ανάπτυξης (Development Kit), διατρέχει τον κώδικα ενός πακέτου και δημιουργεί τεκμηρίωση για κάθε μια από τις κλάσεις που αυτό περιέχει.

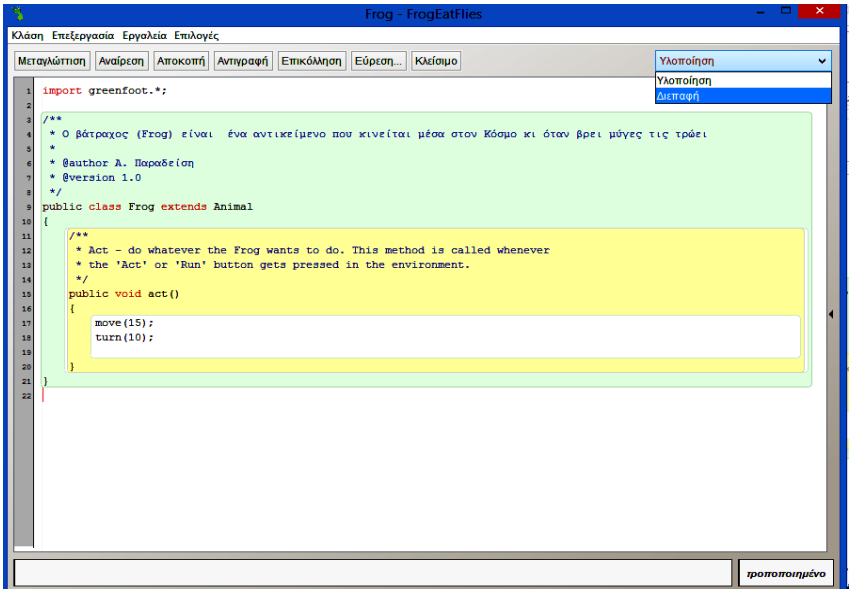
Παρακάτω στην Εικόνα 2.5.4, φαίνεται η τεκμηρίωση της κλάσης Math μέσα από τον παγκόσμιο ιστό, όπως έχει ανακτηθεί από την διεύθυνση http://docs.oracle.com/javase/7/docs/api/



Εικόνα 2.5.4 Τεκμηρίωση κλάσης Math

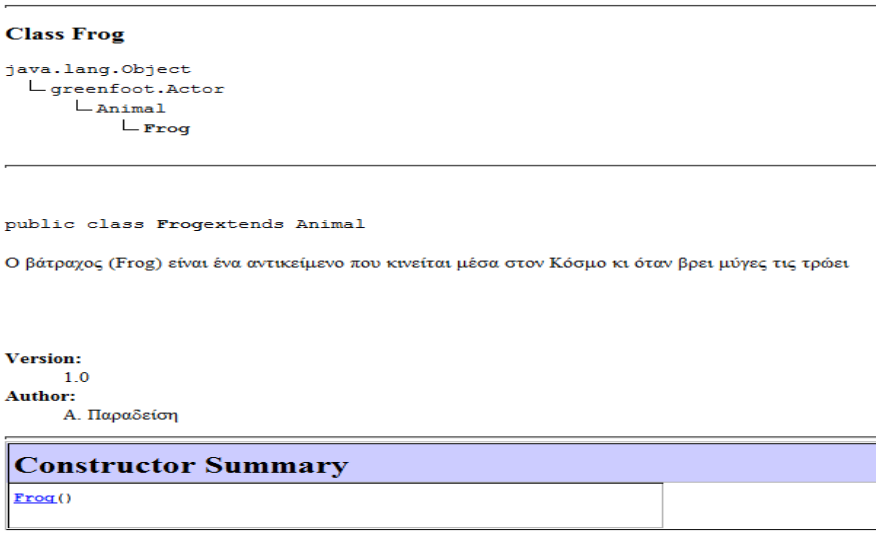
6η εργαστηριακή άσκηση:

Ας γυρίσουμε στο σενάριο μας κι ας προσθέσουμε κάποια σχόλια μέσα στις κλάσεις που έχουμε δημιουργήσει. Στην κλάση Frog ανοίγουμε τον επεξεργαστή και στο πάνω μέρος του αρχείου, στην περιοχή των σχολίων, γράφουμε μια μικρή περιγραφή της κλάσης όπως: «Ο βάτραχος (Frog) είναι ένα αντικείμενο που κινείται μέσα στον Κόσμο κι όταν βρει μύγες τις τρώει». Παρακάτω δίπλα στη λέξη @author θα γράψετε το όνομα σας και δίπλα στη λέξη @version θα γράψετε τον αριθμό 1.0 που δηλώνει ότι αυτή είναι η πρώτη έκδοση του σεναρίου σας (Εικόνα 2.5.5)



Εικόνα 2.5.5 Εισαγωγή σχολίων

Τα σχόλια δημιουργήθηκαν και τώρα θα εμφανίζονται στην τεκμηρίωση της κλάσης Frog. Μπορούμε να δούμε την τεκμηρίωση της κλάσης επιλέγοντας τη **Διεπαφή** από το αναδυόμενο μενού στην πάνω δεξιά γωνία του επεξεργαστή (Εικόνα 2.5.5).



Class Frog

java.lang.Object
└ greenfoot.Actor
└ Animal
└ Frog

public class Frogextends Animal

Ο βάτραχος (Frog) είναι ένα αντικείμενο που κινείται μέσα στον Κόσμο κι όταν βρει μύγες τις τρώει

Version:
1.0
Author:
A. Παραδείση

Constructor Summary

Frog()

Εικόνα 2.5.6 Τεκμηρίωση της κλάσης Frog

2.6. Διαγνωστικά Μηνύματα

Όλες οι γλώσσες προγραμματισμού, που παρέχουν τη δυνατότητα δημιουργίας παραθυρικών εφαρμογών, διαθέτουν μια σειρά διαλόγων που ονομάζονται **κοινοί διάλογοι** (**common dialogs**) και διευκολύνουν την εισαγωγή δεδομένων από τον χρήστη, την προβολή μηνυμάτων κλπ. Η Java παρέχει το πακέτο **javax.swing** για τη δημιουργία γραφικού περιβάλλοντος διεπαφής (GUI). Η κλάση **JOptionPane** του πακέτου διευκολύνει την χρήση παραθύρων διαλόγου για διαγνωστικά μηνύματα, είσοδο και έξοδο δεδομένων. Για μια ολοκληρωμένη γνώση της κλάσης JOptionPane και τις μεθόδους της, μπορείτε να ανατρέξετε στο documentation της java στην διεύθυνση: http://docs.oracle.com/javase/7/docs/api/.

Στην κλάση JOptionPane υπάρχουν τέσσερεις τύποι παραθύρων διαλόγου:

- Επιβεβαίωσης (ConfirmDialog)

- Εισόδου (InputDialog)
- Μηνύματος (MessageDialog)
- Επιλογής (OptionDialog)

7η εργαστηριακή άσκηση:

Ανοίγουμε το σενάριο FrogEatFlies στο Greenfoot. Αυτό που θέλουμε να κάνουμε είναι να προσθέσουμε στο πρόγραμμα μας τη λειτουργία να δείχνει ένα καλωσόρισμα στο χρήστη με παραθυρικό τρόπο, όταν αυτός πατήσει με το ποντίκι του πάνω στο αντικείμενο Frog.

Ανοίγουμε τον επεξεργαστή της κλάσης Frog για να γράψουμε τον κώδικα που χρειάζεται.

Η πρώτη μέθοδος της κλάσης JOptionPane που θα χρησιμοποιήσουμε είναι η showInputDialog που κάνει δυνατή την εισαγωγή δεδομένων, ζητώντας παράλληλα τα δεδομένα αυτά από τον χρήστη. Η δεύτερη μέθοδος της κλάσης JOptionPane που θα χρησιμοποιήσουμε είναι η showMessageDialog που εμφανίζει ένα μήνυμα με το καλωσόρισμα στο συγκεκριμένο χρήστη.

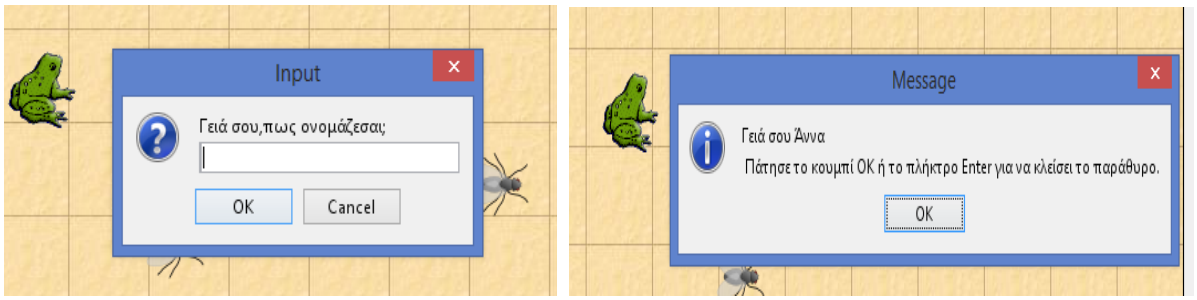
Στην αρχή του αρχείου της κλάσης θα γράψουμε την εντολή: import javax.swing.JOptionPane;

για να ενημερώσουμε τον μεταγλωττιστή ότι θα χρησιμοποιήσουμε την κλάση JOptionPane που βρίσκεται στο πακέτο javax.swing.

Κατόπιν μέσα στο σώμα της μεθόδου act θα διαγράψουμε τις εντολές move και turn που υπάρχουν και θα γράψουμε τον παρακάτω κώδικα:

```
public void act()
   {
      if (Greenfoot.mousePressed(this)) {

         String inputStr = JOptionPane.showInputDialog("Γειά σου,πως ονομάζεσαι;");
         JOptionPane.showMessageDialog(null, "Γειά σου " + inputStr + "\n Πάτησε το κουμπί OK ή το
                                              πλήκτρο Enter για να κλείσει το παράθυρο.");
}
```

Κλείνουμε τον επεξεργαστή και κάνουμε μεταγλώττιση του προγράμματος. Κατόπιν, με δεξί κλικ πάνω στην κλάση Frog επιλέγουμε new Frog (), δημιουργούμε ένα νέο αντικείμενο Frog και το τοποθετούμε στον Κόσμο μας. Πατάμε το πλήκτρο **Εκκίνηση** και κάνουμε κλικ με το ποντίκι πάνω στο βάτραχο. Στο παράθυρο που εμφανίζεται δίνουμε το όνομα μας και κατόπιν εμφανίζεται το μήνυμα καλωσορίσματος, όπως φαίνεται στην Εικόνα 2.6.1.

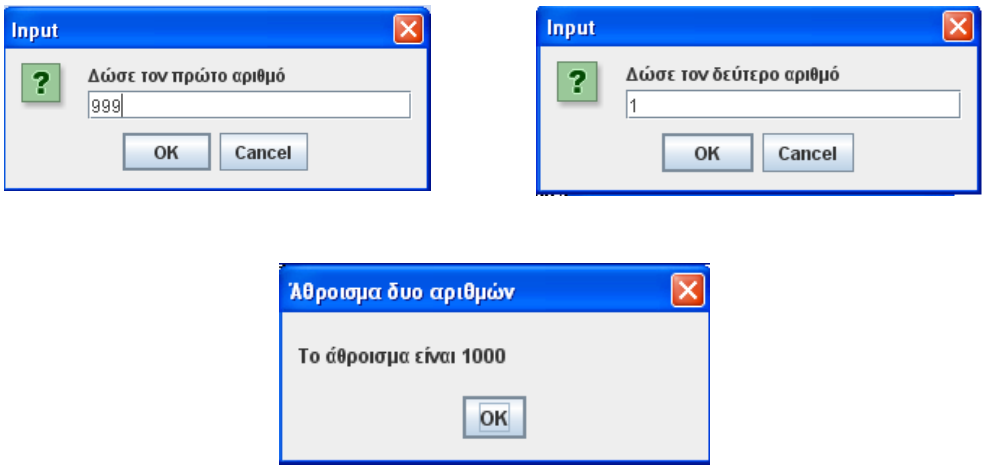


Εικόνα 2.6.1 Παράθυρα διαλόγου στο Greenfoot

Μπορούμε να τροποποιήσουμε την act έτσι ώστε να διαβάζει δύο (2) αριθμούς που θα εισάγει ο χρήστης και να εμφανίζει το άθροισμά τους.

```
import javax.swing.JOptionPane;

        public void act ( ) {
            String x = JOptionPane.showInputDialog("Δώσε τον πρώτο αριθμό");
            String y = JOptionPane.showInputDialog("Δώσε τον δεύτερο αριθμό");

            int number1 = Integer.parseInt(x);
            int number2 = Integer.parseInt(y);

            int sum = number1 + number2;

            JOptionPane.showMessageDialog(null, "Το άθροισμα είναι " + sum,
                                    "Άθροισμα δυο αριθμών", JOptionPane.PLAIN_MESSAGE);
        }
```

Τα παράθυρα διαλόγου που βλέπουμε στην οθόνη όταν τρέξουμε το πρόγραμμα φαίνονται στην Εικόνα 2.6.2.



Εικόνα 2.6.2 Παράθυρα διαλόγου της κλάσης JOptionPan

Αντί για PLAIN_MESSAGE για το είδος του μηνύματος υπάρχουν και οι εξής επιλογές:

τις οποίες μπορείτε να δοκιμάσετε και εσείς για να δείτε τα αντίστοιχα αποτελέσματα.

2.7. Δραστηριότητες

Δραστηριότητα 1

Να γράψετε ένα πρόγραμμα το οποίο θα διαβάζει αριθμούς από το πληκτρολόγιο μέχρι το άθροισμά τους να ξεπεράσει το 1000 και στη συνέχεια θα υπολογίζει και θα εμφανίζει το άθροισμά τους. Η είσοδος/έξοδος να υλοποιηθεί με χρήση της JOptionPane

Δραστηριότητα 2

Να μελετήσετε την τεκμηρίωση της κλάσης JOptionPane και να γράψετε πρόγραμμα το οποίο να χρησιμοποιεί τα διάφορα είδη διαγνωστικών μηνυμάτων.

Κεφάλαιο 3

Αντικείμενα και Μέθοδοι

3. Αντικείμενα και Μέθοδοι

Εισαγωγή

Στο 1ο κεφάλαιο είδαμε, τη δημιουργία κλάσεων και αντικειμένων, χωρίς την ανάγκη συγγραφής κώδικα στο περιβάλλον Greenfoot. Με αυτόν τον τρόπο όμως, κάθε φορά που εκτελείται από την αρχή το πρόγραμμα, πρέπει να ξαναδημιουργούμε τα αντικείμενα ένα – ένα.

Στο παρόν κεφάλαιο θα δούμε μια αυτοματοποιημένη διαδικασία δημιουργίας των αντικειμένων, η οποία να εκτελείται κάθε φορά που κατασκευάζεται ο κόσμος. Επίσης θα δούμε, πως μπορούμε να καθορίσουμε τη συμπεριφορά των αντικειμένων μας, υλοποιώντας τις δικές μας μεθόδους.

Στόχοι

Οι μαθητές να μπορούν να:

- Αναγνωρίζουν και να περιγράφουν τον τρόπο δημιουργίας αντικειμένων
- Αναζητούν και να χρησιμοποιούν τις υφιστάμενες μεθόδους που χρειάζονται
- Χρησιμοποιούν αντικείμενα έτοιμων τύπων για την επίλυση προβλημάτων
- Εξηγούν τον μηχανισμό μεταβίβασης παραμέτρων
- Αναγνωρίζουν τον τύπο επιστροφής μιας μεθόδου
- Υλοποιούν μεθόδους για τον χειρισμό γεγονότων του ποντικιού ή του πληκτρολογίου

Ενότητες κεφαλαίου

- Συμβολοσειρές
- Δημιουργία Αντικειμένων
- Κλήση μεθόδων των αντικειμένων
- Κλήση Στατικών Μεθόδων
- Δημιουργία των δικών μας μεθόδων
- Αναφορές και Πέρασμα Παραμέτρων
- Τύποι Επιστροφής
- Μέθοδοι και Κληρονομικότητα
- Καθοδήγηση από γεγονότα
- Πίνακες

3.1 Συμβολοσειρές

Εκτός από τους βασικούς τύπους της Java που γνωρίσαμε, ένας πολύ χρήσιμος τύπος είναι η κλάση String, του πακέτου java.lang, η οποία χρησιμεύει στην αναπαράσταση συμβολοσειρών (αλφαριθμητικά). Οι συμβολοσειρές είναι μία σειρά (ακολουθία) από σύμβολα, που μπορεί να είναι γράμματα, αριθμοί ή οποιοδήποτε άλλο σύμβολο του κώδικα UNICODE. Η κλάση String επίσης, περικλείει και όλη τη βασική λειτουργικότητα των συμβολοσειρών, την οποία και χρειαζόμαστε σε πρώτο στάδιο. Μια σταθερά τύπου συμβολοσειρά περικλείεται ανάμεσα σε διπλά εισαγωγικά " ".

Ένα String είναι ένα αντικείμενο και όπως όλα τα αντικείμενα, πρέπει να δηλωθεί και να δημιουργηθεί με την εντολή new, όπως στο ακόλουθο παράδειγμα: String s1 = new String();

Εναλλακτικά μπορούμε να δημιουργήσουμε ένα String, αποδίδοντας του ως τιμή μία σταθερά συμβολοσειρά, όπως στο παράδειγμα: String s2= "George";

Ο τελεστής + όταν εφαρμόζεται σε συμβολοσειρές τις ενώνει (string concatenation) σε μία πρόταση. Π.χ.: Από την παράσταση "I like Java" + " " + "Programming" προκύπτει η συμβολοσειρά "I like Java Programming"

Αν γράψουμε: System.out.println ("Το άθροισμα είναι " + 15+ 25);

Στην οθόνη εμφανίζεται το μήνυμα: Το άθροισμα είναι 1525.

Παρατηρούμε ότι όταν του τελεστή + προηγείται κάποιο string, μετατρέπει ότι βρίσκει δεξιά του επίσης σε string, ανεξάρτητα αν είναι αριθμός όπως το 15 στο παράδειγμα μας. Το ίδιο γίνεται και με το 25 στη συνέχεια. Έτσι στο παράδειγμα μας εμφανίζονται απλά οι δύο αριθμοί χωρίς να υπολογίζεται το άθροισμα τους.

Αν θέλουμε να γίνει πρώτα η πράξη της πρόσθεσης των δύο αριθμών και να εμφανιστεί στην οθόνη το άθροισμά τους , πρέπει η πράξη της πρόσθεσης των δύο αριθμών να μπει σε παρένθεση ώστε να προηγηθεί της συνένωσης με το string. Δηλαδή πρέπει να γράψουμε:

System.out.println ("Το άθροισμα είναι " + (15 + 25));

Αντίθετα, αν γράψουμε: System.out.println (15+25+ " ευρώ"); αυτό που θα εμφανιστεί στην οθόνη θα είναι: 40 ευρώ. Δηλαδή, εδώ προηγείται η πρόσθεση των δύο αριθμών και στη συνέχεια γίνεται η συνένωση με τη συμβολοσειρά « ευρώ» που ακολουθεί.

Η κλάση String εφοδιάζει τα αντικείμενα της με διάφορες χρήσιμες μεθόδους όπως η charAt(index), η οποία επιστρέφει τον χαρακτήρα στην θέση index, η μέθοδος length, η οποία επιστρέφει το μήκος της λέξης, η μέθοδος equals(string), η οποία συγκρίνει δύο συμβολοσειρές αν έχουν το ίδιο περιεχόμενο. Να σημειώσουμε ότι η αρίθμηση των χαρακτήρων σε ένα αντικείμενο τύπου String ξεκινάει από το 0 και όχι από το 1.

Ας δούμε μερικά ακόμα παραδείγματα:

```
String s1  = "Write Once, ";          // Δεσμεύει μνήμη για το αντικείμενο s1 και της
                                         αποδίδει περιεχόμενο το Write Once,
String s2  = "Run Anywhere.";         // Δεσμεύει μνήμη για το αντικείμενο s2 και της
                                         αποδίδει περιεχόμενο το Run Anywhere.
String s3  = s1 + s2;                 // Δεσμεύει μνήμη για το αντικείμενο s3 και της
                                         αποδίδει περιεχόμενο το "Write Once, Run Anywhere."
System.out.println( s3.charAt(0));    // Εμφανίζει το γράμμα 'W'
System.out.println( s3.length() );    // Εμφανίζει τον αριθμό 25 που είναι το μήκος της
                                         συμβολοσειράς s3
```

System.out.println("Σύγκριση: " + s1.equals(s2)); //Εμφανίζει το κείμενο: Σύγκριση: false, αφού τα δύο string s1 και s2 δεν έχουν το ίδιο περιεχόμενο.

Σημείωση: Δε συγκρίνουμε δύο συμβολοσειρές με τον τελεστή ισότητας ==, ακόμη και αν έχουν το ίδιο περιεχόμενο θα παίρνουμε πάντα σαν αποτέλεσμα false. Επειδή τα δύο string είναι αντικείμενα ο τελεστής == δε θα συγκρίνει το περιεχόμενο τους, αλλά τη διεύθυνση τους στη μνήμη.

3.2 Δημιουργία Αντικειμένων

8η εργαστηριακή άσκηση:

Η δημιουργία αντικειμένων (στιγμιότυπα) των κλάσεων, για να γίνει με αυτοματοποιημένο τρόπο, πρέπει να γράψουμε εντολές της γλώσσας Java, μέσα από τον επεξεργαστή κώδικα. Αυτό στις προηγούμενες ασκήσεις γινόταν κάθε φορά που, είτε ανοίγαμε το σενάριο μας, είτε μεταγλωττίζαμε τον κώδικα μας, με δεξί κλικ πάνω στην κλάση και κατόπιν επιλογή του πλαισίου «new όνομα_κλάσης()».

Για να εμφανίζονται ο βάτραχος και οι μύγες, αυτόματα, μόλις ανοίγουμε το σενάριο μας, πρέπει να προσθέσουμε εντολές στην μέθοδο εκείνη που κατασκευάζει τον Κόσμο FrogWorld. Αυτή η μέθοδος λέγεται **κατασκευαστής (constructor)** και έχει το ίδιο όνομα με την κλάση. **Ο** κατασκευαστής (constructor) είναι μια ειδικού τύπου μέθοδος, που εκτελείται πάντα και αυτόματα όταν δημιουργείται ένα νέο στιγμιότυπο (αντικείμενο) μιας κλάσης και έχει σκοπό την αρχικοποίηση του αντικειμένου. Κάθε κλάση περιέχει ένα προεπιλεγμένο (default) κατασκευαστή (αυτόματα από τη Java), χωρίς παραμέτρους και χωρίς εντολές στο σώμα του, παρόλο που δεν φαίνεται στον κώδικά της.

Ας ανοίξουμε τον επεξεργαστή του FrogWorld για να εξηγήσουμε τον κώδικα που περιέχει:

```
import greenfoot.*;
public class FrogWorld extends World
{
    public FrogWorld ()
  {
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
  }
}
```

Στην πρώτη γραμμή παρατηρούμε την εισαγωγή όλων των κλάσεων του πακέτου greenfoot, με την εντολή import.

Στην δεύτερη γραμμή δηλώνεται, με τη χρήση της λέξης **extends**, η σχέση της κληρονομικότητας μεταξύ των δύο κλάσεων FrogWorld (υποκλάση) και World (υπερκλάση).

Στην τρίτη γραμμή δηλώνεται ο κατασκευαστής της κλάσης FrogWorld. Όπως βλέπετε ο κατασκευαστής ορίζεται μέσα στην κλάση FrogWorld και έχει το ίδιο όνομα μ' αυτή.

Στην τέταρτη γραμμή, μέσα στη μέθοδο του κατασκευαστή, βλέπουμε την εντολή super(600, 400, 1). Με την εντολή **super** καλούμε τον κατασκευαστή της υπερκλάσης World (από την τεκμηρίωση της κλάσης World βλέπουμε ότι υπάρχει ο κατασκευαστής: World(int worldWidth, int worldHeight, int cellSize)) και ως παραμέτρους δίνουμε το πλάτος (600 κελιά), το ύψος (400 κελιά) και τη διάσταση κάθε κελιού (1 pixel) με σκοπό να δημιουργηθεί το πλέγμα του Κόσμου FrogWorld. Η εντολή super, στην περίπτωση που καλεί τον κατασκευαστή της υπερκλάσης, πρέπει να είναι η πρώτη δήλωση που γράφουμε στον κατασκευαστή της υποκλάσης της.

Τώρα ας δηλώσουμε, μέσα στον κατασκευαστή της FrogWorld, ποια αντικείμενα, των κλάσεων που έχουμε ορίσει, θα δημιουργούνται. Για να έχουμε ένα βάτραχο και μια μύγα στο σενάριο μας θα γράψουμε τις εντολές:

Frog kermit = **new** Frog();
Fly myga=**new** Fly();

Ο τελεστής **new** δημιουργεί ένα αντικείμενο της κλάσης, το οποίο και αποτελεί μοναδικό στιγμιότυπο της και επιστρέφει επίσης ένα **δείκτη αναφοράς (reference**) σε αυτό. Ο τελεστής new δεσμεύει επίσης την απαιτούμενη μνήμη για το αντικείμενο που δημιουργείται. Οι μεταβλητές kermit και myga είναι οι **δείκτες αναφοράς** αντίστοιχα στα αντικείμενα που δημιουργήσαμε προηγουμένως. Χρησιμοποιώντας αυτό το δείκτη αναφοράς μπορούμε να προσπελάσουμε το αντικείμενο.

Τα αντικείμενα που έχουν δημιουργηθεί βρίσκονται κάπου στη μνήμη του υπολογιστή. Αν θέλουμε να τα εμφανίσουμε μέσα στο πλέγμα του Κόσμου μας, πάμε στην κλάση World και βλέπουμε αν διαθέτει κάποια έτοιμη μέθοδο που κάνει αυτή τη δουλειά (αν δεν υπάρχει έτοιμη μέθοδος πρέπει να φτιάξουμε εμείς μια, γράφωντας κώδικα). Παρατηρούμε ότι υπάρχει η μέθοδος addObject της κλάσης World, την οποία και θα χρησιμοποιήσουμε. Η μέθοδος αυτή περιέχει τρεις παραμέτρους. Η πρώτη δηλώνει το αντικείμενο και οι δυο επόμενες τις συντεταγμένες που θα τοποθετηθεί. Ο κώδικας της κλάσης FrogWorld γίνεται:

```
public class FrogWorld extends World
{

    /**
     * Constructor for objects of class FrogWorld.
     *
     */
    public FrogWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);
        Frog kermit= new Frog();
        addObject(kermit, 100, 100 );

        Fly myga = new Fly();
        addObject( myga, 200, 200 );
```

Η παραπάνω εντολή κλήσης της μεθόδου addObject προσθέτει το αντικείμενο kermit στο σημείο του Κόσμου με συντεταγμένες (100,100) και το αντικείμενο myga στο σημείο του Κόσμου με συντεταγμένες (200,200). Η αρχή των αξόνων του Κόσμου μας είναι η πάνω αριστερή γωνία.

Με τον ίδιο τρόπο δημιουργούμε μερικά ακόμα αντικείμενα μύγες (Fly) στο πλέγμα του Κόσμου μας κα τα προσθέτουμε με τη μέθοδο addObject σε διαφορετικές συντεταγμένες του FrogWorld.

Παρακάτω φαίνεται ο κώδικας του κατασκευαστή της κλάσης FrogWorld στον οποίο έχουμε προσθέσει τις απαραίτητες εντολές για την δημιουργία αντικειμένων, ενός βατράχου και τριών μυγών.

```
import greenfoot.*;
public class  FrogWorld extends World
{
   public FrogWorld ()
   {
      super(600, 400, 1);
      Frog kermit= new Frog();
      addObject( kermit, 100, 100 );
      Fly myga = new Fly();
```

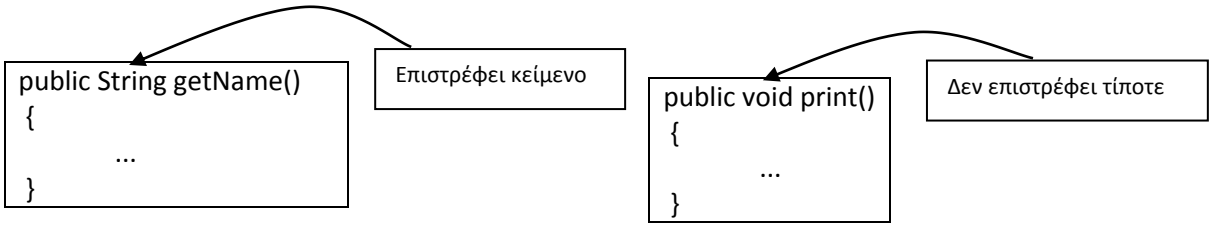
```
            addObject(myga, 200, 200 );
            Fly myga1= new Fly();
            addObject(myga1, 443, 160);
            Fly myga2 = new  Fly();
            addObject(myga2, 398, 335);
        }
    }
```

3.3 Κλήση μεθόδων των αντικειμένων

Είδαμε στο προηγούμενο κεφάλαιο ότι οι μέθοδοι στη Java ορίζουν τη συμπεριφορά των αντικειμένων, δηλαδή τις λειτουργίες που μπορεί να εκτελέσει ένα αντικείμενο. Μια μέθοδος είναι ένα μπλοκ κώδικα που έχει ένα όνομα και μπορούμε να την καλέσουμε από οπουδήποτε μέσα από το πρόγραμμα μας (είναι το ισοδύναμο των συναρτήσεων και των υποπρογραμμάτων στον αντικειμενοστρεφή προγραμματισμό).

Στη Java η σύνταξη μιας μεθόδου περιλαμβάνει τα εξής στοιχεία:

- ***Τον προσδιοριστή πρόσβασης***. Μπορεί να είναι **public**, **private** ή **protected** (θα συζητηθούν αργότερα).
- **Τον *τύπο της πληροφορίας που επιστρέφουν* (return type)**. Δείχνει τον τύπο δεδομένων της τιμής που επιστρέφεται από την μέθοδο. Π.χ. η μέθοδος int getSpeed() όταν εκτελεσθεί θα επιστρέψει έναν ακέραιο αριθμό που θα μας πληροφορεί για την τρέχουσα ταχύτητα του αντικειμένου. Ένα άλλο παράδειγμα μεθόδου είναι η boolean isAtEdge(), που είναι μια από τις μεθόδους της κλάσης Actor του Greenfoot. Η λέξη boolean μπροστά από το όνομα της μεθόδου σημαίνει ότι η τιμή που επιστρέφει η μέθοδος είναι true ή false και μας πληροφορεί αν το αντικείμενο έχει φτάσει ή όχι στην άκρη του πλέγματος. Αν η μέθοδος δεν επιστρέφει κάποια τιμή τότε χρησιμοποιούμε τη λέξη void (Η λέξη void σημαίνει «τίποτα»). Π.χ.



- ***Το όνομα της μεθόδου*.** Καλό είναι για όνομα να χρησιμοποιούμε μια λέξη που να υποδηλώνει το τι κάνει η συγκεκριμένη μέθοδος. Π.χ. η μέθοδος turn, δίνει εντολή στο αντικείμενο να στρίψει.
- ***Τη λίστα παραμέτρων.*** Αυτές γράφονται μέσα στις παρενθέσεις, μετά το όνομα της μεθόδου. Οι μέθοδοι μπορεί να περιέχουν καμία ή και περισσότερες από μια παραμέτρους. Οι παράμετροι ορίζονται με τον τύπο δεδομένων μπροστά και το όνομα να ακολουθεί και χωρίζονται μεταξύ τους με κόμμα. Αν δεν έχουμε παραμέτρους χρησιμοποιούμε απλά 2 κενές παρενθέσεις.
- ***Το σώμα της μεθόδου*** σε αγκύλες—με τον κώδικα και τις τοπικές μεταβλητές.

Τα τρία στοιχεία της σύνταξης μιας μεθόδου: ο τύπος επιστροφής, το όνομα μεθόδου και η λίστα παραμέτρων ονομάζονται **υπογραφή της μεθόδου** (method signature).

```
[public ή private ή protected] [τύπος επιστροφής] [όνομα μεθόδου](παράμετρος1, ..., παράμετροςN)
{
  σώμα μεθόδου
}
```

Μια ειδική μέθοδος της Java είναι η main η οποία είναι επιφορτισμένη με το να ξεκινάει την εκτέλεση του προγράμματος. Στη Java για να γίνει ένα πρόγραμμα εκτελέσιμο, πρέπει οπωσδήποτε σε κάποιο από τα αρχεία πηγαίου κώδικα, από τα οποία αποτελείται, να υπάρχει τουλάχιστον μία μέθοδος main.

Η σύνταξη της μεθόδου είναι: public static void main(String[] args) { }

Όταν ο διερμηνέας της Java εκτελεί ένα πρόγραμμα ξεκινά από την κλήση της μεθόδου main. Η main κατόπιν καλεί όλες τις υπόλοιπες μεθόδους για να εκτελεστεί το πρόγραμμα.

9η εργαστηριακή άσκηση:

Στο σενάριο FrogEatFlies έχουμε ήδη ένα βάτραχο και μερικές μύγες και μπορούν πλέον να αλληλοεπιδράσουν μεταξύ τους. Η αλληλεπίδραση (interaction) έγκειται κυρίως, στην εκτέλεση εντολών από τα αντικείμενα, μέσα από τη μετάδοση μηνυμάτων. Πρώτα θα γράψουμε κώδικα ώστε όλα τ' αντικείμενα να κινούνται μέσα στο πλέγμα του Κόσμου, χρησιμοποιώντας τις μεθόδους που έχουν κληρονομήσει από την κλάση Actor.

Ανοίγοντας την τεκμηρίωση της κλάσης Actor, βλέπουμε όλες τις μεθόδους της. Οι μέθοδοι που χρειαζόμαστε είναι η *move,* η οποία δίνει σήμα στο αντικείμενο να μετακινηθεί εμπρός σε απόσταση που ορίζουμε στην τιμή που δίνουμε στην παράμετρο της και η *turn,* η οποία λέει στο αντικείμενο να στρίψει κατά γωνία ίση με την τιμή της παραμέτρου της. Μια μικρή περιγραφή για την κάθε μέθοδο βλέπουμε στην Εικόνα 3.3.1, όπως φαίνεται στην τεκμηρίωση της κλάσης Actor.

move

```
public void move(int distance)
```

Move this actor the specified distance in the direction it is currently facing.

The direction can be set using the `setRotation(int)` method.

Parameters:

`distance` - The distance to move (in cell-size units); a negative value will move backwards

See Also:

`setLocation(int, int)`

---

turn

```
public void turn(int amount)
```

Turn this actor by the specified amount (in degrees).

Parameters:

`amount` - the number of degrees to turn; positive values turn clockwise

Εικόνα 3.3.1 Οι μέθοδοι move και turn

Στο περιβάλλον του Greenfoot, όταν δημιουργείται μια νέα υποκλάση της κλάσης Actor, προστίθεται αυτόματα στον κώδικα της νέας υποκλάσης η μέθοδος act. Η μέθοδος αυτή είναι αρχικά κενή και μέσα στο σώμα της γράφεται ο κώδικας που καθορίζει τις ενέργειες που πρέπει να κάνει το αντικείμενο.Η μέθοδος αυτή καλείται σε όλα τα αντικείμενα του Κόσμου, που έχουμε δημιουργήσει, κάθε φορά που πατάμε το κουμπί *Δράση* ή *Εκκίνηση* στο περιβάλλον του Greenfoot.

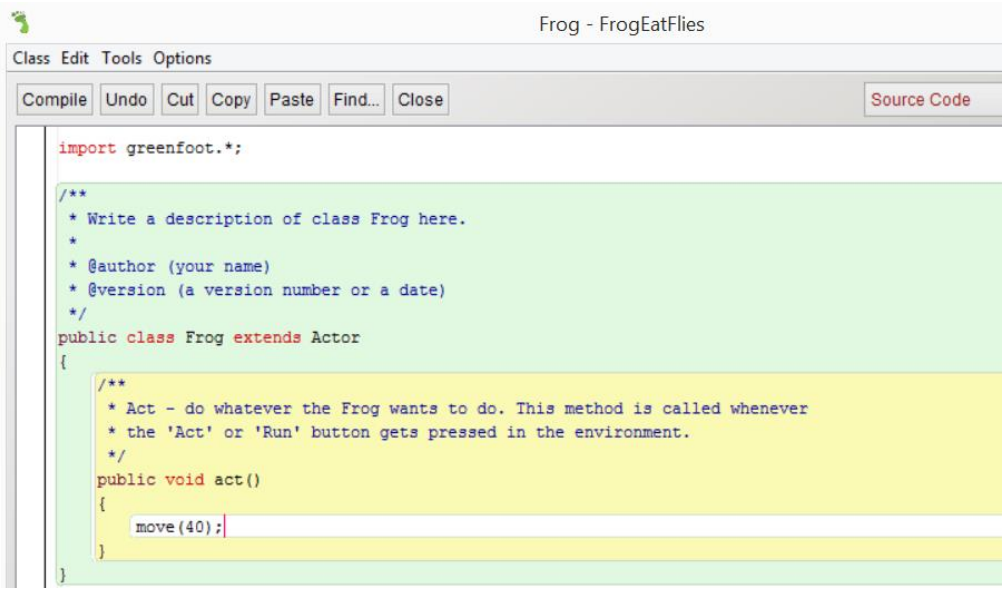
Άρα, τις μεθόδους move και turn θα τις καλούμε μέσα από τη μέθοδο act της κλάσης του αντικειμένου.

Ας δούμε τη μέθοδο move. Η μέθοδος ορίζεται ως εξής:

```
void move( πλήθος βημάτων )
```

Η move έχει ως παράμετρο έναν ακέραιο αριθμό (int) ο οποίος εκφράζει το πλήθος των βημάτων (την απόσταση) που θέλουμε να προχωρήσει το αντικείμενο.

Για παράδειγμα ανοίγουμε τη μέθοδο act του βατράχου (επιλέγοντας *Άνοιγμα επεξεργαστή*) και γράφουμε την εντολή move(40), δηλαδή του δίνουμε την εντολή να μετακινηθεί κατά 40 βήματα. Με όμοιο τρόπο προσθέτουμε μια εντολή move και στην κλάση Fly, αλλά με διαφορετικό πλήθος βημάτων.



Εικόνα 3.3.2 Συγγραφή κώδικα στη μέθοδο act()

Πατώντας το κουμπί ***Εκκίνηση,*** κάτω στην οθόνη, εκτελείται διαρκώς (επαναληπτική διαδικασία) η act, για όλα τα αντικείμενα του Κόσμου μας, τα οποία κινούνται με διαφορετική ταχύτητα, ανάλογα με την τιμή της παραμέτρου που έχουν στη move, μέχρι να φτάσουν στην άκρη του Κόσμου. Τ' αντικείμενα στην πραγματικότητα δεν σταματούν με αυτόν τον τρόπο. Εξακολουθούν να προσπαθούν να κινηθούν, αλλά το Greenfoot δεν τ' αφήνει να κινούνται έξω από τον κόσμο (αν τ΄ άφηνε, πώς θα τα μεταφέραμε πάλι πίσω;).

Αλλάζοντας, σε ένα αντικείμενο, την παράμετρο της μεθόδου move με ένα μεγαλύτερο αριθμό η κίνηση θα είναι ταχύτερη (αφού σε κάθε βήμα της επαναληπτικής διαδικασίας θα διανύει μεγαλύτερη απόσταση), ενώ με μικρότερο θα είναι πιο αργή. Μπορείτε να μαντέψετε τι θα συμβεί αν βάλετε έναν αρνητικό αριθμό;

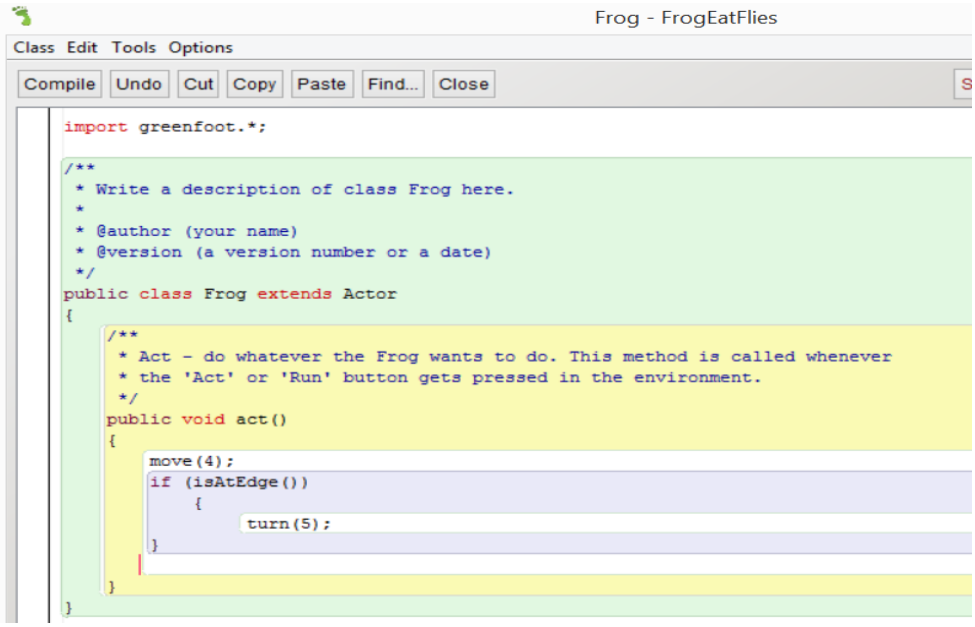
Ας προσθέσουμε μέσα στο σώμα της μεθόδου act και την εντολή *turn*, γράφοντας: turn(5);

Άλλαξε κάτι στην κίνηση του αντικειμένου;

Όσο μικρότερος ο αριθμός στην παράμετρο της μεθόδου turn, τόσο μεγαλύτερο τόξο θα διαγράφει το κινούμενο αντικείμενο (αφού σε κάθε βήμα της επαναληπτικής διαδικασίας θα στρίβει κατά μικρότερη γωνία).

Στη συνέχεια θα κάνουμε τ' αντικείμενα ν' αντιδρούν σε ερεθίσματα από το περιβάλλον τους, παραδείγματος χάρη όταν φτάνουν στην άκρη του Κόσμου να στρίβουν για να συνεχίσουν την κίνηση τους. Η μέθοδος που θα χρησιμοποιήσουμε είναι η isAtEdge(). Η μέθοδος αυτή δεν παίρνει παραμέτρους και επιστρέφει την πληροφορία true ή false, πράγμα που σημαίνει ότι μπορεί να χρησιμοποιηθεί ως συνθήκη μέσα σε μια δομή επιλογής if.

Να γράψετε μέσα στο σώμα της μεθόδου act τον κώδικα, ώστε ο βάτραχος να κινείται ευθεία κατά 4 βήματα τη φορά και **αν** φτάνει στην άκρη του Κόσμου τότε να στρίβει κατά 5 μοίρες. Ο κώδικας στην κλάση Kermit μετά την υλοποίηση των παραπάνω θα γίνει:



Να γράψετε τον αντίστοιχο κώδικα, ώστε η μύγα να κινείται ευθεία κατά 8 βήματα τη φορά και αν φτάνει στην άκρη του Κόσμου τότε να στρίβει κατά 15 μοίρες.

3.4 Κλήση Στατικών Μεθόδων

Στη Java υπάρχουν δύο τύποι μεθόδων: αυτές που ανήκουν στο αντικείμενο, ονομάζονται ***μέθοδοι στιγμιοτύπου (instance methods)*** και καθορίζουν τη συμπεριφορά του αντικειμένου και αυτές που ανήκουν στην κλάση, ονομάζονται ***μέθοδοι κλάσης (class methods)*** και δεν αναφέρονται σε συγκεκριμένο αντικείμενο**.**

Οι μέθοδοι του αντικειμένου (μέθοδοι στιγμιότυπου – instance methods), μπορούν να κληθούν μόνο μέσω κάποιου αντικειμένου. Η κλήση μιας μεθόδου στιγμιοτύπου συντάσσεται με το όνομα του αντικειμένου ακολουθούμενο από την τελεία και το όνομα της μεθόδου, δηλαδή:

Όνομα αντικειμένου. Όνομα μεθόδου (παράμετροι);

Τέτοια παραδείγματα είναι οι μέθοδοι move και turn, που συναντήσαμε προηγουμένως, οι οποίες δίνουν εντολή σε συγκεκριμένο αντικείμενο να μετακινηθεί ή να στρίψει αντίστοιχα. Αν λοιπόν θέλουμε το αντικείμενο kermit της κλάσης Frog να κινηθεί 100 βήματα θα πρέπει να δώσουμε την εντολή:

kermit.move(100)

Είναι φανερό λοιπόν ότι η χρήση της move χωρίς να αναφερόμαστε σε συγκεκριμένο αντικείμενο δεν έχει νόημα (όταν είμαστε μέσα στην κλάση του αντικειμένου, η αναφορά του ονόματος του είναι περιττή, όπως βλέπουμε και στην Εικόνα 3.3.2).

Πολλές φορές όμως, θέλουμε να σχεδιάσουμε ένα σύνολο μεθόδων οι οποίες δεν είναι συνδεδεμένες με κάποιο αντικείμενο της κλάσης, αλλά υλοποιούν λειτουργίες χρήσιμες για την ίδια την κλάση. Σε αυτή την περίπτωση ορίζουμε μια μέθοδο ως **μέθοδο κλάσης (**class method), χρησιμοποιώντας στην αρχή της δήλωσης της, τη λέξη κλειδί **static**. Οι μέθοδοι κλάσης είναι γνωστές και ως **στατικές μέθοδοι**. Μπορούμε λοιπόν, να καλέσουμε και να χρησιμοποιήσουμε στατικές μεθόδους μιας κλάσης, χωρίς να χρειάζεται να δημιουργηθεί πρώτα κάποιο αντικείμενο αυτής της κλάσης .

Για παράδειγμα, έστω ότι ορίζουμε τη στατική μέθοδο:

```
static int numberOfFrogs( )          //Δήλωση στατικής μεθόδου η οποία μας
                                       επιστρέφει το πλήθος των βατράχων που
                                       υπάρχουν αυτή τη στιγμή στον κόσμο μας
```

Η κλήση της μεθόδου μπορεί να γίνει, είτε από την κλάση, είτε από κάποιο αντικείμενό της κλάσης, όπως φαίνεται παρακάτω:

frogs = Frog.numberOfFrogs() ή frogs = kermit.numberOfFrogs()

όπου το kermit είναι ένα αντικείμενο της κλάσης Frog.

Παράδειγμα: Όλες οι μέθοδοι της κλάσης Math, η οποία βρίσκεται στο πακέτο java.lang, είναι στατικές. Η κλήση τους γίνεται ως εξής: int x = **Math.**max (a, b) ; int y=**Math**.abs(a); int z=**Math**.min(a,b);

Επίσης όλες οι μέθοδοι της κλάσης Greenfoot, η οποία βρίσκεται στο πακέτο greenfoot, είναι στατικές. Η κλήση τους γίνεται ως εξής: x=**Greenfoot**.getRandomNumber(limit);

10η εργαστηριακή άσκηση:

Στο σενάριο μας FrogEatFlies τα αντικείμενα μας κινούνται μέσα στο πλέγμα του Κόσμου, αλλά ή κίνηση τους είναι πάντα η ίδια. Αυτό που θέλουμε στη συνέχεια είναι να προσθέσουμε τυχαία συμπεριφορά στην κίνηση των αντικειμένων μας. Αυτό επιτυγχάνεται στο Greenfoot χρησιμοποιώντας μια μέθοδο που μας παρέχει η κλάση Greenfoot, την getRandomNumber, η οποία επιστρέφει τυχαίους ακέραιους αριθμούς. Η μέθοδος αυτή είναι στατική, δηλαδή δε χρειάζεται να δημιουργηθεί πρώτα κάποιο αντικείμενο αυτής της κλάσης για να τη χρησιμοποιήσουμε.

Η υπογραφή της μεθόδου είναι: int getRandomNumber(int limit)

Η μέθοδος αυτή δέχεται έναν ακέραιο αριθμό (limit) ως παράμετρο, που καθορίζει το όριο των τυχαίων αριθμών που μπορεί να παράγει, δηλαδή παράγει τυχαίους αριθμούς από το 0 έως το limit-1. Π.χ. η getRandomNumber(20) επιστρέφει τυχαίους αριθμούς από το 0 έως 19.

Γράψουμε κώδικα μέσα στη μέθοδο act της κλάσης Frog ώστε, ο βάτραχος να στρίβει κατά 15 μοίρες, μόνο αν ο τυχαίος αριθμός που παράγεται είναι μικρότερος του 10, αλλιώς να συνεχίζει ευθεία και θα στρίβει κατά 5 μοίρες, μόνο όταν φτάνει στην άκρη του Κόσμου. Ο κώδικας έχει την παρακάτω μορφή:

```
public void act()
{
   move(4);
   if (Greenfoot.getRandomNumber(100)<10)
    {
     turn(15);
    }
   if (isAtEdge())
       {
          turn(5);
       }
 }
```

Αφού η getRandomNumber(100) μας δίνει κάθε φορά ένα τυχαίο αριθμό μεταξύ 0 και 99 και δεδομένου ότι οι αριθμοί είναι ομοιόμορφα κατανεμημένοι, στο 10% των περιπτώσεων ο αριθμός που θα παράγεται θα είναι κάτω από 10 και η συνθήκη Greenfoot.getRandomNumber(100)<10 θα είναι αληθής.

Μπορείτε να προγραμματίσετε το βάτραχο ώστε να μη στρίβει κατά 15 μοίρες πάντα, αλλά κατά τυχαία κλίση μέχρι 20 μοίρες; Ποια παράμετρο του προγράμματός σας θα αλλάξετε;

Αντικαταστήστε την εντολή turn(15); με την:

```
int angle = Greenfoot.getRandomNumber(20);
turn(angle);
```

3.5 Δημιουργία των δικών μας μεθόδων

Κατά τη διάρκεια ανάπτυξης μιας εφαρμογής, ορισμένα τμήματα κώδικα (μέθοδοι) χρειάζεται να επαναλαμβάνονται πολλές φορές. Συνήθως, τα τμήματα αυτά αναφέρονται σε λειτουργίες που χρησιμοποιούνται συχνά. Όπως θα δούμε έχουμε τη δυνατότητα να δώσουμε ένα όνομα σε αυτά τα τμήματα κώδικα και να τα καλούμε με αυτό, όποτε τα χρειαζόμαστε. Πρόκειται δηλαδή για μεθόδους που τις δημιουργούμε εμείς για να κάνουν μια συγκεκριμένη δουλειά και αποτελούν μέρη μιας κλάσης.

11η εργαστηριακή άσκηση:

Έως τώρα, στο σενάριο μας, προσθέτουμε διαρκώς γραμμές κώδικα μέσα στη μέθοδο act των κλάσεων μας. Αν συνεχίσουμε έτσι η μέθοδος act θα μεγαλώνει διαρκώς και σε λίγο θα γίνει πολύ δύσκολο να διαβάζουμε τον κώδικα της και να καταλαβαίνουμε τι κάνει. Αν θέλουμε να βελτιώσουμε τον τρόπο που γράφουμε τον κώδικα μας, πρέπει να τον κόψουμε σε μικρότερα μέρη, δημιουργώντας μια μέθοδο για κάθε ομάδα εντολών που επιτελεί μια συγκεκριμένη λειτουργία. Οι νέες μέθοδοι δηλώνονται έξω από το σώμα της μεθόδου act().

Μέχρι τώρα μέσα στη μέθοδο act της κλάσης Frog εκτελούνται 3 διαφορετικές ενέργειες: η κίνηση του βατράχου προς την κατεύθυνση που βρίσκεται, η τυχαία στροφή του και η στροφή του κατά 5 μοίρες όταν φτάσει την άκρη του Κόσμου. Αυτό που θα κάνουμε στη συνέχεια είναι να φτιάξουμε 2 διαφορετικές μεθόδους, μια για κάθε στροφή. Η ενέργεια move(4) δε χρειάζεται να μπει σε ξεχωριστή μέθοδο γιατί είναι μια εντολή μόνο.

Ορίζουμε μια νέα μέθοδο με το όνομα randomTurn, η οποία στρέφει το βάτραχο κάθε φορά κατά τυχαία κλίση μέχρι 20 μοίρες. Έτσι η μέθοδος *act* της κλάσης Frog μπορεί να ξαναγραφτεί, περιέχοντας λιγότερες λεπτομέρειες, αφού στη θέση του κώδικα της μεθόδου **randomTurn()**, γράφουμε απλώς το όνομα της.

```
public void act()
 {
    move(4);
    randomTurn();

    if (isAtEdge())
        {
           turn(5);
        }
 }
 public void randomTurn()
 {
   if (Greenfoot.getRandomNumber(100)<10)
   {
    int angle = Greenfoot.getRandomNumber(20);
    turn(angle);
   }
 }
```

Δημιουργήστε μια νέα μέθοδο μέσα στην κλάση Frog με όνομα turnAtEdge, χωρίς παραμέτρους. Επιλέξτε το κομμάτι του κώδικα μέσα στη μέθοδο act, όπου γίνεται η στροφή του βάτραχου κατά 5 μοίρες και βάλτε το στη συνέχεια μέσα στη μέθοδο turnAtEdge. Τέλος, καλέστε τη

μέθοδο μέσα από την μέθοδο act, γράφοντας απλώς το όνομα της (Μην ξεχάσετε να διαγράψετε το κομμάτι του κώδικα που πήρατε από τη μέθοδο act και το βάλατε μέσα στη μέθοδο turnAtEdge).

Με τη δημιουργία των δύο νέων μεθόδων καταφέραμε δυο πράγματα. Να απλοποιήσουμε αφενός, τον κώδικα της μεθόδου *act,* ώστε να γίνει πιο απλός και κατανοητός για αυτόν που τον διαβάζει και θέλει να καταλάβει τι κάνει και αφετέρου, να υλοποιήσουμε τις μεθόδους randomTurn και turnAtEdge ώστε να είναι ανεξάρτητες από την *act*. Οπότε, αν θέλουμε στη συνέχεια να αλλάξουμε κάτι, όπως π.χ. το αντικείμενο να στρίβει κάθε φορά κατά τυχαία κλίση μέχρι 40 μοίρες και όχι μέχρι 20 μοίρες, αρκεί να αλλάξουμε τον κώδικα μέσα στην randomTurn (αντικαθιστώντας το 20 με το 40), ενώ η act δεν θα χρειαστεί καμία αλλαγή.

Σημείωση: Στη Java τα ονόματα των μεθόδων ξεκινούν με μικρά γράμματα και δεν περιέχουν κενά. Αν το όνομα περιέχει πολλές λέξεις τότε χρησιμοποιούμε κεφαλαίο γράμμα κάθε φορά που αρχίζει μια νέα λέξη π.χ. turnAtEdge.

3.6 Αναφορές και Πέρασμα Παραμέτρων

12η εργαστηριακή άσκηση:

Η μέθοδος turnAtEdge, που δημιουργήσαμε στην προηγούμενη άσκηση, κάνει το βάτραχο να στρίβει κατά 5 μοίρες κάθε φορά που φτάνει στην άκρη του Κόσμου. Αν θέλουμε να στρίβει κατά διαφορετικό αριθμό μοιρών, π.χ. 8 μοίρες, πρέπει να ορίσουμε μια διαφορετική μέθοδο turnAtEdge για κάθε περίπτωση. Αντί να το κάνουμε αυτό μπορούμε να χρησιμοποιήσουμε μία παράμετρο στην μέθοδο, ώστε να μπορεί να χρησιμοποιηθεί κάθε φορά με διαφορετικό αριθμό μοιρών. Περνάμε λοιπόν, τον αριθμό των μοιρών ως παράμετρο της μεθόδου turnAtEdge, ώστε κάθε φορά που την καλούμε να ορίζουμε εμείς το πόσες μοίρες θα στρίβει ο βάτραχος. Για το σκοπό αυτό χρησιμοποιούμε μια ακέραια μεταβλητή την degrees.

```
    public void turnAtEdge (int degrees)
{
   if (isAtEdge())
       {
          turn(degrees);
       }
}
public void act()
{
   move(4);
   randomTurn();

   for (int i=0; i<20; i++)
    {
      turnAtEdge( i );
    }
 }
```

Στον παραπάνω κώδικα βλέπουμε ότι, για να εκτελεστεί η μέθοδος turnAtEdge, χρειάζεται να της δώσουμε έναν ακέραιο αριθμό (που είναι η τιμή για την παράμετρο degrees) μέσα στις παρενθέσεις, μετά το όνομα της μεθόδου.

Παρακάτω στον κώδικα μας, χρησιμοποιούμε μιά δομή επανάληψης for όπου η μεταβλητή i παίρνει τιμές από 0 έως 20 και κάθε φορά, στην κλήση της μεθόδου turnAtEdge που γίνεται μέσα στο βρόγχο for, η τρέχουσα τιμή της i περνά ως τιμή της παραμέτρου.

Τις μεταβλητές, που χρησιμοποιούνται στον ορισμό της μεθόδου, τις ονομάζουμε **παραμέτρους** της μεθόδου, ενώ τις τιμές που παίρνουν οι παράμετροι κατά την κλήση της μεθόδου, τις ονομάζουμε **ορίσματα**. Στο παραπάνω παράδειγμα η μεταβλητή degrees είναι η παράμετρος ενώ το i είναι το όρισμα της μεθόδου turnAtEdge.

Η μεταβίβαση παραμέτρων στη Java είναι, όπως λέμε, **κατά τιμή** (**pass-by-value)** δηλαδή δημιουργούνται αντίγραφα των ορισμάτων, οπότε οποιαδήποτε αλλαγή στις παραμέτρους εντός της μεθόδου δεν έχει καμία επίδραση στα ορίσματα-μεταβλητές που έχουν οριστεί εκτός της μεθόδου. Στο παραπάνω παράδειγμα αυξάνεται κατά ένα η μεταβλητή i όμως η αλλαγή αυτή δεν μεταφέρεται πίσω στην *degrees* η οποία μένει αναλλοίωτη, αφού η i είναι ένα αντίγραφο της degrees.

Στην περίπτωση όμως που η παράμετρος μας δεν αναφέρεται σε βασικό τύπο δεδομένων (int, float, double, κλπ) αλλά σε αντικείμενο κάποιας κλάσης, δεν δημιουργείται αντίγραφο του αντικειμένου. Αυτό συμβαίνει γιατί όταν αναφερόμαστε σε ένα αντικείμενο σύνθετου τύπου η μεταβλητή δεν περιέχει το ίδιο το αντικείμενο αλλά μια διεύθυνση στην περιοχή της μνήμης όπου είναι αποθηκευμένο το αντικείμενο. Η διεύθυνση αυτή λέγεται **αναφορά (reference)** και αποτελεί ουσιαστικά έναν δείκτη στην περιοχή της μνήμης που βρίσκεται το αντικείμενο. Κατά το πέρασμα παραμέτρων δημιουργείται αντίγραφο της αναφοράς και όχι του ίδιου του αντικειμένου, οπότε μέσω της αναφοράς μπορούμε να τροποποιήσουμε το αντικείμενο.

Μέχρι τώρα στο σενάριο μας τα δύο αντικείμενα, ο βάτραχος και η μύγα, δεν αλληλοεπιδρούν μεταξύ τους. Οι βάτραχοι όμως τρώνε τις μύγες, οπότε το επόμενο βήμα που πρέπει να κάνουμε είναι, να βάλουμε τα δύο αντικείμενα ν' αλληλοεπιδρούν, δηλαδή ο βάτραχος μόλις βρει κάποια μύγα να την τρώει.

Οι μέθοδοι της κλάσης Actor, που χρησιμοποιούμε για να προγραμματίσουμε τη συμπεριφορά του βατράχου είναι η isTouching (η οποία ανιχνεύει αν ένα αντικείμενο αγγίζει κάποιο άλλο) και η removeTouching (η οποία εξαφανίζει ένα αντικείμενο όταν το ακουμπήσει κάποιο άλλο). Και οι δύο μέθοδοι δέχονται ως παράμετρο το όνομα μιας κλάσης.

Μέσα στην κλάση Frog ορίζουμε μια νέα μέθοδο με όνομα *eatFlies* που υλοποιεί την ενέργεια που περιγράψαμε παραπάνω.

```
   public void eatFlies()
{
   if (isTouching(Fly.class))
    {
      removeTouching(Fly.class);
    }
}
```

Καλέστε τη μέθοδο eatFlies μέσα από την μέθοδο act.

3.7 Τύποι Επιστροφής

Οι συναρτήσεις σε πολλές γλώσσες προγραμματισμού, όπως στην Pascal και την Python επιστρέφουν ένα αποτέλεσμα με το όνομά τους. Το ίδιο συμβαίνει και με τις μεθόδους της java. Ωστόσο, υπάρχουν μέθοδοι οι οποίοι δεν επιστρέφουν κάποια τιμή, αλλά υλοποιούν κάποια ενέργεια όπως για παράδειγμα την μετακίνηση ενός αντικειμένου. Αυτές οι μέθοδοι που δεν επιστρέφουν τίποτα δηλώνονται με τον τύπο επιστροφής ***void***, όπως για παράδειγμα οι παραπάνω μέθοδοι *act* και *eatFlies*.

13η εργαστηριακή άσκηση:

Στη συνέχεια του σεναρίου μας θέλουμε να προσθέσουμε ένα μετρητή ώστε, κάθε φορά που ο βάτραχος τρώει μια μύγα, να παίρνει ένα πόντο και να κρατάμε το σκορ, το οποίο στη συνέχεια θα εμφανίζεται στην οθόνη μας.

Ανοίγουμε την κλάση Frog για να δηλώσουμε μια μεταβλητή αντικειμένου (Ενότητα 2.1.3) που θα την ονομάσουμε fliesEaten η οποία αποθηκεύει τον αριθμό των μυγών που έφαγε ο βάτραχος. Η τιμή αυτής της μεταβλητής αυξάνεται κατά ένα, όταν εκτελείται η μέθοδος eatFlies.

Ορίζουμε τη νέα μεταβλητή αντικειμένου στην αρχή της κλάσης Frog ως εξής:

private int fliesEaten;

Όπως βλέπουμε ο τύπος της πληροφορίας που αποθηκεύει η μεταβλητή είναι ένας ακέραιος (int) αριθμός.

Η αρχικοποίηση των ακεραίων μεταβλητών γίνεται αυτόματα από τη Java, έτσι η μεταβλητή fliesEaten έχει αρχική τιμή 0. Αν θέλαμε η αρχική της τιμή να είναι διαφορετική από το 0 (π.χ. το 40), τότε θα μπορούσαμε να την αρχικοποιήσουμε χρησιμοποιώντας την έκφραση:

private int fliesEaten = 40;

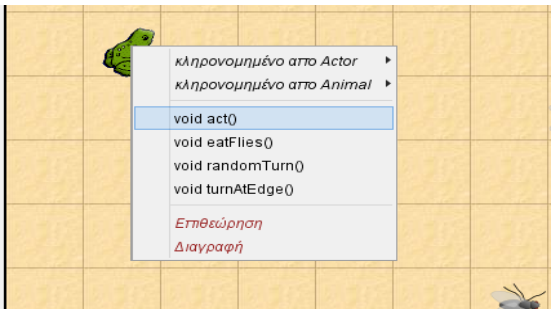
Ο κώδικας που αυξάνει την τιμή της *fliesEaten* γράφεται μέσα στη μέθοδο eatFlies και μέσα στο σώμα της εντολής if. Ο τελεστής ++ μετά το όνομα της μεταβλητής αυξάνει την τιμή της κατά ένα.

Η μέθοδος eatFlies γίνεται:

```
public void eatFlies()
{
   if (isTouching(Fly.class))
    {
      removeTouching(Fly.class);
      fliesEaten ++;
    }
}
```

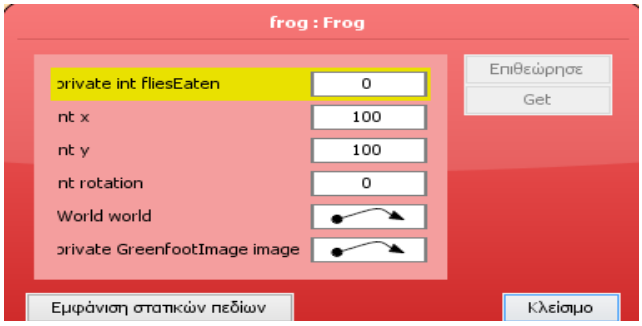
Όταν γράφουμε Fly.class αναφερόμαστε σε ένα αντικείμενο που περιγράφει την κλάση Fly. Αυτό μας δίνει τη δυνατότητα να ελέγξουμε μέσω της isTouching(Fly.class) αν το αντικείμενό μας ακουμπάει ένα αντικείμενο της κλάσης Fly.

Αφού μεταγλωττίσουμε τον κώδικά μας, κάνουμε δεξί κλικ πάνω στο βάτραχο και παρατηρούμε, στο παράθυρο που ανοίγει, ότι υπάρχουν οι μέθοδοι που ορίσαμε μέσα στον κώδικα της κλάσης Frog (Εικόνα 3.7.1).



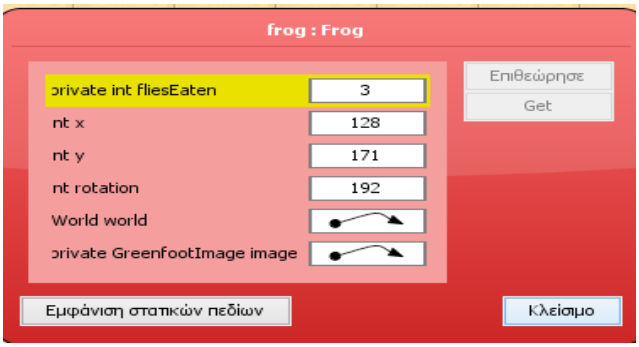
Εικόνα 3.7.1 Μέθοδοι της κλάσης Froa

Επίσης, αν πατήσουμε στο ***Επιθεώρηση*** θα δούμε ότι στις ιδιότητες του αντικειμένου βάτραχος έχει προστεθεί μία ακόμα, η fliesEaten με τιμή 0, αυτή που δηλώσαμε προηγουμένως (Εικόνα 3.7.2).



Εικόνα 3.7.2 Ιδιότητες του αντικειμένου frog

Αν τρέξουμε το πρόγραμμα μας και ο βάτραχος φάει μερικές μύγες, κάνοντας πάλι δεξί κλικ πάνω στο βάτραχο και επιλέγοντας το ***Επιθεώρηση***, θα δούμε ότι η μεταβλητή fliesEaten έχει αποθηκεύσει τη νέα τιμή, δηλαδή τον αριθμό των μυγών που έφαγε ο βάτραχος (Εικόνα 3.7.3).



Εικόνα 3.7.3 Η τιμή της μεταβλητής fliesEaten

Αυτό που χρειαζόμαστε τώρα είναι να εμφανίζουμε στην οθόνη την τιμή της μεταβλητής fliesEaten με τη μορφή ενός μετρητή. Ο μετρητής αυτός θα είναι δυναμικό κείμενο, δηλαδή κείμενο που μεταβάλλεται κάθε φορά που αλλάζει η τιμή της μεταβλητής.

Το κείμενο θεωρείται κι αυτό αντικείμενο και για να μπορέσουμε να το χειριστούμε πρέπει πρώτα να ορίσουμε την κλάση του. Δημιουργούμε μια νέα υποκλάση της *Actor* και την ονομάζουμε Counter, χωρίς να δώσουμε κάποια εικόνα για το αντικείμενο που δημιουργείται από αυτή την κλάση. Αντίθετα, του δίνουμε μια δυναμική εικόνα κατά τη εκτέλεση του σεναρίου, χρησιμοποιώντας τη μέθοδο setImage(GreenfootImage image) της κλάσης Actor. Παρατηρούμε ότι, η παράμετρος που δέχεται αυτή η μεθόδος είναι ένα αντικείμενο εικόνα της κλάσης GreenfootImage.

Η βιβλιοθήκη του greenfoot περιέχει μια κλάση που ονομάζεται GreenfootImage, η οποία παρέχει μεθόδους για το χειρισμό των εικόνων. Οι μέθοδοι της κλάσης GreenfootImage είναι μέθοδοι αντικειμένου, αυτό σημαίνει ότι για να τις χρησιμοποιήσουμε πρέπει να δημιουργήσουμε πρώτα ένα αντικείμενο τύπου *GreenfootImage* χρησιμοποιώντας τη λέξη *new*.

Η κλάση GreenfootImage διαθέτει πέντε διαφορετικούς κατασκευαστές, δηλαδή πέντε διαφορετικές μεθόδους με το ίδιο όνομα για να δημιουργούμε αντικείμενα αυτής της κλάσης (Εικόνα 3.7.4). Οι διαφορετικοί κατασκευαστές διαφοροποιούνται μεταξύ τους ως προς τις παραμέτρους που δέχονται (το πλήθος των παραμέτρων ή/και τον τύπο των παραμέτρων). Αυτό γίνεται όταν θέλουμε ο κατασκευαστής να κάνει διαφορετικές λειτουργίες, ανάλογα με τις παραμέτρους που λαμβάνει, όταν δηλώνουμε τα αντικείμενα μιας συγκεκριμένης κλάσης. Αυτή είναι μια δυνατότητα που μας δίνει η Java και ονομάζεται **υπερφόρτωση (overloading).**

Constructor Summary

`GreenfootImage(GreenfootImage image)` Create a GreenfootImage from another GreenfootImage.
`GreenfootImage(int width, int height)` Create an empty (transparent) image with the specified size.
`GreenfootImage(java.lang.String filename)` Create an image from an image file.
`GreenfootImage(java.lang.String string, int size, java.awt.Color foreground, java.awt.Color background)` Creates an image with the given string drawn as text using the given font size, with the given foreground color on the given background color.
`GreenfootImage(java.lang.String string, int size, java.awt.Color foreground, java.awt.Color background, java.awt.Color outline)` Creates an image with the given string drawn as text using the given font size, with the given foreground color on the given background color.

Εικόνα 3.7.4 Οι κατασκευαστές της κλάσης GreenfootImage

Μέσα στην κλάση *Counter* δηλώνουμε μια μεταβλητή αντικειμένου που την ονομάζουμε points για ν' αποθηκεύει κάθε φορά την τρέχουσα τιμή του σκορ.

Στον κατασκευαστή (constructor) της κλάσης Counter χρησιμοποιούμε το δεύτερο κατασκευαστή της κλάσης GreenfootImage (Εικόνα 3.7.4) για να δημιουργηθεί μια διαφανή εικόνα, με συγκεκριμένο πλάτος και συγκεκριμένο ύψος και την αποθηκεύουμε στην μεταβλητή με όνομα counterImage. Η εικόνα αυτή θα έχει πλάτος 200 pixels και ύψος 30 pixels. Κατόπιν, με την κλήση

της μεθόδου setImage, αναθέτουμε στον μετρητή (Counter) την εικόνα που δημιουργήσαμε προηγουμένως.

Θέτουμε αρχική τιμή στην μεταβλητή points την τιμή 0 και καλούμε στη συνέχεια μια νέα μέθοδο που ονομάζουμε update().

```
public class Counter extends Actor
{
   private int points;

   public Counter()
   {
      GreenfootImage counterImage=new GreenfootImage(200,30);
      setImage(counterImage);
      points = 0;
      update();
   }
}
```

Η μέθοδος update θα κάνει τις εξής ενέργειες:

1. Παίρνει την τρέχουσα εικόνα του Counter και την αποθηκεύει σε μια μεταβλητή τύπου GreenfootImage που ονομάζουμε img.
2. Καλεί τη μέθοδο clear() για να καθαρίζει την εικόνα του Counter από την προηγούμενη τιμή του.
3. Ορίζει το χρώμα του κειμένου του μετρητή (Counter), χρησιμοποιώντας τη μέθοδο setColor(java.awt.Color color) της κλάσης GreenfootImage. Το χρώμα του κειμένου (η παράμετρος της μεθόδου setColor) προέρχεται από την κλάση Color του πακέτου java.awt, άρα θα πρέπει, όπως είδαμε προηγουμένως, να δηλώσουμε την εισαγωγή της στον κώδικα στην αρχή της κλάσης Counter χρησιμοποιώντας τη λέξη import :

   ```
   import java.awt.Color;
   ```

4. Καλεί τη μέθοδο drawString με παραμέτρους το κείμενο και τις συντεταγμένες της οθόνης που αυτό θα εμφανίζεται.

```
public void update()
{
GreenfootImage img = getImage();
img.clear();
img.setColor(Color.BLACK);
img.drawString("Σκορ:  " + points, 4,20);
}
```

Πριν τρέξουμε την εφαρμογή μας, πρέπει να δημιουργήσουμε στον Κόσμο (FrogWorld), όπως κάναμε και με τα υπόλοιπα αντικείμενα του σεναρίου μας, το νέο αντικείμενο Counter (μετρητής) που θα χρειαστούμε για να κρατάμε το σκόρ, κάνοντας δεξί κλικ πάνω στην κλάση του, επιλέγοντας new Counter() και στη συνέχεια να το σύρουμε στη θέση που επιθυμούμε να εμφανίζεται μέσα στο πλέγμα του Κόσμου. Όπως κάναμε με το βάτραχο και τη μύγα, για να εμφανίζεται ο μετρητής αυτόματα κάθε φορά που εκτελούμε το σενάριο μας, πρέπει να τον δηλώσουμε μέσα στην κλάση FrogWorld. Γράψτε τις εντολές για τη δημιουργία του στον κατασκευαστή της FrogWorld, δίνοντας το όνομα ***metritis*** στη μεταβλητή που θα αποθηκεύσει το αντικείμενο και τις συντεταγμένες x και y του σημείου που θέλετε να εμφανίζεται ο μετρητής μέσα στο πλέγμα του Κόσμου.

Μια ακόμα μέθοδο, που θα χρειαστούμε μέσα στην κλάση Counter, είναι αυτή που θ' αναλάβει ν' αυξάνει τους πόντους του μετρητή και να καλεί στη συνέχεια τη μέθοδο update, ώστε ν' ανανεώνεται η εικόνα του Counter με τη νέα τιμή του σκορ. Αυτή τη μέθοδο θα την ονομάσουμε addScore και θα έχει ως παράμετρο, έναν ακέραιο αριθμό.

```
public void addScore(int value)
{
   points += value;
   update();
}
```

Ο μετρητής είναι έτοιμος. Μεταγλωττίζουμε και τρέχουμε το σενάριο. Το κείμενο του μετρητή εμφανίζεται στην οθόνη, αλλά δεν αυξάνεται η τιμή του όταν ο βάτραχος τρώει κάποια μύγα. Αυτό συμβαίνει γιατί τα δύο αντικείμενα βάτραχος (Frog) και μετρητής (Counter), δεν γνωρίζουν το ένα την ύπαρξη του άλλου και δεν μπορούν να επικοινωνήσουν απευθείας. Το μόνο αντικείμενο στον Κόσμο μας που γνωρίζει (έχει αναφορά) και στα δύο αντικείμενα, είναι ο Κόσμος του σεναρίου μας (FrogWorld).

Αυτό που πρέπει να γίνει είναι να γραφεί ο κώδικας, ώστε όταν ο βάτραχος τρώει μια μύγα να μπορεί να στέλνει ένα μήνυμα στο αντικείμενο Counter για να αυξάνει το σκορ του και να το εμφανίζει. Περνώντας μια αναφορά του αντικειμένου Counter στον κατασκευαστή του αντικειμένου βάτραχος, τα δύο αντικείμενα θα μπορούν να επικοινωνήσουν.

Μέχρι τώρα δεν έχουμε ορίσει κάποιο κατασκευαστή για την κλάση Frog και όπως είδαμε στο κεφάλαιο 3.2, εκτελείται ο default κατασκευαστής του αντικειμένου. Θα δημιουργήσουμε ένα κατασκευαστή για την κλάση Frog που θα εξυπηρετεί τις ανάγκες μας, αφήνοντας προς το παρόν το σώμα του χωρίς κάποια εντολή.

```
public Frog(Counter score)
{
}
```

Πηγαίνουμε στον κατασκευαστή της κλάσης FrogWorld και στο σώμα του αλλάζουμε την εντολή για τη δημιουργία του βατράχου με την παρακάτω, ώστε να δημιουργεί το αντικείμενο βάτραχος με το νέο κατασκευαστή που ορίσαμε προηγουμένως:

Frog kermit= new Frog (metritis);

Η τελική μορφή της κλάσης FrogWorld :

```
public FrogWorld()
  {
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
    Fly myga = new Fly();
    Fly myga1= new Fly();
    Fly myga2= new Fly();
    addObject(myga, 200, 200 );
    addObject(myga1, 443, 160);
    addObject(myga2, 398, 335);

    Counter metritis = new Counter();
    addObject(counter, 590, 390);

    Frog kermit= new Frog(metritis);
    addObject( kermit, 100, 100 );
```

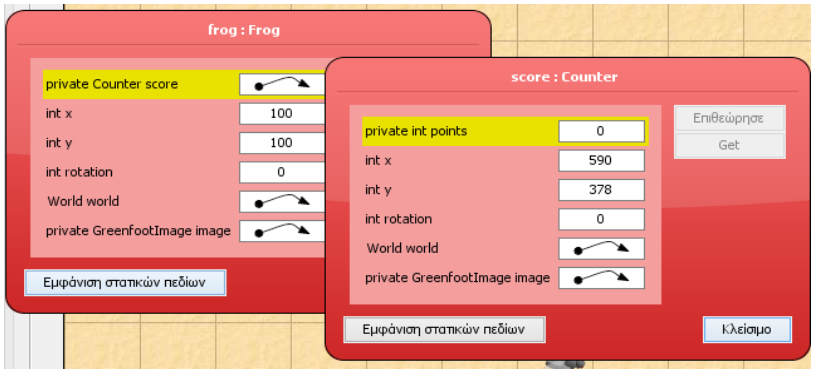
```
}
```

Ο μετρητής counter δημιουργήθηκε και η αναφορά του πέρασε στο αντικείμενο βάτραχος. Τώρα το αντικείμενο βάτραχος «γνωρίζει» το αντικείμενο Counter, με το οποίο σχετίζεται και επιτρέπεται η αποστολή μηνυμάτων από το αντικείμενο βάτραχος προς το αντικείμενο Counter (όχι όμως και το ανάποδο, δεδομένου ότι η συσχέτιση έχει μονόδρομη κατεύθυνση).

Επιστρέφουμε στην κλάση Frog για να γράψουμε τις εντολές (μηνύματα) που θα στέλνει το αντικείμενο βάτραχος προς το αντικείμενο Counter ώστε αυτό ν' αυξάνει την τιμή του.

Πρώτα πρέπει να δημιουργήσουμε μια μεταβλητή, με όνομα score, που θ' αναφέρεται σε ένα αντικείμενο τύπου Counter, όχι όμως απευθείας. Στην μεταβλητή score είναι αποθηκευμένη η διεύθυνση στη μνήμη όπου βρίσκονται τα δεδομένα του αντικειμένου. Αυτή η διεύθυνση ονομάζεται **αναφορά (reference).** Έτσι θα δώσουμε την εντολή: private Counter score;

Στην Εικόνα 3.7.5 βλέπουμε τις ιδιότητες του βατράχου και φαίνεται καθαρά στην 1η γραμμή η αναφορά στο αντικείμενο τύπου Counter. Πατώντας στο ***Επιθεώρησε*** μας ανοίγει το δεύτερο παράθυρο απ' όπου βλέπουμε τις ιδιότητες του.



Εικόνα 3.7.5 Ιδιότητες του αντικειμένου Frog

Κατόπιν μέσα στο σώμα του κατασκευαστή της κλάσης Frog που δημιουργήσαμε νωρίτερα, αναθέτουμε την παράμετρο (metritis) σ 'αυτό το πεδίο (score). Ο λόγος που το κάνουμε αυτό οφείλεται στο ότι η παράμετρος metritis θα πάψει να υπάρχει, μόλις η εκτέλεση του προγράμματος προχωρήσει στην επόμενη μέθοδο. Γι' αυτό την αποθηκεύουμε σ' ένα πεδίο του αντικειμένου βάτραχος κι έτσι έχουμε μόνιμη αναφορά στο αντικείμενο Counter μέσα από το αντικείμενο βάτραχος.

Η κλάση Frog γίνεται:

```
public class Frog extends Animal
{
   private Counter score;
   public Frog (Counter metritis)
   {
          score= metritis ;
   }
public void eatFlies()
{
   if (isTouching(Fly.class))
    {
      removeTouching(Fly.class);
      score.addScore(1);
       }
}
```

Τώρα η μεταβλητή fliesEaten δεν μας είναι πια απαραίτητη και μπορούμε να τη διαγράψουμε, αφού τον αριθμό των μυγών που τρώει ο βάτραχος τον κρατάει ο μετρητής που δημιουργήσαμε.

3.8. Μέθοδοι και Κληρονομικότητα

14η εργαστηριακή άσκηση:

Θα προσθέσουμε ένα ακόμα αντικείμενο στον Κόσμο, μια πασχαλίτσα (ladybug). Θα ονομάσουμε την κλάση της *Bug* και θα της δώσουμε την εικόνα ladybug1.png. Η κλάση αυτή θα είναι υποκλάση της Animal.

Στη συνέχεια ορίζουμε την μέθοδο act της κλάσης Animal έτσι ώστε το αντικείμενο να κινείται πρώτα πάνω και μετά δεξιά. Η μέθοδος act δεν ορίζεται ξανά στην κλάση Bug, οπότε τα αντικείμενα της Bug διατηρούν τη συμπεριφορά της μεθόδου act της Animal. Σύμφωνα με την ορολογία του αντικειμενοστρεφούς προγραμματισμού η κλάση Bug κληρονομεί από την Animal τη μέθοδο act.

```
import greenfoot.*;
public class Animal extends Actor
{
    public void act()
    {
        move(100);
        turn(90);
        move(100);
    }
}
```

```
import greenfoot.*;

public class Bug extends Animal
{

}
```

Αν τώρα δώσουμε την εντολή act στην πασχαλίτσα αυτή θα κινηθεί νοτιοανατολικά. Στη συνέχεια, ορίζουμε τη μέθοδο act και στην κλάση Bug, χωρίς να πειράξουμε τον κώδικα της κλάσης Animal, έτσι ώστε η πασχαλίτσα να εκτελεί τα ίδια βήματα αλλά με κατεύθυνση βορειοδυτικά.

```
import greenfoot.*;

public class Bug extends Animal
{
  public void act()
  {
    turn(-90);
    move(100);
    turn(-90);
    move(100);
  }
}
```

Τώρα η πασχαλίτσα κινείται βορειοδυτικά, γιατί υπερισχύει η μέθοδος act της κλάσης Bug στην οποία ανήκει, από αυτή της κλάσης Animal.

Το ίδιο συμβαίνει με τις κλάσεις Frog και Fly, όπου υπερισχύει η μέθοδος act όπως την ορίσαμε μέσα στις δικές τους κλάσεις.

Γενικά, όταν δημιουργούμε μια κλάση, ως υποκλάση μιας άλλης, εκτός από τις νέες μεθόδους που προσθέτουμε, επιλέγουμε και ποιες από τις υπάρχουσες μεθόδους θα ορίσουμε ξανά, ώστε να ενεργούν με διαφορετικό τρόπο στην υποκλάση. Με αυτό τον τρόπο εξειδικεύεται η συμπεριφορά των αντικειμένων της νέας κλάσης.

15η εργαστηριακή άσκηση:

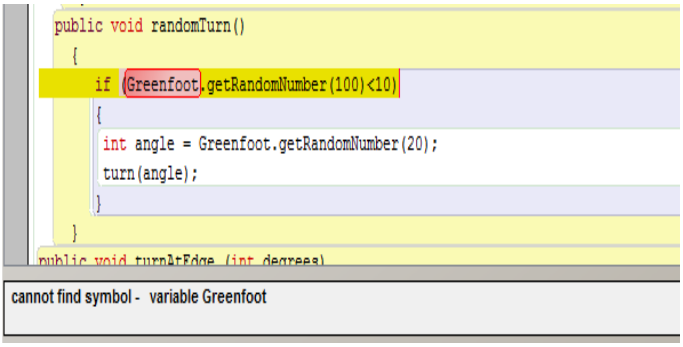
Ας κάνουμε μια μικρή επανάληψη των όσων μάθαμε μέχρι τώρα.

Θα ανοίξουμε την κλάση Frog του σεναρίου μας και θα διαβάσουμε τον κώδικα της. Στην πρώτη γραμμή βλέπουμε την εντολή: import greenfoot.*;

Αυτή η εντολή κάνει τη βιβλιοθήκη των κλάσεων του greenfoot διαθέσιμη για χρήση στην δική μας κλάση (Frog) και πληροφορεί συγχρόνως το μεταγλωττιστή γιά το πού θα βρει τις κλάσεις που θα χρησιμοποιήσουμε στον κώδικα. Ο αστερίσκος στο τέλος δηλώνει ότι θέλουμε να είναι διαθέσιμες όλες τις κλάσεις της βιβλιοθήκης greenfoot κι όχι κάποια συγκεκριμένη κλάση.

Η εντολή import πρέπει να προηγείται όλων των δηλώσεων κλάσεων.

Αν διαγράψουμε αυτή την εντολή και πατήσουμε μεταγλώττιση, τότε θα εμφανιστεί ένα μήνυμα λάθους στη μέθοδο randomTurn, στο σημείο που γίνεται η κλήση της μεθόδου getRandomNumber, η οποία ανήκει σε μια άλλη κλάση, την κλάση Greenfoot της βιβλιοθήκης του greenfoot. Αφού δεν έχουμε δηλώσει στην αρχή ότι θα χρησιμοποιήσουμε μεθόδους που ανήκουν στη βιβλιοθήκη του greenfoot, ο μεταγλωττιστής δεν αναγνωρίζει την εντολή που γράφουμε (Εικόνα 3.8.1).



Εικόνα 3.8.1 Μήνυμα λάθους σε κλήση μεθόδου άλλης κλάσης

Αμέσως μετά ακολουθούν τα σχόλια που δίνουν μια μικρή περιγραφή της κλάσης. Είδαμε ότι τα σχόλια ξεκινούν με /** και τελειώνουν με */. Μπορούμε όμως να χρησιμοποιήσουμε τη διπλή ανάποδη κάθετο (//) για να γράψουμε σχόλια που καταλαμβάνουν μόνο μια γραμμή και μπορούμε να βάλουμε σχόλια μιας γραμμής σε οποιαδήποτε γραμμή του κώδικα.

Ακολουθεί η δήλωση της κλάσης που συντάσσεται συνήθως σύμφωνα με τον εξής κανόνα:

public class «όνομα κλάσης» extends «όνομα υπερκλάσης»

{

Πεδία

Κατασκευαστές

Μέθοδοι

}

Τις λέξεις public class θα τις γράφουμε πάντα σε κάθε κλάση που δημιουργούμε. Η λέξη public ονομάζεται **προσδιοριστής πρόσβασης** και θα μιλήσουμε γι' αυτή αναλυτικότερα στο επόμενο κεφάλαιο. Ακολουθεί το όνομα της κλάσης που το ορίζουμε εμείς. Κατόπιν μπαίνει η λέξη extends και το όνομα της υπερκλάσης, που μας πληροφορεί ότι η κλάση που δημιουργούμε είναι υποκλάση της υπερκλάσης που δηλώνουμε μετά τη λέξη extends.

Τέλος ακολουθεί το ζευγάρι των αγκύλων { και } που ορίζουν το σώμα της κλάσης και εκεί μέσα δηλώνονται οι μέθοδοι.

3.9 Καθοδήγηση από γεγονότα

Όλες οι σύγχρονες εφαρμογές απαιτούν την είσοδο κάποιων δεδομένων από τον χρήστη ή τον έλεγχο (καθοδήγηση) από τον χρήστη κατά την εκτέλεση τους. Αυτό γίνεται κυρίως μέσω του

πληκτρολογίου και του ποντικιού. Ειδικά, τα παιχνίδια που ο βαθμός αλληλεπίδρασης με τον χρήστη είναι πολύ υψηλός, ο χειρισμός των μέσων εισόδου είναι κομβικής σημασίας. Αυτό γίνεται μέσα από τον χειρισμό γεγονότων που προκαλούνται όπως το πάτημα ενός πλήκτρου ή η κίνηση του ποντικιού πάνω από ένα αντικείμενο.

Αν μελετήσουμε την τεκμηρίωση της βιβλιοθήκης του Greenfoot θα βρούμε διάφορες μεθόδους για τον χειρισμό αυτών των γεγονότων όπως είναι οι παρακάτω:

```
public static String getKey()
```
Επιστρέφει το όνομα του πλήκτρου που πατήθηκε τελευταία. Αν δεν έχει πατηθεί κάποιο πλήκτρο, από την τελευταία φορά που κλήθηκε, η getKey επιστρέφει null.

```
public static boolean isKeyDown(String keyName)
```
Ελέγχει αν έχει πατηθεί το πλήκτρο με όνομα keyName και επιστρέφει αντίστοιχα true ή false.

```
public static boolean mouseClicked(Object obj)
```
Ελέγχει αν έχει γίνει κλικ με το ποντίκι στο αντικείμενο obj και επιστρέφει αντίστοιχα true ή false. Αν αντί για obj δώσουμε την τιμή null τότε επιστρέφεται true, αν έχει γίνει κλικ οπουδήποτε.

```
public static MouseInfo getMouseInfo()
```
Επιστρέφει ένα αντικείμενο *MouseInfo* το οποίο περιέχει πληροφορίες για την κατάσταση του ποντικιού. Με την κλήση των μεθόδων *getX(), getY()* ξέρουμε τις συντεταγμένες του σημείου όπου έγινε το κλικ ενώ, η *getActor()* μας επιστρέφει το αντικείμενο πάνω στο οποίο έγινε το κλικ.

16η εργαστηριακή άσκηση:

Παρακάτω, δίνουμε ένα παράδειγμα ελέγχου των κινήσεων του βατράχου μέσω των πλήκτρων και του ποντικιού. Εδώ φαίνεται και η σημασία της μεθόδου act στην εκτέλεση του προγράμματος. Όταν πατήσουμε το ***Εκκίνηση*** η act επαναλαμβάνεται συνέχεια. Έτσι μπορούμε να πατάμε πολλές φορές τα πλήκτρα ↑, ↓, ←, → για να μετακινούμε το βάτραχο. Αν πατήσουμε το ποντίκι σε οποιοδήποτε σημείο μέσα στον Κόσμο, εμφανίζονται οι συντεταγμένες της θέσης που έγινε το κλικ.

Η μέθοδος randomTurn που έκανε το βάτραχο να στρίβει τυχαία μέσα στον Κόσμο δε μας χρειάζεται πια, γιατί την κίνηση του βατράχου θα την ελέγχει πλέον ο χρήστης. Οπότε διαγράφουμε τη μέθοδο randomTurn. Επίσης θα διαγράψουμε και τη μέθοδο turnAtEdge για τον ίδιο λόγο.

Δημιουργούμε μια νέα μέθοδο, με το όνομα checkKeys(), που ελέγχει αν πατήθηκε κάποιο πλήκτρο από το πληκτρολόγιο ή το κουμπί απο το ποντίκι. Η υπογραφή της μεθόδου είναι:
public void checkKeys().

Στο σώμα της μεθόδου ορίζουμε δύο ακέραιες μεταβλητές x και y που αποθηκεύουν την τρέχουσα θέση του βάτραχου στους άξονες x και y. Κατόπιν, προσθέτουμε τη δομή ελέγχου if που ελέγχει αν πατήθηκε κάποιο πλήκτρο και δίνει εντολή στο βάτραχο να κινηθεί προς την ανάλογη κατεύθυνση. Δηλαδή, αν πατηθεί το αριστερό πλήκτρο, η τιμή της θέσης του βατράχου στον άξονα x μειώνεται κατά 4 και ο βάτραχος μετακινείται απο την τρέχουσα θέση του 4 βήματα πίσω, όταν εκτελεστεί η μέθοδος setLocation. Ανάλογες ενέργειες πραγματοποιούνται και με το πάτημα των άλλων πλήκτρων.

Επίσης, προσθέτουμε και τη δομή ελέγχου if που ελέγχει αν πατήθηκε το κουμπί απο το ποντίκι και εμφανίζονται οι συντεταγμένες της θέσης που έγινε το κλικ.

```
import greenfoot.*;

public class Frog extends Animal
{
   public void checkKeys ()
   {
      int x = getX();
      int y = getY();
      if ( Greenfoot.isKeyDown("left") ) {
         x = x - 4;
      }
      else if ( Greenfoot.isKeyDown("right") ) {
         x = x + 4;
      }
      else if ( Greenfoot.isKeyDown("up") ) {
         y = y - 4;
      }
      else if ( Greenfoot.isKeyDown("down") ) {
         y = y + 4;
      }
      setLocation(x,y);

      if (Greenfoot.mouseClicked( null ) ) {
         MouseInfo info = Greenfoot.getMouseInfo();
         System.out.println( info.getX() + "  " + info.getY() );
      }
   }
}
```

Οι getX(), getY(), setLocation() είναι μέθοδοι της κλάσης Actor, που έχει κληρονομήσει η Frog.

Η μέθοδος checkKeys() θα καλείται μέσα από τη μέθοδο act της κλάσης Frog, ενώ θα πρέπει να διαγράψουμε την κλήση των μεθόδων randomTurn() και turnAtEdge () μέσα από την act, αφού η κίνηση του βατράχου θα ορίζεται πλέον από το πληκτρολόγιο.

Το επόμενο βήμα είναι ν' αλλάξουμε τον παραπάνω κώδικα και αντί να εμφανίζονται οι συντεταγμένες της θέσης που έγινε κλικ με το ποντίκι, να εξαφανίζεται το αντικείμενο που γίνεται κλικ πάνω του με το ποντίκι. Αυτό επιτυγχάνεται με τη χρήση μεθόδων που θα βρούμε μέσα στη βιβλιοθήκη του greenfoot.

Οι μέθοδοι που θα χρειαστούμε είναι:

- Απο την κλάση Greenfoot: η mouseClicked που επιστρέφει αληθής (True), αν έχει πατηθεί με το ποντίκι κάποιο αντικείμενο και η getMouseInfo, που επιστρέφει ένα αντικείμενο τύπου MouseInfo με πληροφορίες όπως, το αντικείμενο στο οποίο έγινε κλικ απο το ποντίκι και οι συντεταγμένες στις οποίες έγινε το πάτημα του πλήκτρου του ποντικιού.
- Απο την κλάση MouseInfo: η getActor, που επιστρέφει το αντικείμενο στο οποίο έγινε κλικ με το ποντίκι.
- Απο την κλάση World: η removeObject, που αφαιρεί ένα αντικείμενο από τον Κόσμο. Η υπογραφή της μεθόδου removeObject είναι: void removeObject(Actor object). Η μέθοδος δεν επιστρέφει καμία πληροφορία (void) και δέχεται σαν παράμετρο ένα αντικείμενο τύπου Actor. Το αντικείμενο στην παράμετρο της μεθόδου είναι αυτό που θα αφαιρεθεί από τον Κόσμο.
- Απο την κλάση Actor: η getWorld, που επιστρέφει τον Κόσμο στον οποίο βρίσκονται τα αντικείμενα.

Ο κώδικας που πρέπει να γράψετε είναι:

```
if (Greenfoot.mouseClicked( null ) ) {
    MouseInfo info = Greenfoot.getMouseInfo();
    Actor actor=info.getActor();
   getWorld().removeObject(actor);
}
```

3.10 Πίνακες

Ένας πίνακας είναι ένα αντικείμενο, που έχει την ικανότητα και την ιδιότητα να μπορεί να δέχεται και να αποθηκεύει, σε συνεχόμενες θέσεις στη μνήμη, έναν προκαθορισμένο αριθμό στοιχείων, με την προϋπόθεση ότι όλα ανήκουν στον ίδιο τύπο δεδομένων. Δηλαδή τα στοιχεία που θα αποθηκευτούν θα είναι όλα ακέραιοι αριθμοί, ή όλα δεκαδικοί, ή όλα χαρακτήρες, ή όλα αναφορά σε κάποια αντικείμενα της ίδιας κλάσης κτλ. Το μέγεθος του πίνακα καθορίζεται την ίδια χρονική στιγμή της δημιουργίας του και δεν επιτρέπεται καμία αλλαγή στο μέγεθος του μετά τον αρχικό ορισμό του. Με άλλα λόγια, ένας πίνακας δεν έχει την ικανότητα να αυξομειώνεται δυναμικά ανάλογα με τον αριθμό των στοιχείων του, κατά την διάρκεια εκτέλεσης ενός προγράμματος. Το όνομα ενός πίνακα πάντα συνοδεύεται από τα σύμβολα [και] για να ξεχωρίζει από τον ορισμό οποιασδήποτε άλλης μεταβλητής.

Στο παρακάτω σχήμα φαίνεται ένας πίνακας δέκα θέσεων με ακέραιους αριθμούς. Η αρίθμηση των θέσεων ενός πίνακα ξεκινάει από το 0, όπως συμβαίνει και στις περισσότερες γλώσσες προγραμματισμού.

Για να αναφερθούμε, σε κάθε ένα από τα στοιχεία που είναι αποθηκευμένα στον πίνακα, πρέπει να χρησιμοποιήσουμε τον ανάλογο αριθμό (index) που αντιπροσωπεύει τη θέση τους στον πίνακα.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	10	20	30	40	50	60	70	80	90

Ο παραπάνω πίνακας ορίζεται με τις παρακάτω ισοδύναμες εντολές:

```
int  a[ ]  = { 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 };
      ή
int  a[ ]  =  new int[10];
for ( int i = 0;  i < 10;  i++)  {
   a[ i ] = 10 * i;
}
```

Για να αναφερθούμε σε ένα στοιχείο του πίνακα, χρειάζεται να δηλώσουμε το όνομα του πίνακα και τη θέση αυτού του στοιχείου. Για παράδειγμα για να θέσουμε μια νέα τιμή στο στοιχείο στη θέση 2 του πίνακα, δηλαδή στο 3ο στοιχείο του πίνακα την τιμή 23 δίνουμε την εντολή:

```
a[ 2 ] = 23;
```

Αν πάλι θέλουμε να αυξήσουμε την τιμή του στοιχείου a[2] κατά 1 δίνουμε την εντολή

a[2]++; που είναι ισοδύναμη με την εντολή a[2] ← a[2] + 1

Δηλαδή μπορούμε να διαχειριζόμαστε κάθε στοιχείο ενός πίνακα όπως και μια μεταβλητή.

Προσοχή!!! Όπως φαίνεται και από το σχήμα, η πρώτη θέση ενός πίνακα είναι η θέση 0. Αυτό σημαίνει ότι όταν ορίζουμε έναν πίνακα 10 στοιχείων, αυτά βρίσκονται στις θέσεις από 0 έως 9.
Άρα το 4ο στοιχείο βρίσκεται στη θέση 3 του πίνακα και το 10ο στη θέση 9.
Γενικά το n-οστό στοιχείο βρίσκεται στη θέση n-1 του πίνακα, δηλαδή είναι το a[n-1] .
Όταν θέλουμε να διαβάσουμε, να τυπώσουμε ή να επεξεργαστούμε έναν πίνακα σαρώνουμε τα στοιχεία του χρησιμοποιώντας ένα for και αυτό επειδή σχεδόν πάντα γνωρίζουμε το πλήθος των στοιχείων του πίνακα.

Στη Java μπορούμε να ορίσουμε πολυδιάστατους πίνακες

```
int  matrix[ ][ ]  =  new int[10][10];
```

Μια σημαντική διαφορά των πινάκων στη java σε σχέση με άλλες γλώσσες προγραμματισμού, είναι ότι στη Java κάθε πίνακας έχει ένα πεδίο με όνομα length το οποίο περιέχει το μέγεθος του πίνακα. Για παράδειγμα η προηγούμενη δομή for μπορεί να γραφτεί και ως εξής:

```
for ( int i = 0;  i < a.length;  i++)  {
    a[ i ] = 10 * i;
}
```

Το παρακάτω τμήμα κώδικα υπολογίζει το άθροισμα όλων των στοιχείων του πίνακα

```
int S = 0;
for ( int i = 0;  i < a.length;  i++) {
    S = S + a[ i ];
}
String strArray[ ] = new String[10];   // Δεσμεύει δέκα θέσεις
```

Προσοχή!!! Οι δέκα θέσεις του πίνακα που δεσμεύονται αφορούν τη διεύθυνση μνήμης στην οποία βρίσκονται οι συμβολοσειρές. Αρχικά έχουν την τιμή null. Για να τους δώσουμε τιμή θα πρέπει να δημιουργήσουμε ξεχωριστά κάθε αντικείμενο strArray*[i]* χρησιμοποιώντας τον τελεστή *new* όπως στο παρακάτω παράδειγμα:

```
for (int i=0; i<strArray.length; i++)
   strArray[ i ] = new String( "Η θέση μου στον πίνακα είναι " + i );
```

Με αυτόν τον τρόπο αποθηκεύουμε σε κάθε θέση του πίνακα τη συμβολοσειρά που θέλουμε.

17η εργαστηριακή άσκηση:

Επιστρέφουμε στο σενάριο μας FrogEatFlies, όπου θ' αλλάξουμε τον κώδικα της μεθόδου checkKeys της προηγούμενης άσκησης, που περιέχει μια σειρά από if εντολές και θα την υλοποιήσουμε με διαφορετική λογική, χρησιμοποιώντας πίνακες.

Κατ' αρχήν δηλώνουμε στην αρχή της κλάσης Frog ένα πίνακα συμβολοσειρών με το όνομα pliktra, που αποθηκεύει τις διαφορετικές ονομασίες των πλήκτρων:

```
private  String[] pliktra={"up","right","down","left"};
```

Ας σημειώσουμε εδώ ότι το "up" αντιστοιχεί στον αριθμό (index) της θέσης 0 του πίνακα, ενώ το "left" αντιστοιχεί στον αριθμό (index) της θέσης 3 του πίνακα.

Έτσι, αντί να έχουμε τέσσερεις διαφορετικές δομές ελέγχου if, μπορούμε να καλούμε τον πίνακα pliktra μέσα σ' ένα βρόγχο for που θα περιέχει μόνο μια δομή επανάληψης if, ως εξής:

```
for (int i=0; i<pliktra.length; i++) {
      if (Greenfoot.isKeyDown( pliktra[i]) ) {
```

Τώρα το ερώτημα είναι, πως θα συνδέσουμε τον index για κάθε επιλογή πλήκτρου με την κατάλληλη κίνηση που πρέπει να κάνει ο βάτραχος;

Στην προηγούμενη άσκηση είδαμε ότι η κίνηση του βατράχου καθορίζεται απο την αλλαγή στην τιμή της συντεταγμένης x ή της y κάθε φορά. Στην κίνηση προς τ' αριστερά αφαιρούμε 4 απο το x, ενω στην κίνηση προς τα δεξιά προσθέτουμε 4 στο x. Το αντίστοιχο κάνουμε και με το y. Επίσης, όταν οι τιμές των x και y δεν αλλάζουν, είναι σαν να προσθέτουμε στην τιμή τους το μηδέν. Συνοψίζοντας τα παραπάνω, έχουμε διαφορετικές τιμές που πρέπει να προσθέσουμε στο x και στο y και αυτές διαφέρουν ανάλογα με την κατεύθυνση που πρέπει να κινηθεί ο βαάτραχος. Άρα, θα χρειαστούμε δύο πίνακες. Ένα πίνακα που αποθηκεύει τις τιμές που προσθέτουμε στο x, για κάθε κατεύθυνση και ένα πίνακα που αποθηκεύει τις τιμές που προσθέτουμε στο y, για κάθε κατεύθυνση.

Οι δύο πίνακες θα δηλωθούν κι αυτοί στην αρχή της κλάσης Frog, ως εξής:

```
private int[] xOffset={0,4,0,-4};
private int[] yOffset={-4,0,4,0};
```

Τέλος, μέσα στο σώμα της if θα καλούμε τη μέθοδο setLocation, που θα προσθέτει την κατάλληλη τιμή απο τους δύο πίνακες, στην τιμή των x και y αντίστοιχα.

Ο κώδικας της κλάσης Frog μετά την αλλαγή της μεθόδου checkKeys θα γίνει:

```
public class Frog extends Actor
{
    private  String[] pliktra={"up","right","down","left"};
    private int[] xOffset={0,4,0,-4};
    private int[] yOffset={-4,0,4,0};

    /**
     * Act - do whatever the Frog wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkKeys();

    }

    public void checkKeys ()
    {
        for (int i=0; i<pliktra.length; i++) {
            if (Greenfoot.isKeyDown(pliktra[i])) {

                setLocation(getX()+xOffset[i],getY()+yOffset[i]);
            }
        }

        /
        if (Greenfoot.mouseClicked( null ) ) {
            MouseInfo info = Greenfoot.getMouseInfo();
            Actor actor=info.getActor();
            getWorld().removeObject(actor);
        }
    }

}
```

3.11 Δραστηριότητες

Δραστηριότητα 1

Να επεκτείνετε την προηγούμενη εφαρμογή στο Greenfoot, έτσι ώστε ο χρήστης να ορίζει αρχικά πόσα αντικείμενα χρειάζεται, κάνοντας κλικ σε κατάλληλα σημεία στον Κόσμο, όπου και θα δημιουργούνται τα αντικείμενα. Τα αντικείμενα αυτά να αποθηκεύονται σε έναν πίνακα από αντικείμενα αυτού του τύπου, τον οποίο θα έχετε δημιουργήσει προηγουμένως. Όταν γεμίσει ο πίνακας να εμφανίζεται κατάλληλο μήνυμα στον χρήστη και να τερματίζεται η διαδικασία δημιουργίας.

Δραστηριότητα 2

Θέλουμε στο σενάριο μας να εμφανίζεται μια νέα μύγα στον Κόσμο, όταν κάποια τρώγεται από το βάτραχο. Αυτό γίνεται, αν το αντικείμενο βάτραχος, μπορεί να περάσει κάποιο μήνυμα στο αντικείμενο Κόσμος. Ο Κόσμος (FrogWorld), η περιοχή δηλαδή στην οθόνη που εκτυλίσσεται το σενάριο μας, είναι κι αυτός ένα αντικείμενο. Το μήνυμα που θα περάσει η κλάση Frog προς τον Κόσμο θα είναι: «τοποθέτησε μια νέα μύγα, μόλις κάποια εξαφανιστεί».

Δραστηριότητα 3

Μερικές φορές θέλουμε να απλοποιήσουμε τον κώδικά μας, ώστε να απαλλαγεί από επουσιώδεις λεπτομέρειες. Για παράδειγμα είδαμε ότι για να εμφανίσουμε ένα μήνυμα στην έξοδο πρέπει να χρησιμοποιήσουμε την εντολή:

```
System.out.println("Το μήνυμά μου");
```

Για να μην γράφουμε ολόκληρη την εντολή θα μπορούσαμε να ορίσουμε μια νέα μέθοδο writeln ως εξής:

```
public static void writeln(String message)
{
   System.out.println(message);
}
```

Με αυτόν τον τρόπο η παραπάνω εντολή γράφεται πλέον ως εξής:

```
writeln("Το μήνυμά μου");
```

η οποία φαίνεται πολύ πιο απλή, γιατί έχουμε αποκρύψει το γεγονός ότι καλούμε μια μέθοδο του αντικειμένου *out* της κλάσης *System*.

Να χρησιμοποιήσετε παρόμοια λογική για να ορίσετε αντίστοιχες μεθόδους, για εισαγωγή δεδομένων και εκτύπωση αποτελεσμάτων ή διαγνωστικών μηνυμάτων, απλοποιώντας τις αντίστοιχες μεθόδους της κλάσης *JOptionPane*.

Δραστηριότητα 4

Να υλοποιήσετε σε Java το παιχνίδι "Μάντεψε τον αριθμό". Στο παιχνίδι αυτό αρχικά, ο χρήστης δίνει το κάτω και το πάνω όριο των αριθμών και παράγει τυχαία έναν μυστικό αριθμό μεταξύ αυτών των ορίων. Στη συνέχεια, ξεκινάει το παιχνίδι το οποίο έχει ως εξής: Ο παίκτης μπορεί να κάνει τις εξής ερωτήσεις:

Είναι μικρότερος από ….
Είναι μεγαλύτερος από ….
Είναι ίσος με ….

Όπου …. Είναι ένας αριθμός που δίνει ο χρήστης. Το πρόγραμμά σας θα απαντά με ναι ή όχι και θα τερματίζει όταν ο παίκτης βρει τελικά τον αριθμό. Στο τέλος, θα εμφανίζει και πόσες ερωτήσεις χρειάστηκε να κάνει ο παίκτης

Δραστηριότητα 5

Υλοποιήστε το παραπάνω παιχνίδι "Μάντεψε τον αριθμό", αλλά τώρα ο παίκτης είναι αυτός που σκέφτεται τον αριθμό και ο υπολογιστής είναι αυτός που ψάχνει. Σκεφτείτε έναν αποδοτικό τρόπο για να βρει ο υπολογιστής γρήγορα τον μυστικό αριθμό.

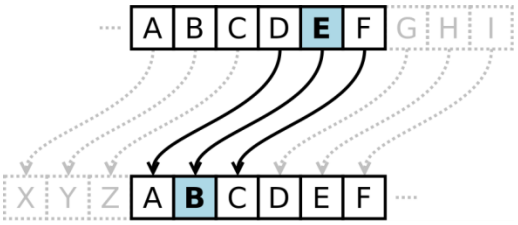
Δραστηριότητα 6

Γράψτε μια μέθοδο ***public String*** *reverseString(**String** word)* η οποία επιστρέφει τη λέξη word αντεστραμμένη, π.χ. αν δώσουμε Java θα πάρουμε avaJ.

Γράψτε μια μέθοδο ***public boolean*** *isPalindrome(**String** word)* η οποία ελέγχει αν η λέξη word είναι παλινδρομική (καρκινική), δηλαδή διαβάζεται το ίδιο και προς τις δυο κατευθύνσεις, π.χ. radar. Να χρησιμοποιήσετε τη μέθοδο reverseString .

Δραστηριότητα 7

Στον αλγόριθμο κρυπτογράφησης του Καίσαρα, όπως φαίνεται στο παρακάτω σχήμα κρυπτογράφησης, κάθε γράμμα αντιστοιχεί στο γράμμα που βρίσκεται έναν αριθμό θέσεων πίσω του στο αλφάβητο. Στο παρακάτω σχήμα το Ε κρυπτογραφείται ως Β, διότι το Β βρίσκεται 3 θέσεις πίσω του. Ο αριθμός 3 είναι η παράμετρος του αλγορίθμου.



Να υλοποιήσετε τη μέθοδο ***public String*** *encode(**String** plaintext, **int** shift)* η οποία δέχεται τη λέξη που θέλουμε να κρυπτογραφήσουμε και την παράμετρο shift. Για παράδειγμα, αν θέλουμε να κρυπτογραφήσουμε τη λέξη Java αλλάζοντας κάθε γράμμα με αυτό που βρίσκεται 3 θέσεις πίσω του, καλούμε τη μέθοδο ως εξής:

String *cipherText = encode("Java", 3)*

Στη συνέχεια, να υλοποιήσετε την παραπάνω μέθοδο έτσι ώστε, να υλοποιεί και αποκρυπτογράφηση. Στην περίπτωση αυτή θα πρέπει να δοθεί η παράμετρος κρυπτογράφησης με αρνητικό πρόσημο π.χ. –3.

Να σημειωθεί ότι η μετατόπιση των γραμμάτων είναι κυκλική, δηλαδή μετά το Ω είναι το Α.

Κεφάλαιο 4

Αντικειμενοστρεφής προγραμματισμός

4. Αντικειμενοστρεφής προγραμματισμός

Εισαγωγή

Υπάρχουν διάφορα είδη προγραμματισμού όπως ο συναρτησιακός (Lisp, Haskell), ο λογικός (Prolog) ο διαδικασιακός (Pascal) και ο αντικειμενοστρεφής προγραμματισμός. Κάθε προγραμματιστικό υπόδειγμα έχει τα πλεονεκτήματά του. Ωστόσο αυτή τη στιγμή το επικρατέστερο προγραμματιστικό υπόδειγμα είναι το αντικειμενοστρεφές. Όλες οι σύγχρονες γλώσσες προγραμματισμού υποστηρίζουν τα βασικά χαρακτηριστικά του αντικειμενοστρεφούς μοντέλου, στο οποίο η εφαρμογή σχεδιάζεται με βάση τη δομή των δεδομένων (data – centric) τα οποία αναπαριστώνται με τη μορφή αντικειμένων διάφορων κλάσεων. Ο αντικειμενοστρεφής προγραμματισμός (object-oriented programming), ή ΑΠ, εμφανίστηκε στα τέλη της δεκαετίας του 1960 και καθιερώθηκε κατά τη δεκαετία του 1990, αντικαθιστώντας σε μεγάλο βαθμό το παραδοσιακό υπόδειγμα του δομημένου προγραμματισμού. Στον ΑΠ οι διαδικασίες που ορίζονται για τον χειρισμό και την επεξεργασία των δεδομένων είναι τμήμα μιας δομής δεδομένων που τα περιβάλλει ως αυτόνομη οντότητα. Αυτή η δομή δεδομένων καλείται αντικείμενο και αποτελεί πραγματικό στιγμιότυπο στη μνήμη ενός σύνθετου, τύπου δεδομένων ονόματι κλάση. Η κλάση ορίζει τόσο τα δεδομένα όσο και τις διαδικασίες οι οποίες επιδρούν επάνω τους. Αυτή υπήρξε η πρωταρχική καινοτομία του ΑΠ.

Τα πιο σημαντικά χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού που θα παρουσιαστούν σε αυτή την ενότητα είναι:

- Ενθυλάκωση (*Encapsulation*)
- Κληρονομικότητα (Inheritance)
- Πολυμορφισμός (Polymorphism)

Οι πιο χαρακτηριστικές γλώσσες του αντικειμενοστρεφούς υποδείγματος είναι η Simula, η SmallTalk, η C++ και η Java την οποία μελετάμε σε αυτό το μάθημα.

Στόχοι

Οι μαθητές να μπορούν να:

- Ορίζουν τις δικές τους κλάσεις
- Επεκτείνουν ήδη υπάρχουσες κλάσεις
- Αναγνωρίζουν την επαναχρησιμοποίηση κώδικα μέσα από την κληρονομικότητα
- Εξηγούν τη διαφορά μεταξύ υπερφόρτωσης και πολυμορφισμού
- Δημιουργούν τα δικά τους πακέτα ως βιβλιοθήκες λογισμικού
- Αναγνωρίζουν τα πλεονεκτήματα του αντικειμενοστρεφούς προγραμματισμού

Ενότητες κεφαλαίου

- Ορισμός σύνθετων τύπων / κλάσεων
- Κατασκευαστές και Καταστροφείς
- Προσδιοριστές Πρόσβασης
- Κληρονομικότητα
- Υπερφόρτωση
- Πολυμορφισμός
- Διασυνδέσεις (Interfaces)
- Πακέτα (packages)
- Η αξία της αφαίρεσης

4.0 Η βιβλιοθήκη JTF της ACM

Η βιβλιοθήκη JTF (Java Task Force) της ACM έχει αναπτυχθεί για εκπαιδευτική χρήση και έχει στόχο να διευκολύνει την ανάπτυξη εφαρμογών στη γλώσσα Java. Η χρήση της βιβλιοθήκης μας δίνει τη δυνατότητα να χρησιμοποιήσουμε έτοιμα αντικείμενα και μεθόδους για είσοδο/έξοδο δεδομένων, σχεδιασμό γραφικής διεπαφής ή χειρισμό γεγονότων, αποφεύγοντας επουσιώδεις τεχνικές λεπτομέρειες. Μπορείτε να κατεβάσετε τη βιβλιοθήκη από τον δικτυακό τόπο http://jtf.acm.org. Στον ίδιο δικτυακό τόπο υπάρχουν και οι απαραίτητοι οδηγοί για τη χρήση της βιβλιοθήκης.

Παρακάτω φαίνεται ένα απλό πρόγραμμα με χρήση της βιβλιοθήκης της ACM.

```
import acm.program.*;

public class Test extends Program {

        public void run( ) {
                println("Το πρώτο μου .... απλό πρόγραμμα");
        }
}
```

Με την εκτέλεση του προγράμματος εκτελείται η μέθοδος run. Η κλάση Test κληρονομεί από την Program της ACM διάφορα πακέτα που περιλαμβάνουν μεθόδους εισόδου/εξόδου όπως είναι η println, για να κάνουμε τη ζωή μας πιο εύκολη.

Η προσθήκη της JTF στο Eclipse γίνεται πολύ εύκολα. Αφού δημιουργήσετε ένα νέο project πρέπει να προσθέσετε το acm.jar στις εξωτερικές βιβλιοθήκες (external libraries/jars). Αν δεν χρησιμοποιείτε το Eclipse αλλά προτιμάτε να δουλεύετε από τη γραμμή εντολών, τότε θα πρέπει να συμπεριλάβετε τη βιβλιοθήκη και κατά τη μεταγλώττιση και κατά την εκτέλεση.

4.1. Ορισμός Σύνθετων τύπων – κλάσεων

Με εξαίρεση τους βασικούς τύπους της Java (int, float, boolean, κλπ) τα πάντα είναι αντικείμενα κάποιας κλάσης. Η κλάση είναι ουσιαστικά ένας σύνθετος τύπος δεδομένων εφοδιασμένος με τις λειτουργίες χειρισμού των δεδομένων. Εκτός από τις κλάσεις που έρχονται με τη γλώσσα μπορούμε και εμείς να ορίσουμε τις δικές μας κλάσεις και να δημιουργήσουμε τα δικά μας αντικείμενα όπως κάναμε στις προηγούμενες ενότητες με τις κλάσεις Car και Frog. Οι κλάσεις αυτές μπορούν να θεωρηθούν σαν καλούπια τα οποία δίνουν ζωή σε όμοια αντικείμενα τα οποία είναι το ένα αντίγραφο του άλλου. Τα αντικείμενα αυτά τα λέμε πολλές φορές και στιγμιότυπα της κλάσης, με το σκεπτικό ότι η κλάση είναι απλά μια αφαιρετική αναπαράσταση των αντικειμένων ενώ τα αντικείμενα είναι αυτά που έχουν υπόσταση και υλοποιούν την αναπαράσταση αυτή.

Παρακάτω δίνουμε τον ορισμό ενός σύνθετου τύπου δεδομένων / κλάσης Person για την αποθήκευση των στοιχείων όλου το έμψυχου δυναμικού ενός σχολείου (μαθητές, καθηγητές, διοικητικό προσωπικό).

```
        public class Person {
            public String firstname;
            public String lastname;
            public int id;
        }
```

Όλοι έχουν στη βάση δεδομένων του σχολείου όνομα, επώνυμο και έναν μοναδικό αριθμό μητρώου (id).

Να σημειωθεί ότι κάθε κλάση πρέπει να ορίζεται σε ξεχωριστό αρχείο με το ίδιο όνομα και κατάληξη .java, π.χ. Person.java .

4.2. Κατασκευαστές και καταστροφείς

Κάθε κλάση πρέπει να έχει τουλάχιστον μία μέθοδο η οποία εκτελείται όταν δημιουργείται ένα αντικείμενο της (στιγμιότυπό της). Αυτές οι μέθοδοι λέγονται κατασκευαστές (constructors) της κλάσης. Ο κατασκευαστής έχει την ευθύνη δυο λειτουργιών: **1)** να δεσμεύσει την απαραίτητη μνήμη για το αντικείμενο που δημιουργείται **2)** να αρχικοποιήσει όλες τις μεταβλητές ή τα αντικείμενα που είναι ενσωματωμένα.

Η δήλωση του κατασκευαστή μοιάζει αρκετά με τη δήλωση της μεθόδου, αλλά έχει κάποιες διαφορές:

- Δεν έχει ποτέ τύπο επιστροφής (π.χ. int, boolean, κ.α) ούτε όμως συντάσσεται με τη λέξη κλειδί void μετά τη λέξη public.
- Το όνομα του είναι πάντα το ίδιο με το όνομα της κλάσης

Μπορούν να υπάρχουν διάφορες εκδόσεις κατασκευαστών για μια κλάση που διαφέρουν στο πλήθος των παραμέτρων που δέχονται, αλλά για τη δημιουργία ενός αντικειμένου (στιγμιότυπου) μπορούμε να χρησιμοποιήσουμε μόνο έναν από αυτούς. Ο κατασκευαστής που δεν παίρνει κανένα όρισμα λέγεται **προκαθορισμένος κατασκευαστής** (default constructor) της κλάσης και δημιουργείται αυτόματα αν εμείς δεν ορίσουμε κανέναν κατασκευαστή.

Ας δούμε διάφορα είδη κατασκευαστών για την κλάση Person:

```
public Person() {
  firstname = “Γιάννης”;
  lastname  = “Αγιάννης”;
  id = 0;
}
```

Τα δεδομένα firstname, lastname, id λέγονται πεδία της κλάσης Person. Κάθε αντικείμενο της κλάσης Person εμπεριέχει τα τρία αυτά πεδία.

Η παρακάτω εντολή δεν δημιουργεί κανένα αντικείμενο, αλλά μόνο δυο θέσεις στη μνήμη στην οποία θα καταχωρηθούν οι διευθύνσεις στη μνήμη των αντικειμένων alan και alonzo όταν αυτά δημιουργηθούν. Στην Java αυτά ονομάζονται αναφορές στα αντικείμενα.

```
Person alan, alonzo;
```

Για να δημιουργήσουμε τα αντικείμενα πρέπει να καλέσουμε τον κατασκευαστή της κλάσης χρησιμοποιώντας τη λέξη *new*. Η δήλωση αρχίζει με τη λέξη new που ακολουθείται από τον κατασκευαστή του αντικειμένου.

```
alan   = new Person();
alonzo = new Person();
```

Τώρα έχουν δημιουργηθεί δυο αντικείμενα με τις ίδιες προκαθορισμένες τιμές. Αν θέλουμε να αλλάξουμε τα δεδομένα χρησιμοποιούμε τον τελεστή τελεία (dot-notation) . για πρόσβαση στα δεδομένα:

```
alan.firstname = “Alan”;
alan.lastname = “Turing”;
alan.id = 1;
```

Αντί να δώσουμε τιμές στα αντικείμενα με αυτόν τον τρόπο θα ήταν πολύ καλύτερα αν αυτές οι τιμές δίνονταν από τον κατασκευαστή. Μπορούμε να ορίσουμε έναν άλλο κατασκευαστή ο οποίος να δέχεται σαν ορίσματα τις τιμές των πεδίων:

```
public Person(String firstname, String lastname, int id) {
    this.firstname = firstname;
    this.lastname  = lastname;
    this.id = id;
  }
```

Η δεσμευμένη λέξη **this** είναι μια αναφορά στο αντικείμενο που δημιουργείται. Παρατηρήστε ότι η παράμετρος firstname του κατασκευαστή έχει το ίδιο όνομα με το πεδίο firstname της κλάσης. Πως θα ξέρει ο μεταγλωττιστής ποια από τις δυο μεταβλητές εννοούμε; Στην πραγματικότητα η μεταβλητή που θα "επικρατήσει" είναι η παράμετρος η οποία επικαλύπτει το πεδίο της κλάσης. Για να ξεχωρίσουμε τα πεδία της κλάσης αναφερόμαστε σε αυτά μέσω της αναφοράς this. Η αναφορά this είναι πολύ χρήσιμη γιατί ουσιαστικά το ίδιο αντικείμενο περιέχει μια αναφορά στον εαυτό του. Για να δημιουργήσουμε τώρα το ίδιο ακριβώς αντικείμενο με προηγουμένως δίνουμε την παρακάτω εντολή:

```
Person alan   = new Person("Alan", "Turing", 1);
```

Σε ορισμένες γλώσσες προγραμματισμού όπως η C++ και η Object PASCAL σε αντιδιαστολή με τους κατασκευαστές (constructors) υπάρχουν οι καταστροφείς (destructors) οι οποίοι εκτελούν ακριβώς τα αντίστροφα βήματα, δηλαδή απελευθερώνουν μνήμη αν έχει δεσμευτεί, κλείνουν αρχεία που έχουν ανοίξει και γενικότερα αποδεσμεύουν ομαλά τους υπολογιστικούς πόρους που χρησιμοποιεί το αντικείμενο αφού δεν χρειάζονται πλέον. Στην Java δεν υπάρχουν καταστροφείς διότι η ευθύνη της αποδέσμευσης δεν βαρύνει τον προγραμματιστή αλλά τη γλώσσα. Το έργο αυτό το έχει αναλάβει ο συλλέκτης απορριμμάτων (Garbage Collector) ο οποίος ελέγχει τη μνήμη για αντικείμενα που δεν χρησιμοποιούνται πλέον και πρέπει να "καταστραφούν". Ο συλλέκτης απορριμμάτων είναι μέρος της εικονικής μηχανής (Virtual Machine) της Java η οποία μόλις διαπιστώσει ότι ο σωρός (heap) της μνήμης κοντεύει να γεμίσει ενεργοποιεί το συλλέκτη απορριμμάτων. Έτσι ο προγραμματιστής δε χρειάζεται να ανησυχεί για το πότε και αν θα ελευθερώσει ένα συγκεκριμένο τμήμα της μνήμης.

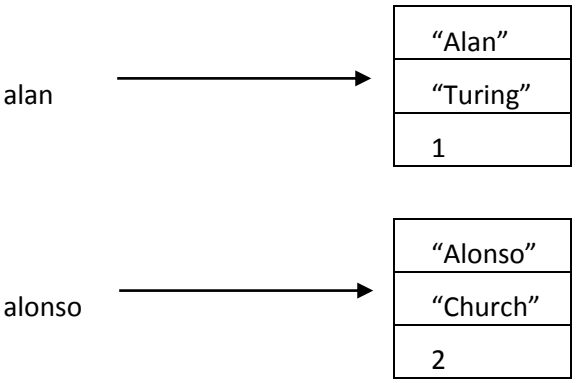
Ωστόσο υπάρχουν περιπτώσεις που θα θέλαμε πριν αποδεσμευτεί ένα αντικείμενο να κάνουμε κάποιες σημαντικές ενέργειες, όπως να κλείσουμε ομαλά κάποια αρχεία. Για αυτόν τον σκοπό υπάρχει στην Java η μέθοδος *finalize*() η οποία υπάρχει σε κάθε κλάση και την οποία μπορούμε να υλοποιήσουμε μόνο αν το θεωρούμε απολύτως απαραίτητο. Η finalize καλείται από τον συλλέκτη απορριμμάτων λίγο πριν την καταστροφή του αντικειμένου. Η υλοποίηση της finalize αναφέρεται ως finalizer.

Αναφορές σε αντικείμενα

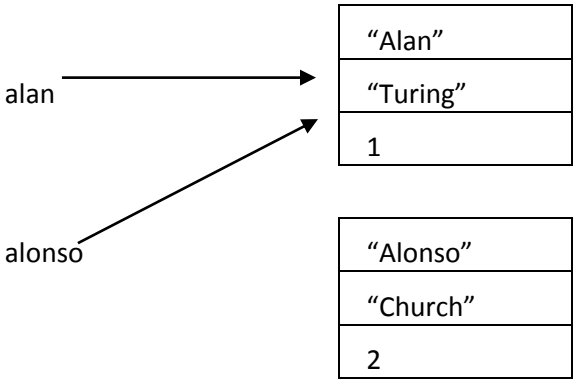
Στο κεφάλαιο 3 όταν περιγράψαμε το πέρασμα παραμέτρων κατά τιμή είπαμε ότι όταν περνάμε ως παράμετρο σε μια μέθοδο ένα αντικείμενο δεν δημιουργείται αντίγραφό του αλλά αντίγραφο της μεταβλητής που περιέχει τη διεύθυνση μνήμης όπου βρίσκεται το αντικείμενο. Αυτό φαίνεται καλύτερα στο παρακάτω παράδειγμα:

```
Person alan   = new Person("Alan", "Turing", 1);
Person alonso = new Person("Alonso","Church",2);
alonso = alan;
alonso.id = 496;
```

Μετά την εκτέλεση των παραπάνω εντολών έχει αλλάξει το πεδίο id του αντικειμένου alan; Ας δούμε το παρακάτω επεξηγηματικό παράδειγμα. Αρχικά έχουμε δημιουργήσει τα δυο αντικείμενα alan και alonso της κλάσης Person. Η κατάσταση της μνήμης φαίνεται στο παρακάτω σχήμα:



Στη συνέχεια αφού δώσουμε στην εντολή alonso = alan η νέα κατάσταση στη μνήμη είναι η εξής:



Δηλαδή ο Alonso δείχνει πλέον στη θέση μνήμης του αντικειμένου alan. Τώρα φαίνεται ότι οι μεταβλητές alan και alonso δεν είναι οι ίδιες αντικείμενα αλλά δείκτες στη μνήμη όπου βρίσκονται τα πραγματικά αντικείμενα. Μέσω αυτών των δεικτών ή αναφορών όπως λέγονται στην java μπορούμε να στέλνουμε μηνύματα στα αντικείμενα καλώντας π.χ. τις μεθόδους τους. Φανταστείτε την αναφορά σαν ένα τηλεχειριστήριο και το αντικείμενο σαν μια τηλεόραση. Μπορούμε να μετακινούμαστε σε ένα δωμάτιο αλλά δεν χρειάζεται να μεταφέρουμε μαζί μας την τηλεόραση παρά μόνο το τηλεχειριστήριο μέσω του οποίου μπορούμε να χειριζόμαστε την τηλεόραση. Το μοντέλο αυτό προσομοιάζει στο αντίστοιχο μοντέλο μνήμης της Java αν δεχτούμε ότι μπορούμε να αλλάξουμε κανάλι στην τηλεόραση μόνο από το τηλεχειριστήριο και όχι από κοντά, οπότε αν χαθεί η χαλάσει το τηλεχειριστήριο δεν ελέγχουμε την τηλεόραση.

Στο παραπάνω παράδειγμα το δεύτερο αντικείμενο δεν έχει πλέον κάποια αναφορά που να δείχνει σε αυτό, οπότε δεν μπορούμε πια να το χρησιμοποιήσουμε. Θεωρείται από την Java σαν μια περιοχή μνήμης που πρέπει να αποδεσμευτεί κάτι το οποίο θα γίνει μόλις την εντοπίσει ο **συλλέκτης απορριμμάτων (garbage collector),** το ειδικό πρόγραμμα της εικονικής μηχανής της Java που έχει σκοπό την εκκαθάριση της μνήμης.

Μεταβίβαση παραμέτρων κατ'αναφορά

Στο επόμενο παράδειγμα φαίνεται η διαφορά στη μεταβίβαση παραμέτρων μεταξύ μεταβλητών βασικών τύπων όπως int, float, double και αντικειμένων.

```
public static void update(int number, Person who) {
    number = number + 100;
    who.id = number;
```

```
}
……
Person alan   = new Person(“Alan”, “Turing”, 1);
int N = 10;
update( N, alan );
println(N, alan.id);
```

Αν εκτελέσετε το παραπάνω παράδειγμα θα δείτε ότι ενώ η μεταβλητή N δεν αλλάζει τιμή, άρα για αυτή δημιουργείται αντίγραφο, το πεδίο id του αντικειμένου alan έχει γίνει 110. Δηλαδή όταν μεταβιβάζουμε ένα αντικείμενο σε μια μέθοδο οι αλλαγές μέσα γίνονται στο ίδιο και όχι σε αντίγραφό του.

Αν αντί για who.id = number δίναμε την εντολή who = new Person(“leonard”, “euler”, 10) το αντικείμενο alan θα άλλαζε;

4.3. Προσδιοριστές Πρόσβασης

Μέχρι τώρα είδαμε για το πώς ορίζουμε μια κλάση και μία μέθοδο στην κλάση αυτή κι ότι μπροστά από κάθε δήλωση μεθόδου και κλάσης υπάρχει συνήθως η δεσμευμένη λέξη *public* που ονομάζεται προσδιοριστής πρόσβασης. Οι προσδιοριστές πρόσβασης είναι δεσμευμένες λέξεις τις οποίες προσθέτουμε σε κλάσεις, μεθόδους και μεταβλητές για να αλλαχθεί η συμπεριφορά τους. Η java έχει τέσσερεις προσδιοριστές πρόσβασης οι οποίοι είναι:

private-Επιτρέπεται η πρόσβαση στο πεδίο/μέθοδο μόνο μέσα στις μεθόδους αυτής της κλάσης αλλά όχι στις υποκλάσεις της.

protected- Επιτρέπεται η πρόσβαση στο πεδίο/μέθοδο μόνο στις μεθόδους της κλάσης αυτής καθώς και στις υποκλάσεις της.

public- Επιτρέπεται η πρόσβαση στο πεδίο/μέθοδο σε όλους, ακόμα και από αντικείμενο άλλης κλάσης.

τίποτα – Το πεδίο/μέθοδος είναι public αλλά μόνο για το πακέτο που ανήκει.

Ας τροποποιήσουμε την κλάση Person που ορίσαμε παραπάνω και αντί για τον προσδιοριστή πρόσβασης public ας βάλουμε τον προσδιοριστή private στη θέση του.

```
public class Person {
    private String firstname;
    private String lastname;
    private int id;

}
```

και ας επαναλάβουμε τις ίδιες εντολές με προηγουμένως:

```
alan   = new Person();
alan.firstname = “Alan”;
alan.lastname = “Turing”;
alan.id = 1;
```

Το αποτέλεσμα που θα πάρουμε αν μεταγλωττίσουμε τις παραπάνω εντολές θα είναι το παρακάτω μήνυμα λάθους κατά τη μεταγλώττιση:

```
The field Person.id is not visible
```

Το μήνυμα αυτό μας λέει ότι δεν μπορούμε να έχουμε πρόσβαση στο πεδίο id της κλάσης Person. Αυτό συνέβη επειδή ορίσαμε τα πεδία της κλάσης private, κάτι που προφανώς σημαίνει ότι δεν μπορούμε να έχουμε πρόσβαση σε αυτά σε κώδικα εκτός της κλάσης. Ο προγραμματιστής που θα χρησιμοποιήσει την κλάση στον κώδικά του δεν μπορεί να έχει πρόσβαση σε αυτά τα πεδία

απευθείας. Έτσι επιτυγχάνουμε ένα από τα βασικά χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού την ενθυλάκωση.

Ενθυλάκωση (*Encapsulation*) καλείται η απόκρυψη των δεδομένων μιας κλάσης από το υπόλοιπο πρόγραμμα έτσι ώστε η πρόσβαση σε αυτά να επιτρέπεται μόνο μέσω κατάλληλα σχεδιασμένων δημόσιων μεθόδων. Ο σκοπός είναι να περιοριστούν οι ενέργειες του χρήστη/προγραμματιστή της κλάσης στις απολύτως απαραίτητες έτσι ώστε να μην έχει πρόσβαση στα δεδομένα της κλάσης. Τα δεδομένα είναι σαν να βρίσκονται μέσα σε μια προστατευτική κάψουλα και από εκεί προέκυψε ο όρος *encapsulation*. Ένας πιο περιγραφικός όρος στα ελληνικά θα μπορούσε να είναι και ο *εγκλεισμός*. Έτσι επιτυγχάνεται η αφαίρεση των δεδομένων και ο διαχωρισμός της διασύνδεσης (interface) δηλαδή της όψης αυτού που βλέπει ο προγραμματιστής και της υλοποίησης.

Για να επιτρέψουμε την προσπέλαση στα πεδία αυτά αλλά με ελεγχόμενο τρόπο ορίζουμε δημόσιες μεθόδους πρόσβασης όπως παρακάτω:

```
public class Person {
    private String firstname;
    private String lastname;
    private int id;

    public int getId() { return id; }
    public String getFirstname() { return firstname; }
    public String getLastname() { return lastname;  }

    public int setId(int id) {
        if (id>0 && id<10000) {
            this.id = id;
            return id;
        }
        return -1;
    }
    public void setFirstname(String fname) { firstname = fname; }
    public void setLastname(String lname) { lastname = lname;  }

    public void print() {
      System.out.println( firstname + ", " + lastname + ", " + id );
    }

}
```

Δυο σημαντικές κατηγορίες δημόσιων μεθόδων που έχουν στόχο την προσπέλαση σε ιδιωτικά πεδία είναι οι set που δίνουν τιμή στο πεδίο και η get που επιστρέφει την τιμή του. Επίσης έχουμε προσθέσει και μια μέθοδο print που να εκτυπώνει απευθείας ένα αντικείμενο Person. Τώρα ας εκτελέσουμε το παρακάτω πρόγραμμα:

```
        Person alan = new Person();
        alan.print();
        alan.setId(28);
        alan.setFirstname("Alan");
        alan.setLastname("Turing");
        alan.print();
```

Η εκτέλεση των παραπάνω εντολών θα μας δώσει τα παρακάτω αποτελέσματα στην οθόνη:

null, null, 0
Alan, Turing, 28

Την πρώτη φορά εκτυπώνονται τα δεδομένα του αντικειμένου που έχει κατασκευάσει ο προκαθορισμένος κατασκευαστής που έχει οριστεί αυτόματα από την κλάση. Η τιμή null σημαίνει απουσία τιμής και δίνεται σε όλα τα αντικείμενα κλάσεων που δεν έχουν πάρει τιμή.

4.4. Κληρονομικότητα

Κληρονομικότητα ονομάζεται η επέκταση των κλάσεων σε νέες κλάσεις, τις οποίες ονομάζουμε υποκλάσεις, π.χ. Ζώο → Χελώνα. Τα αντικείμενα των υποκλάσεων κληρονομούν τα δεδομένα και τη συμπεριφορά (μέθοδοι) της υπερκλάσης. Μπορούν να τροποποιήσουν τις μεθόδους που κληρονομούν (αν δεν έχουν οριστεί ιδιωτικές) ή να προσθέσουν νέες μεθόδους.

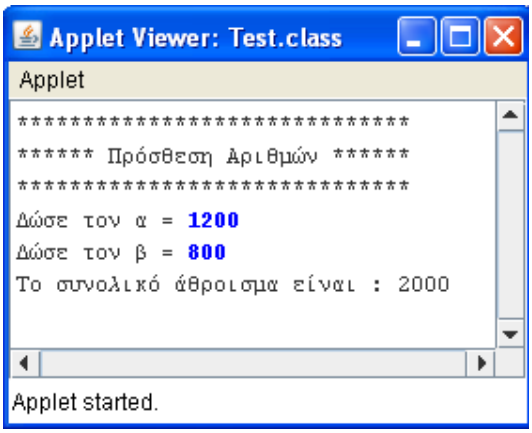
Γενικά χρησιμοποιούμε κληρονομικότητα όταν θέλουμε να επεκτείνουμε ένα υπάρχον σύστημα χωρίς όμως να πειράξουμε τον κώδικα του. Πολλές φορές όμως ανάλογα με την κλάση που επεκτείνουμε το ίδιο ακριβώς πρόγραμμα παρουσιάζει διαφορετική συμπεριφορά. Για παράδειγμα το παρακάτω πρόγραμμα δέχεται δυο αριθμούς και υπολογίζει και εμφανίζει το άθροισμά τους.

```
import acm.program.*;

public class Test extends ConsoleProgram {

        public void run( ) {
                println("*********************************");
                println("******* Πρόσθεση Αριθμών *******");
                println("*********************************");
        int a = readInt("Δώσε τον α = " )
        int b = readInt("Δώσε τον β = " )
        int sum = a + b
                println("Το συνολικό άθροισμα είναι : " + sum);
        }
}
```

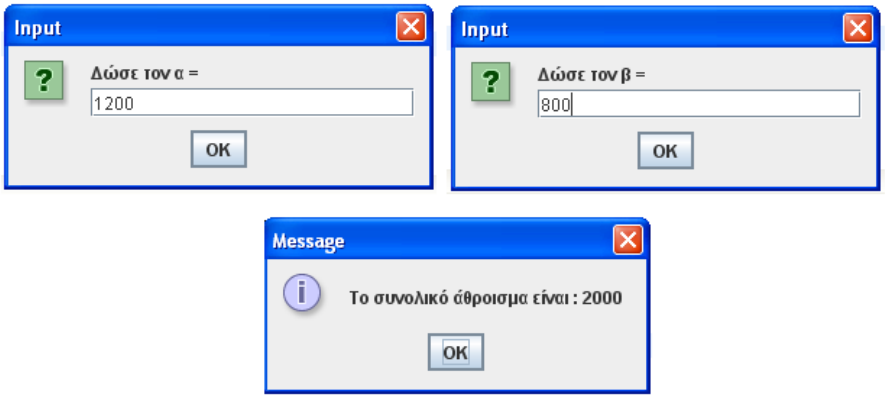
Οι συναρτήσεις readInt και println είναι οι συναρτήσεις της βιβλιοθήκης JTF που θα χρησιμοποιούμε για είσοδο και έξοδο. Υπάρχουν αντίστοιχες συναρτήσεις readFloat, readString, κλπ ανάλογα με τον τύπο των δεδομένων εισόδου.

Η συνάρτηση println συμπεριφέρεται ακριβώς όπως η μέθοδος System.out.println.

Το παραπάνω πρόγραμμα που επεκτείνει την κλάση ConsoleProgram παράγει την παρακάτω έξοδο όταν εκτελεστεί:

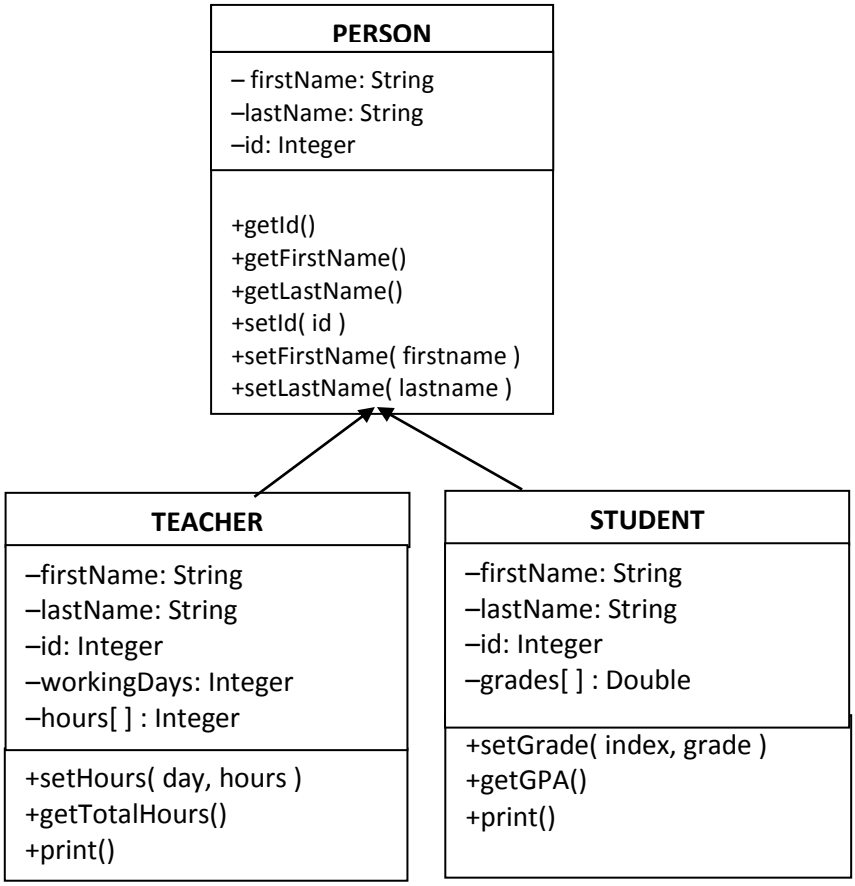


Αν όμως αντί για ConsoleProgram, επεκτείνουμε την κλάση DialogProgram τότε η επικοινωνία με τον χρήστη θα γίνει με κατάλληλα παράθυρα διαλόγων όπως παρακάτω:



Η δυνατότητα αυτή δίνεται επειδή οι κλάσεις ConsoleProgram και DialogProgram προέρχονται από την ίδια γονική κλάση Program οπότε έχουν μια κοινή διασύνδεση.

Στη συνέχεια θα ορίσουμε μερικές νέες κλάσεις οι οποίες κληρονομούν από την Person και αναφέρονται στους μαθητές (Student), καθηγητές (Teacher) ενός σχολείου. Το διάγραμμα που δείχνει την ιεραρχία των κλάσεων δίνεται παρακάτω.



Ένα τέτοιο διάγραμμα είναι γνωστό σαν διάγραμμα κλάσεων. Οι μέθοδοι που κληρονομούνται στις υποκλάσεις δεν ξαναγράφονται στις υποκλάσεις αλλά τα πεδία που κληρονομούνται φαίνονται.

Παρατηρήστε ότι οι κλάσεις που κληρονομούν από την Person έχουν τα ίδια πεδία με αυτήν (firstname, lastname, id) και προσθέτουν επιπλέον αν αυτό είναι απαραίτητο.

Ένα άλλο ενδιαφέρον σημείο είναι ότι κάθε σχήμα ορίζει τη δική του έκδοση της print. Αυτό είναι σημαντικό για το επόμενο κεφάλαιο όπου θα μιλήσουμε για πολυμορφισμό.

Στη συνέχεια δίνουμε τον κώδικα της κλάσης Person:

```
public class Person {
    private String firstname;
    private String lastname;
    private int id;

    public int getId() { return id; }
    public String getFirstname() { return firstname; }
    public String getLastname() { return lastname;  }

    public Person(String firstname, String lastname, int id) {
            setFirstname(firstname);
            setLastname(lastname);
            setId(id);
    }
    public int setId(int id) {
        if (id>0 && id<10000) {
            this.id = id;
            return id;
        }
        return -1;
    }
    public void setFirstname(String fname) { firstname = fname; }
    public void setLastname(String lname) { lastname = lname;  }

    public void print() {
            System.out.print( firstname + ", " + lastname + ", " + id + ", " );
    }
}
```

της κλάσης Teacher

```
        public class Teacher extends Person {
        private int hours[];
        private int workingDays;

        public Teacher(String firstname, String lastname, int id, int wdays,
                    int hours[]) {
                super(firstname, lastname, id);
                workingDays = wdays;
                this.hours = new int[wdays];
                for (int i=0; i<workingDays; i++) {
                        this.hours[i] = hours[i];
                }
        }
        public boolean setHours(int day, int hours) {
                if (day>=0 && day<workingDays && hours>=0 && hours<=7){
```

```
                this.hours[day] = hours;
                return true;
            }
            return false;
        }
        public double getTotalHours() {
            double sum = 0;
            for (int i=0; i<workingDays; i++) {
                sum = sum + hours[i];
            }
            return sum ;
        }
          public void print() {
            System.out.print("Καθηγητής: ");
            super.print();
            System.out.println(" Εβδομαδιαίες ώρες = " + getTotalHours());
          }
        }
```

και της κλάσης Student

```
        public class Student extends Person {
        private double grades[];

        public Student(String firstname, String lastname, int id, double
                grades[]) {
            super(firstname, lastname, id);
            this.grades = new double[grades.length];
            for (int i=0; i<grades.length; i++) {
                this.grades[i] = grades[i];
            }
        }
        public boolean setGrade(int index, double grade) {
            if (index>=0 && index<grades.length && grade>=0 && grade<=20){
                grades[index] = grade;
                return true;
            }
            return false;
        }
        public double getGPA() {
            double sum = 0;
            for (int i=0; i<grades.length; i++) {
                sum = sum + grades[i];
            }
            return sum / grades.length;
        }
        public void print() {
            System.out.print("Μαθητής: ");
            super.print();
            System.out.println(" Μέσος Όρος = " + getGPA());
          }
        }
```

Στα παραπάνω παραδείγματα είναι φανερό ότι όταν μια κλάση (ή τάξη) κληρονομεί από μια άλλη, κληρονομεί όλα τα πεδία και τις μεθόδους από την κλάση-γονέα, προσθέτει καινούργια και τροποποιεί ήδη υπάρχοντα.

Η υπερκλάση Object της Java

Η κλάση Object είναι η υπερκλάση όλων των κλάσεων της Java. Έτσι κάθε κλάση που ορίζουμε κληρονομεί κάποια λειτουργικότητα από τη γονική κλάση Object. Η λειτουργικότητα αυτή εκφράζεται μέσα από ένα σύνολο μεθόδων, τις οποίες συνήθως εξειδικεύουμε για τη δική μας κλάση. Οι βασικότερες από αυτές είναι οι εξής :

clone(): δημιουργεί και επιστρέφει ένα αντίγραφο του αντικειμένου.

equals(Object): εξετάζει την ισότητα 2 αντικειμένων. Επιστρέφει true αν τα αντικείμενα είναι ίδια, αλλιώς false.

toString(): Επιστρέφει αυτό που θέλουμε να εμφανιστεί αν εκτυπώσουμε το αντικείμενο στην οθόνη, μόνο που εμείς ορίζουμε ποια δεδομένα και με ποιον τρόπο θα εμφανιστούν.

getClass(): Επιστρέφει το αντικείμενο που αντιστοιχεί στην συγκεκριμένη κλάση σύμφωνα με το περιβάλλον εκτέλεσης της Java.

Οι παραπάνω μέθοδοι θα πρέπει να υλοποιηθούν ξανά για τη νέα κλάση αλλιώς να χρησιμοποιηθούν ενδέχεται να μην έχουν την αναμενόμενη λειτουργία.

4.5. Υπερφόρτωση

Υπερφόρτωση μεθόδου (method overloading) είναι η κατάσταση κατά την οποία υπάρχουν, μέθοδοι με το ίδιο όνομα και διαφορετικά ορίσματα (παραμέτρους). Αν πρόκειται για μεθόδους της ίδιας κλάσης διαφοροποιούνται μόνο από τις διαφορές τους στα ορίσματα. Για παράδειγμα η κλάση Math του πακέτου java.lang περιλαμβάνει τις εξής μεθόδους:

static double abs(**double** a)

static float abs(**float** a)

static int abs(**int** a)

static long abs(**long** a)

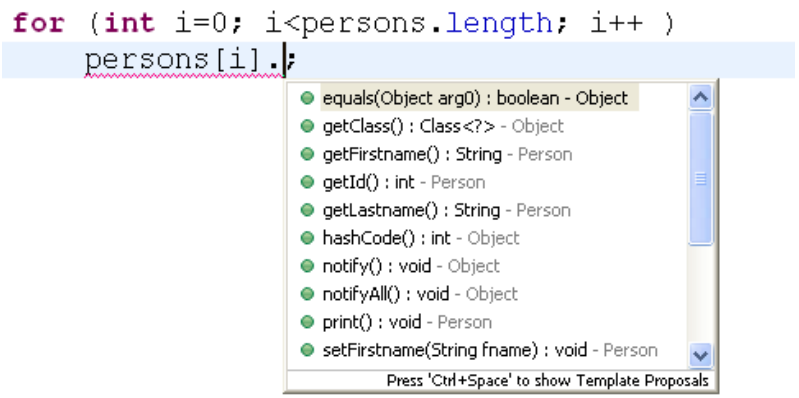
για τον υπολογισμό της απόλυτης τιμής ενός αριθμού. Επειδή όμως έχουμε διαφορετικούς τύπους αριθμών ορίσαμε τέσσερις διαφορετικές μεθόδους μια για κάθε τύπο. Κατά την κλήση της μεθόδου αποφασίζεται ποια μέθοδος θα κληθεί ανάλογα με τον τύπο του ορίσματος που έχει δοθεί.

Υπερφόρτωση έχουμε συνήθως και με τους κατασκευαστές μιας κλάσης, αφού μπορούμε να ορίσουμε πολλά είδη κατασκευαστών όπως ο προκαθορισμένος (χωρίς ορίσματα), ένας κατασκευαστής με ένα όρισμα, ένας με δυο κλπ ή ο κατασκευαστής αντιγραφής ο οποίος εκτελεί μια αντίστοιχη αλλά όχι ίδια λειτουργία με την μέθοδο clone() της κλάσης Object.

4.6. Πολυμορφισμός

Στο προηγούμενο παράδειγμα με τους μαθητές και τους καθηγητές η μέθοδος print της Person ορίζεται ξανά στις υποκλάσεις της, Teacher και Student διαφοροποιημένη για κάθε περίπτωση. Αυτό το ονομάζουμε *υπέρβαση* ή *υποσκέλιση μεθόδου* (method overriding). Σε αυτή την περίπτωση η υποκλάση ορίζει μια μέθοδο ίδια με μια που υπάρχει και στη γονική της κλάση.

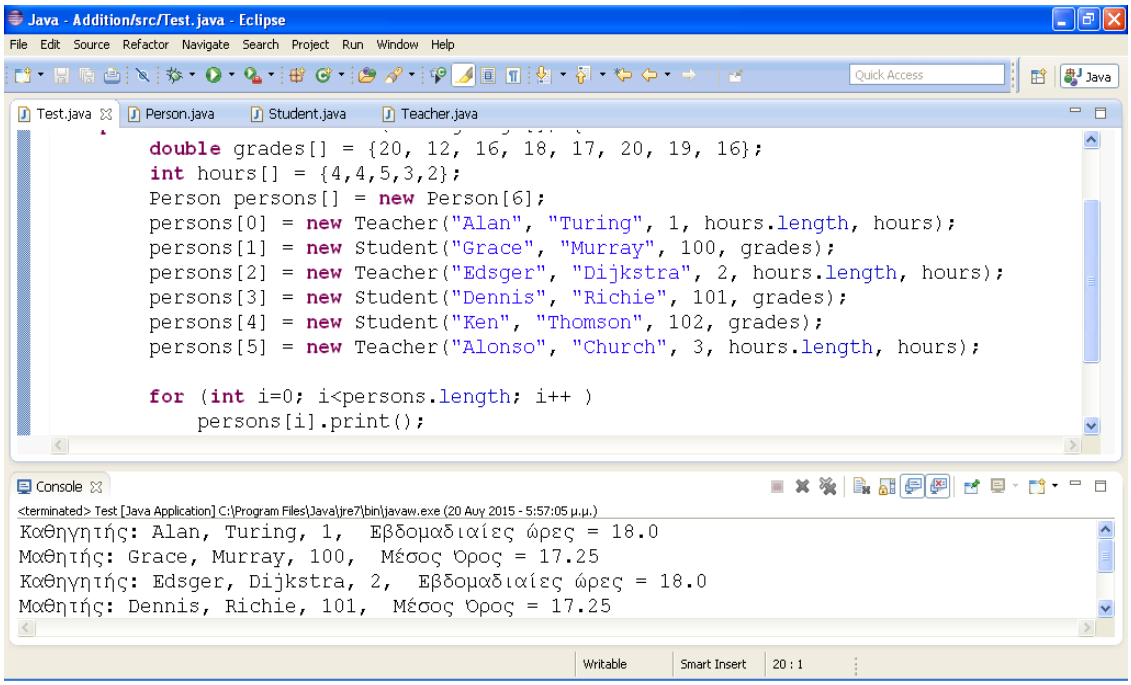
Ας δούμε το παρακάτω παράδειγμα όπου ένας πίνακας περιέχει αναφορές σε αντικείμενα τύπου Person τα μισά από τα οποία είναι καθηγητές και τα άλλα μισά μαθητές. Καλούμε τη μέθοδο print για κάθε αντικείμενο. Παρακάτω φαίνεται η δυνατότητα που προσφέρει το περιβάλλον προγραμματισμού Eclipse να μπορούμε να βλέπουμε όλες τις μεθόδους του αντικειμένου που χρησιμοποιούμε, καθώς πληκτρολογούμε. Η κόκκινη γραμμή κάτω από το persons[i] υπάρχει γιατί το Eclipse εκτελεί και έλεγχο λαθών καθώς πληκτρολογούμε.



Όταν καλείται η μέθοδος print της Person από το αποτέλεσμα που φαίνεται στην επόμενη οθόνη βλέπουμε ότι έχουν κληθεί οι αντίστοιχες print των Teacher και Student κατά περίπτωση.

Χάρη στη δυνατότητα του πολυμορφισμού ο μεταγλωττιστής «ξέρει» πότε να καλέσει ποια μέθοδο, βασισμένος στον τύπο του τρέχοντος αντικειμένου. Δηλαδή πολυμορφισμός είναι η δυνατότητα της γλώσσας προγραμματισμού να αποφασίζει δυναμικά ποια είναι η κατάλληλη μέθοδος για να κληθεί διατρέχοντας την ιεραρχία των κλάσεων.

Αυτό το χαρακτηριστικό του αντικειμενοστρεφούς προγραμματισμού προσδίδει μια υψηλού βαθμού δυναμική συμπεριφορά στα προγράμματά μας. Παρακάτω φαίνεται το κύριο πρόγραμμα main στο περιβάλλον Eclipse στο οποίο δημιουργήσαμε μια σειρά από αντικείμενα Teacher και Student. Στο κάτω μέρος της οθόνης φαίνεται το αποτέλεσμα της εκτέλεσης του προγράμματος.



4.7. Διασυνδέσεις (Interfaces)

Οι διεπαφές ή διασυνδέσεις (interfaces) στη Java μοιάζουν στον τρόπο ορισμού με τις κλάσεις αλλά περιλαμβάνουν μόνο σταθερές, ορισμούς μεθόδων (δεν περιλαμβάνουν υλοποίηση μεθόδων) και άλλους τύπους. Από τις διασυνδέσεις δεν μπορούμε να δημιουργήσουμε στιγμιότυπα. Μπορούμε μόνο να τα υλοποιήσουμε μέσα σε μια κλάση (χρήση “implements”) ή να τα προεκτείνουμε (χρήση extends βλέπε κληρονομικότητα). Ουσιαστικά ένα interface αποτελεί ένα

αφηρημένο σκελετό – σχέδιο μιας κλάσης και αποτελεί ένα είδος συμβολαίου -υπόσχεσης ότι η κλάση που το υλοποιεί παρέχει την υλοποίηση του σχεδίου του δηλαδή των μεθόδων. Ένα interface αποτελεί τύπο αναφοράς. Όταν μια κλάση το υλοποιεί μπορούμε να αναφερθούμε σε στιγμιότυπο της με τον τύπο του interface που υλοποιεί.

Η διασύνδεση περιέχει αποκλειστικά υπογραφές μεθόδων ή αφηρημένες μεθόδους (abstract), δηλαδή μεθόδους που για τις οποίες δεν έχει οριστεί υλοποίηση.

Στο προηγούμενο παράδειγμα είχαμε αντικείμενα δυο κατηγοριών: καθηγητές της κλάσης Teacher και μαθητές της κλάσης Student. Ενώ αυτά τα αντικείμενα ανήκουν στην γονική κλάση Person, δηλαδή είναι και Person, δεν έχουμε αντικείμενα που να είναι μόνο Person στον κόσμο μας. Η κλάση Person θα μπορούσε να χαρακτηριστεί σαν μια αφηρημένη κλάση των Teacher και Student που περιέχει όλα τα χαρακτηριστικά τους. Για να μπορεί να χαρακτηριστεί μια κλάση αφηρημένη, πρέπει να έχει μια τουλάχιστον αφηρημένη μέθοδο (abstract method), δηλαδή μια μέθοδο για την οποία να μην δίνεται υλοποίηση. Υπό αυτή την έννοια η κλάση Person δεν είναι αφηρημένη. Αν όμως προσθέσουμε μια αφηρημένη μέθοδο όπως η παρακάτω

```
public abstract void register();
```

τότε θα γίνει:

```
public abstract class Person {  … }
```

Η αφηρημένη κλάση ορίζει τι πρέπει να κάνουν τα αντικείμενα, δηλαδή τη διασύνδεση (interface) αλλά όχι πώς να το κάνουν, δηλαδή την υλοποίηση (implementation). Ο ορισμός της υλοποίησης των αφηρημένων μεθόδων εναπόκειται στις υποκλάσεις που κληρονομούν από την αφηρημένη κλάση.

Μια αφηρημένη κλάση που δεν έχει γνωρίσματα / πεδία, δηλαδή δεδομένα και όλες οι μέθοδοί της είναι αφηρημένες και δημόσιες δεν θεωρείται πλέον κλάση και ονομάζεται διασύνδεση (interface). Λέμε ότι οι κλάσεις που επεκτείνουν μια διασύνδεση την υλοποιούν για αυτό και χρησιμοποιούμε τη λέξη κλειδί implements.

Αν λοιπόν η Person δεν παρείχε ούτε δεδομένα αλλά ούτε και υλοποίηση για κάποια μέθοδό της θα μπορούσε να οριστεί ως διασύνδεση:

```
public interface Person {  … }
```

και η κλάση Teacher υλοποιεί την διασύνδεση Person:

```
public class Teacher implements Person {  … }
```

4.8. Πακέτα (packages)

Οι βιβλιοθήκες προγραμματισμού στην Java οργανώνονται σε πακέτα (packages). Όπως είδαμε στις προηγούμενες ενότητες για να χρησιμοποιήσουμε ένα τέτοιο πακέτο στο πρόγραμμά μας το ενσωματώνουμε με την εντολή import. Οι κλάσεις οργανώνονται σε πακέτα ανάλογα με την λειτουργία τους, για παράδειγμα στο πακέτο math μέθοδοι και αντικείμενα που έχουν σχέση με μαθηματικές εφαρμογές. Η αναφορά στο πακέτο που χρησιμοποιούμε γίνεται με βάση κάποια ιεραρχία αφού κάποιο πακέτο μπορεί να αποτελεί μέρος ενός ευρύτερου πακέτου. Τα πακέτα χωρίζονται με τελείες, π.χ. java.util .

Αν θέλουμε να οργανώσουμε πολλές κλάσεις σε ένα πακέτο έστω minedu θα πρέπει στο αρχείο κάθε κλάσης η πρώτη γραμμή να είναι η δήλωση:

```
package minedu;
```

Ένα πακέτο έχει το ίδιο όνομα με το όνομα του φακέλου στον οποίο αποθηκεύονται οι κλάσεις του. Αν τώρα θέλουμε να χρησιμοποιήσουμε στον κώδικά μας την κλάση Teacher του πακέτου minedu γράφουμε στην αρχή του αρχείου:

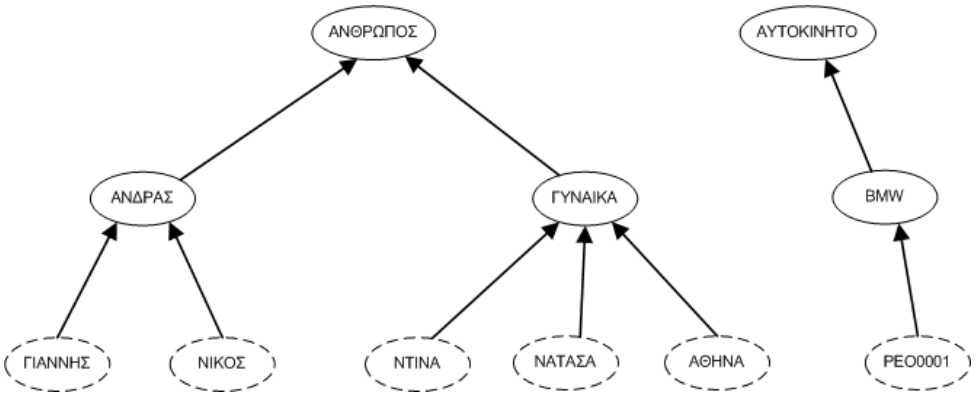
import minedu.Teacher;

4.9. Η αξία της αφαίρεσης

Για να μπορέσετε να καταλάβετε την αντικειμενοστρεφή φιλοσοφία προγραμματισμού σκεφτείτε τον πραγματικό κόσμο που αποτελείται από διάφορα αντικείμενα όπως π.χ. ο Γιάννης, η Ντίνα, το αυτοκίνητο του γείτονα που είναι μια BMW, η τηλεόραση της Νατάσας κλπ. Ο Γιάννης, η Ντίνα και η Νατάσα έχουν πιο πολλές ομοιότητες μεταξύ τους από ότι το αυτοκίνητο ή η τηλεόραση. Ανήκουν στην κατηγορία των ανθρώπων. Ομοίως όλες οι τηλεοράσεις ανήκουν στην κατηγορία των τηλεοράσεων (δεν έχουν την ίδια λειτουργικότητα?) όπως και τα αυτοκίνητα.

Για να αναπαραστήσουμε στον υπολογιστή όλα αυτά τα αντικείμενα και τις κατηγορίες τους χρησιμοποιούμε την έννοια της κλάσης. Η κλάση είναι συνώνυμο της κατηγορίας. Στα αγγλικά **class.** Κάποιοι το μεταφράζουν τάξη στα ελληνικά που ίσως είναι πιο δόκιμο από το κλάση αλλά εμείς αναφερόμαστε σε αυτές τις δομές σαν κλάσεις σε αυτό το μάθημα.

Στη συνέχεια δίνουμε ένα παράδειγμα αντικειμένων, κλάσεων και της ιεραρχίας τους.



Προσέξτε ότι στον πραγματικό κόσμο δεν υπάρχει ο άνθρωπος σαν χειροπιαστό αντικείμενο αλλά σαν έννοια. Οι κλάσεις δηλαδή περιγράφουν αφηρημένες έννοιες. Για αυτό τις κλάσεις τις περιγράφουμε στο σχήμα με συνεχόμενη γραμμή. Οι αναπαραστάσεις των εννοιών αυτών στον πραγματικό κόσμο είναι τα αντικείμενα τα οποία περιγράφονται με διακεκομμένες γραμμές. Όπως φαίνεται μια κλάση περιγράφει μια κατηγορία αντικειμένων. Λέμε ότι το αντικείμενο είναι στιγμιότυπο της κλάσης από την οποία προέρχεται.

Για παράδειγμα ο Γιάννης είναι στιγμιότυπο της κλάσης **Άνδρας**, οι Ντίνα και οι Αθηνά είναι στιγμιότυπα της κλάσης **Γυναίκα**. Τώρα προσέξτε ότι ο Γιάννης είναι και **Άνθρωπος** επίσης. Η έννοια **Άνθρωπος** είναι πιο γενική αυτής του **Άνδρα**. Δηλαδή όλοι οι άνδρες είναι άνθρωποι αλλά όλοι οι άνθρωποι δεν είναι άνδρες.

4.10. Δραστηριότητες

Δραστηριότητα 1

Να υλοποιήσετε σε ένα πακέτο τις κλάσεις Teacher και Student που περιγράφηκαν στην προηγούμενη ενότητα έτσι ώστε να κληρονομούν από την διασύνδεση (interface) Person. Να κάνετε τις απαραίτητες αλλαγές σε όλες τις κλάσεις έτσι ώστε η Person να είναι ορισμένη σαν διασύνδεση.

Δραστηριότητα 2

Να προσθέσετε μια στατική μεταβλητή κλάσης στην κλάση Teacher και μια στην κλάση Student. Οι μεταβλητές αυτές θα συγκρατούν το πλήθος των αντικειμένων κάθε κλάσης που υπάρχουν εκείνη την στιγμή. Αυτό σημαίνει ότι πρέπει να ενημερώνονται κατάλληλα όταν κάποιο νέο αντικείμενο δημιουργείται ή όταν κάποιο άλλο παύει να υπάρχει.

Δραστηριότητα 3

Να ορίσετε τις κλάσεις της ιεραρχίας της παραγράφου 4.9 και να δημιουργήσετε τα αντίστοιχα αντικείμενα.

Δραστηριότητα 4

Να τοποθετήσετε σε μια ιεραρχία κλάσεων τα εξαρτήματα ενός προσωπικού υπολογιστή

Δραστηριότητα 5

Να συντάξετε και να εκτελέσετε το πρόγραμμα της παραγράφου 4.6 όπου δημιουργείται ένας πίνακας persons με αντικείμενα της κλάσης Person.

Δραστηριότητα 6

Να ορίσετε μια δική σας κλάση MyDate για την αναπαράσταση αντικειμένων που αφορούν ημερομηνίες. Κάθε αντικείμενο θα περιέχει τρεις ακέραιους αριθμούς την ημέρα το μήνα και το έτος. Τα πεδία αυτά πρέπει να είναι ιδιωτικά, οπότε θα πρέπει να αναπτύξετε τις αντίστοιχες μεθόδους set/get για την προσπέλασή τους, υλοποιώντας και κατάλληλους ελέγχους. Για παράδειγμα ο αριθμός ημέρας πρέπει να είναι από 1 έως 7. Να αναπτύξετε και άλλες μεθόδους όπως : υπολογισμό ημερών ενός μήνα, έλεγχος αν το έτος είναι δίσεκτο, εκτύπωση της ημερομηνίας, επιστροφή ενός αντικειμένου MyDate που αναπαριστά την ημερομηνία της επόμενης ημέρας κλπ. Να υλοποιήσετε τους παρακάτω κατασκευαστές:

MyDate(int day, int month, int year)

MyDate(Date date)

Δραστηριότητα 7

Να υλοποιήσετε μια κλάση Employee για τους υπαλλήλους μιας εταιρίας ανάπτυξης λογισμικού. Τα στοιχεία κάθε υπαλλήλου τα οποία πρέπει να οριστούν ιδιωτικά είναι:

- Το ονοματεπώνυμο (String)
- Το ονοματεπώνυμο του προϊσταμένου (String)
- Ο κωδικός του τμήματος (int)
- Ο μισθός του υπαλλήλου (double)
- Η γλώσσα προγραμματισμού στην οποία εργάζεται (String)

Να υλοποιήσετε τις αντίστοιχες μεθόδους set/get και έναν κατασκευαστή της κλάσης.

Στη συνέχεια να αναπτύξετε ένα πρόγραμμα το οποίο θα δημιουργεί αντικείμενα της κλάσης Employee με δεδομένα που θα δίνει ο χρήστης και θα υπολογίζει και θα εμφανίζει

1. το συνολικό μισθό όλων των υπαλλήλων
2. Την πιο δημοφιλή γλώσσα προγραμματισμού
3. Τα τμήματα που έχουν μόνο έναν υπάλληλο
4. Για κάθε τμήμα να εμφανίζει το όνομα του προϊσταμένου του.

Κεφάλαιο 5

Προγραμματισμός οδηγούμενος από γεγονότα

5. Προγραμματισμός οδηγούμενος από γεγονότα

Εισαγωγή

Η επικοινωνία μιας εφαρμογής με τον χρήστη, σήμερα, γίνεται χρησιμοποιώντας μια γραφική διεπαφή (ή διασύνδεση) και όχι από τη γραμμή εντολών, οπότε ο σχεδιασμός της γραφικής διεπαφής σε πολλές εφαρμογές είναι το ίδιο ή και περισσότερο κρίσιμος από την εσωτερική λειτουργία της ίδιας της εφαρμογής. Για αυτό υπάρχει ολόκληρη επιστημονική περιοχή της πληροφορικής για τον σχεδιασμό των γραφικών διεπαφών και της αλληλεπίδρασής τους με τον χρήστη, γνωστή ως *Επικοινωνία Ανθρώπου – Μηχανής*.

Μια χρήσιμη βιβλιοθήκη είναι η AWT(Abstract Windowing Toolikt) η οποία είναι ένα σύνολο κλάσεων για τη σχεδίαση παραθυρικών εφαρμογών και την υποστήριξη της διαδραστικότητάς τους. Οι εφαρμογές που βασίζονται στην βιβλιοθήκη AWT προσαρμόζουν την εμφάνισή τους στην πλατφόρμα εκτέλεσης. Έτσι η ίδια εφαρμογή μπορεί να έχει διαφορετική εμφάνιση σε έναν υπολογιστή με λειτουργικό σύστημα windows και ένα έναν με λειτουργικό Linux. Αυτό συμβαίνει γιατί για κάθε μια τέτοια πλατφόρμα υπάρχει μια εικονική μηχανή (JVM) για εκτέλεση στην συγκεκριμένη πλατφόρμα.

Ωστόσο ενώ η βιβλιοθήκη AWT επιτυγχάνει πολύ καλά την ανεξαρτησία από την πλατφόρμα, μειονεκτεί στην ευελιξία και στην ανάπτυξη απαιτητικών GUI. Για αυτό το λόγο έχει πλέον αντικατασταθεί από μια άλλη βιβλιοθήκη η οποία είναι πολύ πιο ευέλικτη, πιο αποδοτική και επιτρέπει την ανάπτυξη πιο σύνθετων εφαρμογών, την βιβλιοθήκη swing. Επίσης τα αντικείμενα της βιβλιοθήκης swing είναι πιο αποδοτικά αφού καταναλώνουν λιγότερους υπολογιστικούς πόρους.

Στο μάθημα αυτό δεν θα ασχοληθούμε με τις τεχνικές λεπτομέρειες της βιβλιοθήκης αλλά θα χρησιμοποιήσουμε την αντίστοιχη βιβλιοθήκη της ACM acm.gui και acm.graphics οι οποίες κρύβουν αρκετές από τις πολύπλοκες λεπτομέρειες της AWT και βοηθούν στο να γίνει η χρήση τους εύκολη και επεξηγηματική.

Χρήσιμη Υπόδειξη: Για να προσθέσουμε την βιβλιοθήκη JTF της ACM θα πρέπει αφού κατεβάσουμε το αρχείο acm.jar από την διεύθυνση http://jtf.acm.org να το προσθέσουμε στο φάκελο του project του Eclipse που έχουμε δημιουργήσει.

Στη συνέχεια με δεξί κλικ πάνω στο project στο αριστερό παράθυρο (Package Explorer) του Eclipse επιλέγουμε *Properties → Java Build Path → Add Jars* εφόσον έχουμε προσθέσει το αρχείο στο φάκελο του Project ή αν βρίσκεται σε άλλο φάκελο *Properties → Java Build Path → Add External Jars*.

Στόχοι

Οι μαθητές να μπορούν να:

- Σχεδιάζουν απλές γραφικές διεπαφές
- Χρησιμοποιούν πλαίσια διαλόγου για επικοινωνία με τον χρήστη
- Υλοποιούν μεθόδους για τον χειρισμό γεγονότων του ποντικιού ή του πληκτρολογίου
- Αναλύουν την εφαρμογή τους σαν ένα σύνολο μεθόδων κάθε μια από τις οποίες ενεργοποιείται από ένα γεγονός

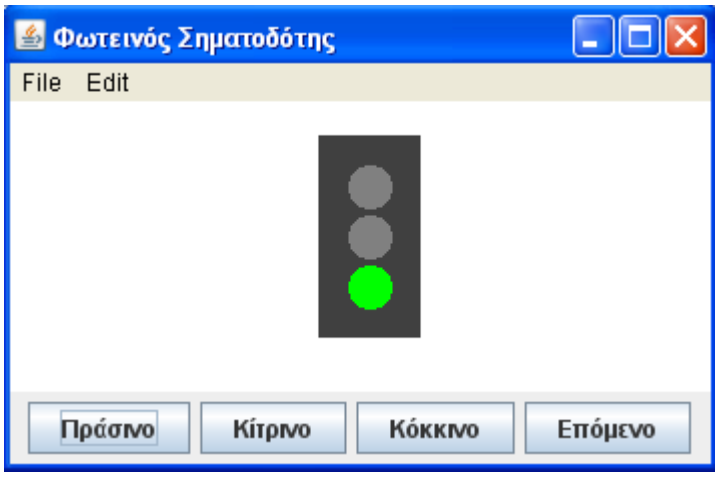
Ενότητες Κεφαλαίου

- Σχεδιασμός απλής γραφικής διεπαφής
- Δημιουργία πλαισίων διαλόγου για είσοδο και έξοδο δεδομένων
- Προγραμματισμός οδηγούμενος από γεγονότα
- Διαχειριστές γεγονότων

5.1. Σχεδιασμός απλής γραφικής διεπαφής

Δεν είναι σκοπός του μαθήματος να εμβαθύνουμε σε κάποια συγκεκριμένη βιβλιοθήκη για υλοποίηση παραθυρικών εφαρμογών αλλά να δείξουμε κυρίως τη φιλοσοφία. Το παρακάτω παράδειγμα που έχουμε τροποποιήσει ελαφρά υπάρχει διαθέσιμο στη διεύθυνση της βιβλιοθήκης της acm μαζί με πολλά άλλα παραδείγματα (demo gallery) για τα διάφορα πακέτα της βιβλιοθήκης (acm.graphics, acm.gui, acm.program, κλπ).

Το παράδειγμα που δίνουμε παρακάτω υλοποιεί έναν φωτεινό σηματοδότη. Στη μέση του παραθύρου υπάρχει ένας καμβάς στον οποίο έχουμε σχεδιάζει ένα κόκκινο φανάρι χρησιμοποιώντας τις μεθόδους GRect για τη δημιουργία παραλληλογράμμου και GOval για τη δημιουργία των τριών κύκλων, του πακέτου acm.graphics.



Η εμφάνιση του φωτεινού σηματοδότη υλοποιείται από την κλάση Stoplight στο αρχείο Stoplight.java. Δεν θα ασχοληθούμε με τις λεπτομέρειες σχεδιασμού των σχημάτων, αλλά εσείς μπορείτε να κατεβάσετε και να μελετήσετε την κλάση Stoplight και τις μεθόδους του πακέτου acm.graphics που χρησιμοποιούνται.

Παρακάτω δίνεται η κλάση StoplightGraphics η οποία μας ενδιαφέρει κυρίως λόγω του χειρισμού των γεγονότων του ποντικιού και της σχεδίασης της διεπαφής.

```
import acm.program.*;
import java.awt.event.*;
import javax.swing.*;

public class StoplightGraphics extends GraphicsProgram {

/* Παίζει το ρόλο του κατασκευαστή, αρχικοποιεί το παράθυρο */
public void init() {
        ……

/* Περιμένει για κάποιο γεγονός */
public void actionPerformed(ActionEvent e) {
        ……
}

/* Το αντικείμενο που δημιουργείται */
private Stoplight signal;
```

```
/* Ορισμός των διαστάσεων του προγράμματος */
public static final int APPLICATION_WIDTH = 350;
public static final int APPLICATION_HEIGHT = 250;

/* Δημιουργεί το αντικείμενο Stoplight, δεν μας απασχολεί στο μάθημα */
public static void main(String[] args) {
        new StoplightGraphics().start(args);
}
}
```

Η κλάση μας κληρονομεί από την κλάση GraphicsProgram που σημαίνει ότι δημιουργείται ένα παράθυρο στο οποίο μπορούμε να σχεδιάσουμε γραφικά αλλά και να προσθέσουμε διάφορα αντικείμενα για επικοινωνία με τον χρήστη όπως JLabel, JButton. Η κατασκευή αυτού του παραθύρου γίνεται από την μέθοδο init (αν σας προβληματίζει το γεγονός ότι ο κατασκευαστής δεν έχει το ίδιο όνομα με την κλάση, αναζητήστε πληροφορίες για την αρχιτεκτονική της applets στην Java).

```
public void init() {
        add(new JButton("Πράσινο"), SOUTH);
        add(new JButton("Κίτρινο"), SOUTH);
        add(new JButton("Κόκκινο"), SOUTH);
        add(new JButton("Επόμενο"), SOUTH);
        signal = new Stoplight();
        add(signal, getWidth() / 2, getHeight() / 2);
        addActionListeners();
}
```

Η μέθοδος add προσθέτει στο κάτω μέρος (SOUTH) του παραθύρου τέσσερα αντικείμενα της κλάσης JButton, δημιουργεί ένα αντικείμενο Spotlight που είναι ο φωτεινός σηματοδότης και τον προσθέτει στη μέση του παραθύρου. Η μέθοδος addActionListeners προσθέτει τους διαχειριστές γεγονότων που έχουμε ορίσει παρακάτω με την actionPerformed, η οποία ορίζει τις ενέργειες που θα λάβουν χώρα ανάλογα με το κουμπί που θα πατηθεί.

```
public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (command.equals("Επόμενο")) {
                signal.advance();
        } else if (command.equals("Πράσινο")) {
                signal.setState(Stoplight.GREEN);
        } else if (command.equals("Κίτρινο")) {
                signal.setState(Stoplight.YELLOW);
        } else if (command.equals("Κόκκινο")) {
                signal.setState(Stoplight.RED);
        }
}
```

5.2. Δημιουργία πλαισίων διαλόγου για είσοδο και έξοδο δεδομένων

Στην ενότητα της κληρονομικότητα δείξαμε πως μπορούμε να χρησιμοποιήσουμε την κλάση DialogProgram και τις μεθόδους της reading, readDouble, readString για είσοδο δεδομένων με ειδικά πλαίσια διαλόγων. Το ίδιο μπορούμε να κάνουμε και στην περίπτωση ενός αντικειμένου GraphicsProgram. Όπως προσθέτουμε με την add κουμπιά (JButton) ή κείμενο (JLabel) μπορούμε να

προσθέσουμε και ειδικά αντικείμενα (JTextField) για είσοδο κειμένου και έτσι να σχεδιάσουμε μια φόρμα εισόδου που να περιέχει αρκετά δεδομένα.

Τα αντικείμενα της κλάσης JTextField χρησιμοποιούνται για την είσοδο δεδομένων πάνω σε κάποια φόρμα. Ότι πληκτρολογεί ο χρήστης είναι τύπου String το οποίο πρέπει εμείς μετά να το μετατρέψουμε στην κατάλληλη μορφή (int, double, κλπ). Για να το αποφύγουμε αυτό η βιβλιοθήκη της acm μας προσφέρει τις κλάσεις IntField και DoubleField για την είσοδο αριθμητικών δεδομένων. Ένα απλό παράδειγμα που δείχνει τη λειτουργικότητα των αντικειμένων αυτών είναι το παράδειγμα του νομισματικού μετατροπέα (currency converter) το οποίο μπορείτε να βρείτε στη βιβλιοθήκη της jtf της acm.

```
import acm.gui.*;
import acm.program.*;
import java.awt.event.*;
import javax.swing.*;

public class CurrencyConverter extends Program {

public void init() { …… }
public void actionPerformed(ActionEvent e) {……}

private double getRateFromChooser(JComboBox chooser) {
        String currencyName = (String) chooser.getSelectedItem();
        return currencyTable.getExchangeRate(currencyName);
}

private CurrencyTable currencyTable;
private JComboBox leftChooser, rightChooser;
private DoubleField leftfield, rightField;

public static final int APPLICATION_WIDTH = 350;
public static final int APPLICATION_HEIGHT = 200;

public static void main(String[] args) {
        new CurrencyConverter().start(args);
}
}
```

Θα δείξουμε αναλυτικά μόνο τις δυο μεθόδους που μας ενδιαφέρουν την init και την actionPerformed, ώστε να φανεί με ποιο τρόπο δημιουργείται η συγκεκριμένη γραφική διεπαφή και πως γίνεται η αλληλεπίδραση με την actionPerformed.

```
public void init() {
        setLayout(new TableLayout(4, 2));
        currencyTable = new CurrencyTable();
        leftChooser = new JComboBox(currencyTable.getCurrencyNames());
        leftChooser.setSelectedItem("US Dollar");
        rightChooser = new JComboBox(currencyTable.getCurrencyNames());
        rightChooser.setSelectedItem("Euro");
        leftField = new DoubleField();
        leftField.setFormat("0.00");
        leftField.setActionCommand("Convert ->");
```

```
            leftField.addActionListener(this);
            rightField = new DoubleField();
            rightField.setFormat("0.00");
            rightField.setActionCommand("<- Convert");
            rightField.addActionListener(this);
            add(leftChooser);
            add(rightChooser);
            add(leftField);
            add(rightField);
            add(new JButton("Convert ->"));
            add(new JButton("<- Convert"));
            String date = "(rates from " + currencyTable.getDate() + ")";
            add(new JLabel(date, JLabel.CENTER), "gridwidth=2");
            addActionListeners();
    }

    public void actionPerformed(ActionEvent e) {
            String cmd = e.getActionCommand();
            if (cmd.equals("Convert ->")) {
                    double fromValue = leftField.getValue();
                    double fromRate = getRateFromChooser(leftChooser);
                    double toRate = getRateFromChooser(rightChooser);
                    double toValue = fromValue * fromRate / toRate;
                    rightField.setValue(toValue);
            } else if (cmd.equals("<- Convert")) {
                    double fromValue = rightField.getValue();
                    double fromRate = getRateFromChooser(rightChooser);
                    double toRate = getRateFromChooser(leftChooser);
                    double toValue = fromValue * fromRate / toRate;
                    leftField.setValue(toValue);
            }
    }
```

5.3. Προγραμματισμός οδηγούμενος από γεγονότα

Είδαμε ότι με την actionPerformed μπορούμε να ορίσουμε τι θα γίνει σε περίπτωση που συμβεί οποιοδήποτε γεγονός, όπως για παράδειγμα το πάτημα ενός συγκεκριμένου κουμπιού με το ποντίκι ή το πάτημα ενός πλήκτρου. Η actionPerformed υπάρχει στην GraphicsProgram και εμείς την ορίζουμε ξανά αξιοποιώντας τη δυνατότητα του πολυμορφισμού που αναφέρθηκε στην προηγούμενη ενότητα. Το χαρακτηριστικό της actionPerformed είναι ότι ενεργοποιείται όταν συμβεί οποιοδήποτε γεγονός. Φανταστείτε την σαν ένα παράλληλο πρόγραμμα που εκτελείται συνεχώς και διαχειρίζεται όλα τα γεγονότα που μας ενδιαφέρουν.

Τα προγράμματα που είχατε υλοποιήσει ως τώρα σε γραμμή εντολών είχαν το χαρακτηριστικό της σειριακής εκτέλεσης των εντολών. Η αλληλουχία των ενεργειών ήταν προκαθορισμένη. Αυτό δεν ισχύει στις παραθυρικές εφαρμογές. Εκεί υπάρχει μια λίστα από γεγονότα (πάτημα ποντικιού, πάτημα πλήκτρου, επιλογή από μενού) για κάθε ένα από τα οποία υπάρχει ένα τμήμα κώδικα έτοιμο να ανταποκριθεί στην προσταγή του χρήστη. Η ακολουθία των γεγονότων δηλαδή δεν είναι προκαθορισμένη αλλά εξαρτάται από τις προτιμήσεις του χρήστη.

5.4. Διαχειριστές γεγονότων

Η βιβλιοθήκη AWT υποστηρίζει διάφορες κλάσεις για τη μοντελοποίηση τέτοιων γεγονότων όπως *FocusEvent, MouseEvent, KeyEvent, WindowEvent*, όλες υποκλάσεις της γονικής κλάσης Event. Για κάθε ένα τέτοιο γεγονός πρέπει να ορίσουμε μια μέθοδο η οποία να χειρίζεται το συγκεκριμένο γεγονός. Αυτές οι μέθοδοι ονομάζονται διαχειριστές γεγονότων (event listeners). Οι κλάσεις αυτές σε αντιστοιχία με τα προαναφερθέντα γεγονότα είναι οι *FocusListener, MouseListener, KeyListener, WindowListener* . Όλες αυτές οι κλάσεις ορίζονται στο πακέτο java.awt.event.

5.4.1. Χειρισμός του ποντικιού

Ας δούμε το παρακάτω παράδειγμα χειρισμού του ποντικιού:

```
import acm.gui.*;
import java.awt.*;
import java.awt.event.*;
import acm.graphics.*;
import acm.program.*;
public class DragObjects extends GraphicsProgram {
   public void init() {
      GRect rect = new GRect(10, 10, 100, 100);
      rect.setFilled(true);
      rect.setColor(Color.RED);
      add(rect);
      GOval oval = new GOval(120, 120, 100, 70);
      oval.setFilled(true);
      oval.setColor(Color.GREEN);
      add(oval);
      addMouseListeners();
   }
   public void mousePressed(MouseEvent e) {
      last = new GPoint(e.getPoint());
      gobj = getElementAt(last);
   }
   public void mouseDragged(MouseEvent e) {
      if (gobj != null) {
         gobj.move(e.getX() - last.getX(), e.getY() - last.getY());
         last = new GPoint(e.getPoint());
      }
   }

   public void mouseClicked(MouseEvent e) {
      if (gobj != null) gobj.sendToFront();
   }

   private GObject gobj; /* Το αντικείμενο που σύρεται με το ποντίκι */
   private GPoint last; /* Η τελευταία θέση του ποντικιού */
}
```

όπου σε έναν καμβά (αντικείμενο τύπου GraphicsProgram) σχεδιάζουμε ένα τετράγωνο και μια έλλειψη τα οποία γεμίζουμε με κόκκινο και πράσινο χρώμα αντίστοιχα.

5.4.2. Χειρισμός του πληκτρολογίου

Ο χειρισμός των γεγονότων του πληκτρολογίου υλοποιείται λογική παρόμοια με αυτή του ποντικιού. Οι μέθοδοι της υπερκλάσης που χρειάζεται να ορίσουμε είναι οι εξής:

void keyPressed(**KeyEvent** e)
Καλείται αυτόματα μόλις ένα πλήκτρο πατηθεί

void keyReleased(**KeyEvent** e)
Καλείται αυτόματα μόλις αφήσουμε το πλήκτρο

void keyTyped(**KeyEvent** e)
Καλείται αυτόματα για κάθε πλήκτρο κατά την πληκτρολόγηση

Για να βρούμε ποιο γράμμα πατήσαμε πρέπει να επεξεργαστούμε το αντικείμενο τύπου KeyEvent. Παρακάτω δίνουμε ένα παράδειγμα χειρισμού των πλήκτρων. Κάθε φορά που ο χρήστης πατάει ένα πλήκτρο εμφανίζεται σε ένα αντικείμενο τύπου JTextArea ο χαρακτήρας που αντιστοιχεί στο πλήκτρο αυτό. Αυτή την φορά δεν θα χρησιμοποιήσουμε την βιβλιοθήκη JTF της ACM ώστε να δείξουμε πως είναι ένα πρόγραμμα με μια απλή γραφική διεπαφή σε Java χρησιμοποιώντας το πακέτο swing για τα γραφικά και το πακέτο awt για τον χειρισμό των γεγονότων.

```
import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;

public class KeyListenerExample {
JTextArea inputText, feedbackText;  // Δυο περιοχές κειμένου
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
                new KeyListenerExample();
                }
        });
}
// Ακολουθεί ο κατασκευαστής ο οποίος δημιουργεί όλη τη γραφική διεπαφή
public KeyListenerExample() {
JFrame guiFrame = new JFrame();  // Δημιουργία του παραθύρου (frame)
guiFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
guiFrame.setTitle("Παράδειγμα χειρισμού του πληκτρολογίου");
guiFrame.setSize(700,200);
guiFrame.setLocationRelativeTo(null);

feedbackText = new JTextArea();
JScrollPane scrollText = new JScrollPane(feedbackText);

inputText = new JTextArea();

inputText.addKeyListener(new KeyListener() {
        @Override
```

```
            public void keyPressed(KeyEvent e) {
              feedbackText.append("Key Pressed: " + e.getKeyChar()+ "\n");
            }
            @Override
            public void keyReleased(KeyEvent e) {
             feedbackText.append("Key Released: " + e.getKeyChar() + "\n");
            }
            @Override
            public void keyTyped(KeyEvent e) {

            feedbackText.append("Key Typed: " + e.getKeyChar() + " " +
                    KeyEvent.getKeyModifiersText(e.getModifiers()) + "\n");
                    }
            });

        guiFrame.add(inputText, BorderLayout.NORTH);
        guiFrame.add(scrollText, BorderLayout.CENTER);
        guiFrame.setVisible(true);
        }
        }
```

Αυτό στο οποίο πρέπει να επικεντρωθούμε στον παραπάνω κώδικα είναι η μέθοδος addKeyListener του inputText. Ουσιαστικά η παρακάτω γραμμή:

```
inputText.addKeyListener(new KeyListener() { … } );
```

προσθέτει τον χειριστή γεγονότων KeyListener στο αντικείμενο inputText, έτσι ώστε να ενεργοποιείται κάθε φορά που λαμβάνει χώρα κάποιο συμβάν στο αντικείμενο αυτό και αφορά το πάτημα κάποιου πλήκτρου.

Το αντικείμενο KeyListener που δημιουργείται και προστίθεται αυτόματα στο inputText είναι εφοδιασμένο με τις παρακάτω μεθόδους χειρισμού του πληκτρολογίου:

```
public void keyPressed(KeyEvent e) {
        feedbackText.append("Key Pressed: " + e.getKeyChar() + "\n");
}
```

Όταν πατηθεί ένα πλήκτρο προσθέτουμε με την append στην περιοχή κειμένου feedbackText ένα μήνυμα μαζί με τον χαρακτήρα που πατήθηκε. Παρατηρήστε ότι η μέθοδος e.getKeyChar() μας επιστρέφει τον χαρακτήρα που αντιστοιχεί στο πλήκτρο που ενεργοποίησε το γεγονός e.

```
public void keyReleased(KeyEvent e) {
        feedbackText.append("Key Released: " + e.getKeyChar() + "\n");
}
```

Ακριβώς τα ίδια με προηγουμένως αλλά ενεργοποιείται όταν αφήσουμε το πλήκτρο

5.5. Δραστηριότητες

Δραστηριότητα 1

Να υλοποιήσετε τον φωτεινό σηματοδότη της παραγράφου 4.1. Να προσθέσετε ακόμα ένα χρώμα το οποίο θα είναι μετά το κόκκινο και πριν το πράσινο.

Δραστηριότητα 2

Να γράψετε ένα πρόγραμμα το οποίο θα περιλαμβάνει μια μεγάλη περιοχή γραφικών (canvas) μέσα στον οποίο θα κινείται μια χελώνα με το πάτημα των πλήκτρων κίνησης (κάτω, πάνω δεξιά, αριστερά)

Δραστηριότητα 3

Να γράψετε ένα πρόγραμμα το οποίο θα περιλαμβάνει μια μεγάλη περιοχή γραφικών (canvas) μέσα στον οποίο θα μπορεί ο χρήστης να κάνει κλικ με το ποντίκι και θα εμφανίζεται κατάλληλο μήνυμα με τις συντεταγμένες του σημείου όπου έκανε κλικ.

Δραστηριότητα 4

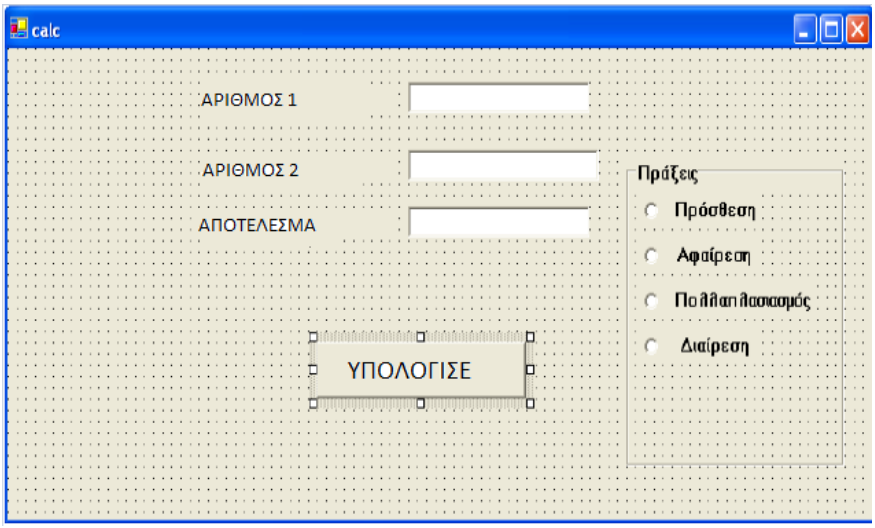
Να γράψετε ένα πρόγραμμα το οποίο θα σχεδιάζει μια σκακιέρα 8x8 και θα επιστρέφει το τετράγωνο στο οποίο ο χρήστης έκανε κλικ, π.χ. β3.

Δραστηριότητα 5

Να κατεβάσετε από τον δικτυακό τόπο κάποια από τα παραδείγματα της βιβλιοθήκης JTF και να πειραματιστείτε με αυτά. Ξεκινήστε με το παράδειγμα της τρίλιζας (Tic Tac Toe) σχεδιάζοντας το γνωστό σχήμα και στη συνέχεια προσπαθήστε να εμφανίσετε Χ ή Ο στο αντίστοιχο κελί που κάνει κλικ ο χρήστης.

Δραστηριότητα 6

Να υλοποιήσετε ένα πρόγραμμα που θα σχεδιάζει μια απλή αριθμομηχανή στην οθόνη (Εικόνα 5.5.1). Ο χρήστης θα δίνει δυο ακέραιους αριθμούς (ΑΡΙΘΜΟΣ 1 και ΑΡΙΘΜΟΣ 2), θα διαλέγει την πράξη που θα γίνει με αυτούς και πατώντας το κουμπί «ΥΠΟΛΟΓΙΣΕ» να εμφανίζεται το αποτέλεσμα στο πλαίσιο κειμένου ΑΠΟΤΕΛΕΣΜΑ.



Εικόνα 5.5.1

Κεφάλαιο 6

Βάσεις Δεδομένων

6. Βάσεις Δεδομένων

Εισαγωγή

Μια βάση δεδομένων είναι μια οργανωμένη συλλογή από δεδομένα, έτσι ώστε οι απαραίτητες λειτουργίες (εισαγωγή, διαγραφή, αναζήτηση, ενημέρωση) να γίνονται με απλό, αποδοτικό και συστηματικό τρόπο. Τα συστήματα διαχείρισης βάσεων δεδομένων (ΣΔΒΔ) παρέχουν αποδοτικούς και αποτελεσματικούς τρόπους διαχείρισης και προσπέλασης σε βάσεις δεδομένων. Σήμερα το επικρατέστερο μοντέλο βάσεων δεδομένων είναι το σχεσιακό μοντέλο (relational model) στο οποίο τα δεδομένα είναι οργανωμένα σε πίνακες και η διαχείρισή τους γίνεται με τη γλώσσα SQL (Structured Query Language).

Η Java παρέχει απλούς μηχανισμούς επικοινωνίας με ένα ΣΔΒΔ μέσα από τη βιβλιοθήκη της JDBC (Java Database Connectivity). Το δυνατό σημείο του JDBC API είναι ότι πρόκειται για ένα καθολικό API το οποίο ισχύει για όλα τα είδη βάσεων δεδομένων. Αν δηλαδή αλλάξουμε ξαφνικά Σύστημα Διαχείρισης Βάσεων Δεδομένων οι αλλαγές που θα πρέπει να κάνουμε στον κώδικα είναι μηδαμινές.

Στην ενότητα αυτή δεν θα επικεντρωθούμε στη γλώσσα SQL η οποία θεωρείται γνωστή από το μάθημα “Βάσεις Δεδομένων”.

Στόχοι

Οι μαθητές να μπορούν να:

- Υλοποιούν τη σύνδεση της εφαρμογής τους με τη βάση δεδομένων
- Σχεδιάζουν τις κλάσεις της εφαρμογής με βάση το σχεσιακό μοντέλο της βάσης δεδομένων
- Υλοποιούν μεθόδους για την αναζήτηση, ενημέρωση και εισαγωγή δεδομένων στη βάση

Ενότητες Κεφαλαίου

- Δημιουργία μιας βάσης δεδομένων
- Σύνδεση με τη βάση δεδομένων
- Θέτοντας ερωτήσεις στη βάση δεδομένων

6.1. Δημιουργία μιας βάσης δεδομένων

Αρχικά θα πρέπει να εγκαταστήσουμε ένα σύστημα διαχείρισης βάσεων δεδομένων. Προτείνεται το σύστημα MySQL το οποίο χρησιμοποιείται δωρεάν υπό την άδεια GNU General Public License και το οποίο χρησιμοποιούν δημοφιλείς εφαρμογές όπως το Google, το Facebook, το youtube και η wikipedia. Θα πρέπει να μεταβείτε στην ιστοσελίδα της MySQL και να κατεβάσετε και να εγκαταστήσετε από την Community έκδοση που είναι δωρεάν:

- τον MySQL Community Server, που αποτελεί το ίδιο το ΣΔΒΔ,
- τον MySQL WorkBench που μας παρέχει μια γραφική διεπαφή για τη διαχείριση της βάσης δεδομένων μας.
- τον οδηγό για τη σύνδεση της java με την MySQL τον MySQL Connector/J. Αν δουλεύουμε στο περιβάλλον Eclipse η προσθήκη του οδηγού γίνεται με τον ίδιο ακριβώς τρόπο με τον οποίο προσθέτουμε μια εξωτερική βιβλιοθήκη. Δηλαδή επιλέγουμε Project → Properties → Java Build Path → Libraries → Add External Jars → Επιλέγουμε το αρχείο του οδηγού JDBC.

Θα δημιουργήσουμε μια βάση δεδομένων μιας βιβλιοθήκης με δυο μόνο πίνακες **Author** για τους συγγραφείς και **Book** για τα βιβλία. Η βάση μπορεί να δημιουργηθεί είτε με γραφικό τρόπο μέσω του MySQL WorkBench είτε από τη γραμμή εντολών με χρήση τη εντολής CREATE_TABLE όπως ξέρουμε.

6.2. Σύνδεση με τη βάση δεδομένων

Αρχικά πρέπει να "φορτώσουμε" τον οδηγό επικοινωνίας ώστε να είναι διαθέσιμος. Αυτό είναι πολύ απλό και γίνεται με την παρακάτω εντολή:

```
Class.forName("com.mysql.jdbc.Driver");
```

Τώρα ο DriverManager θα αναλάβει την επικοινωνία της εφαρμογής με τη βάση δεδομένων. Το επόμενο βήμα είναι η σύνδεση με τη βάση δεδομένων. Εδώ θα πρέπει να θυμόμαστε το όνομα χρήστη *username* και τον κωδικό *password* που έχουμε δώσει στη βάση.

```
Connection con = DriverManager.getConnection( url, username, password);
```

Το url είναι και αυτό ένα String όπως τα username, password το οποίο ορίζει τη διεύθυνση της βάσης και έχει την μορφή:

```
url = "Πρωτόκολλο επικοινωνίας://Διεύθυνση Εξυπηρετητή ΣΔΒΔ/Βάση Δεδομένων"
```

Αν υποθέσουμε ότι δεν συνδεόμαστε με κάποια απομακρυσμένη βάση και κάνουμε απλά εξάσκηση στον τοπικό Server της *MySQL* στον υπολογιστή μας (*localhost*) στη βάση δεδομένων *myLibrary* που έχουμε ήδη δημιουργήσει τότε το *url* θα έχει την μορφή:

```
String url = "jdbc:mysql://localhost/myLibrary"
```

Το μόνο δύσκολο σημείο μέχρι εδώ είναι ο καθορισμός του url. Αν χρησιμοποιήσετε άλλα ΣΔΒΔ θα ακολουθήσετε τις αναλυτικές οδηγίες που υπάρχουν στους αντίστοιχους δικτυακούς τόπους. Η μόνη μέθοδος της κλάσης DriverManager που θα χρειαστεί να χρησιμοποιήσετε, είναι η μέθοδος getConnection() η οποία δημιουργεί τη σύνδεση με τη βάση. Η πρόσβαση στη βάση και η διαχείρισή της γίνεται μέσω του αντικειμένου con.

Δημιουργία πίνακα στη Βάση Δεδομένων

Για να δημιουργήσουμε έναν πίνακα στη βάση δεδομένων myLibrary θα δώσουμε μέσα από τη Java την εντολή CREATE TABLE:

```
CREATE TABLE BOOK ( TITLE VARCHAR(32), ISBN VARCHAR(10), AUTHOR_ID VARCHAR(10), COPIES INTEGER, EDITION INTEGER, PUBLISHER VARCHAR(32) );
```

Όλες οι εντολές SQL που δίνουμε στη βάση δεδομένων μεταδίδονται με τη μορφή ενός String από το αντικείμενο που διαχειρίζεται στη σύνδεση. Το String αυτό όμως θα πρέπει να το δημιουργήσουμε εμείς. Όπως φαίνεται παρακάτω δεν είναι και τόσο δύσκολο:

```
String  command = "CREATE TABLE BOOK ( TITLE VARCHAR(32), "
        + "ISBN VARCHAR(10), AUTHOR VARCHAR(32), COPIES INTEGER, "
        +  "EDITION INTEGER, PUBLISHER VARCHAR(32) )";
```

Το ερωτηματικό ';' στο τέλος της εντολής το προσθέτει ο οδηγός jdbc. Χρησιμοποιούμε τον τελεστή συνένωσης συμβολοσειρών '+' όταν η εντολή εκτείνεται σε πολλές γραμμές.

Στη συνέχεια δημιουργούμε ένα αντικείμενο τύπου Statement το οποίο μεταδίδει την εντολή SQL στη βάση δεδομένων και αφού αυτή εκτελεστεί επιστρέφει τα αποτελέσματα. Η δημιουργία του αντικειμένου γίνεται με την παρακάτω εντολή :

```
Statement statement = con.createStatement( );
```

Για να εκτελέσουμε μια εντολή SQL μέσω του jdbc υπάρχουν δυο μέθοδοι ανάλογα με το είδος της εντολής. Αν η εντολή μας τροποποιεί τη βάση καλούμε την executeUpdate( ) ενώ αν θέλουμε να κάνουμε μόνο ερωτήσεις στη βάση με την εντολή SELECT καλούμε την executeQuery( ). Επειδή η εντολή CREATE TABLE δημιουργεί έναν νέο πίνακα οπότε τροποποιεί τη βάση θα χρησιμοποιήσουμε την executeUpdate( ) :

```
statement.executeUpdate( command );
```

Αν θέλουμε να γεμίσουμε τον πίνακα με κάποια δεδομένα αυτό μπορεί να γίνει είτε μέσω της γραφικής διεπαφής της MySQL, το MySQL WorkBench είτε μέσω της Java πάλι με την executeUpdate() αφού πάλι πρόκειται για εντολή που τροποποιεί τη βάση δεδομένων:

```
String  command = "INSERT INTO BOOK VALUES(" +
          + "'Art of Computer Programming', "
          + " '1234567890', 'Donald Knuth', 28,"
          + "4, 'Addison Wesley' )";

statement.executeUpdate( command );
```

Έτσι μπορούμε να προσθέσουμε στον πίνακα BOOK εγγραφές που αναφέρονται σε συγκεκριμένα βιβλία ώστε να έχουμε κάποια δεδομένα για τις δοκιμές μας.

6.3. Θέτοντας ερωτήσεις στη βάση δεδομένων

Οι ερωτήσεις SELECT που μπορούμε να κάνουμε στη βάση δεδομένων γίνονται μέσω της executeQuery() με την ίδια λογική που γίνονται και οι εντολές ενημέρωσης της βάσης. Για παράδειγμα η παρακάτω αλληλουχία εντολών επιστρέφει όλους τους τίτλους των βιβλίων του Donald Knuth, μαζί με τον αριθμό των διαθέσιμων αντιτύπων για το καθένα:

```
Statement statement = con.createStatement( );
String  command = "SELECT TITLE, COPIES FROM BOOK WHERE " +
          "AUTHOR = 'DONALD KNUTH'";
ResultSet result = statement.executeQuery( command );
```

Σε αντίθεση με την executeUpdate( ) που δεν επιστρέφει αποτελέσματα στην εφαρμογή, η executeQuery( ) μπορεί να μην τροποποιεί τη βάση δεδομένων αλλά επιστρέφει αποτελέσματα στο πρόγραμμα. Τα αποτελέσματα αυτά αποθηκεύονται σε ένα αντικείμενο της κλάσης ResultSet.

Φανταστείτε το αντικείμενο της κλάσης ResultSet σαν έναν πίνακα όπου κάθε γραμμή είναι ένα αποτέλεσμα, και κάθε στήλη ένα από τα πεδία που ζητήσαμε με την εντολή SQL. Για παράδειγμα σίγουρα υπάρχει μια γραμμή που περιέχει ένα String με τον τίτλο "Art of Computer Programming" και τον ακέραιο αριθμό 28 για το πλήθος των αντιτύπων. Το παρακάτω τμήμα κώδικα διατρέχει το σύνολο γραμμή – γραμμή και εξάγει την πληροφορία που χρειαζόμαστε. Πρόκειται για μια τετριμμένη διαδικασία που θα κάνουμε για κάθε ερώτηση στη βάση, η οποία επιστρέφει αποτελέσματα.

```
while (result.next()) {
String title = result.getString("TITLE");
int copies = result.getInteger("COPIES");
System.out.println(title + " " + copies);
}
```

Φανταστείτε έναν νοητό δείκτη ο οποίος δείχνει κάθε φορά τη γραμμή του πίνακα την οποία μπορούμε να προσπελάσουμε. Ο δείκτης αυτός προχωράει στην επόμενη εγγραφή με την κλήση της next. Αρχικά βρίσκεται πάνω από την πρώτη γραμμή οπότε με την πρώτη κλήση τοποθετείται στην πρώτη γραμμή. Παράλληλα όσο υπάρχουν γραμμές επιστρέφει **true**. Μόλις ο νοητός αυτός δείκτης ξεπεράσει και την τελευταία εγγραφή των αποτελεσμάτων επιστρέφει **false** οπότε ξέρουμε ότι δεν υπάρχουν άλλες εγγραφές στο σύνολο των αποτελεσμάτων.

Για να εξάγουμε τα αποτελέσματα που θέλουμε χρησιμοποιούμε μια οικογένεια get μεθόδων οι οποίες παραμετροποιούνται ανάλογα με τον τύπο και το όνομα του γνωρίσματος στην εγγραφή. Για παράδειγμα για να εξάγουμε τον τίτλο του βιβλίου που είναι αλφαριθμητικό χρησιμοποιούμε την getString('TITLE') ενώ για το πλήθος των αντιτύπων που είναι ακέραιος χρησιμοποιούμε την getInteger('COPIES'). Εναλλακτικά, αντί για το όνομα του πεδίου μπορούμε να δώσουμε και τον αύξοντα αριθμό της θέσης του, π.χ. getString(1) για τίτλο και getInteger(2) για το πλήθος των αντιγράφων.

Η βιβλιοθήκη JDBC προσφέρει και άλλες δυνατότητες τις οποίες μπορείτε να αναζητήσετε στην τεκμηρίωση του ομώνυμου API.

6.4. Δραστηριότητες

Δραστηριότητα 1

Να δημιουργήσετε τη βάση δεδομένων myLibrary και να προσθέσετε δεδομένα στον μοναδικό πίνακα της βάσης myLibrary. Στη συνέχεια να εκτελέσετε διάφορες ερωτήσεις μέσω της Java.

Δραστηριότητα 2

Να τροποποιήσετε τον κώδικα που δίνεται για την επεξεργασία των αποτελεσμάτων μέσω της ResultSet ώστε να εμφανίζονται όλα τα στοιχεία των βιβλίων και στο τέλος να εμφανίζεται το πλήθος των αποτελεσμάτων.

Δραστηριότητα 3

Θέλουμε να αναπτύξουμε το σύστημα μηχανογράφησης μιας δανειστικής βιβλιοθήκης. Στο σύστημα καταχωρούνται τα στοιχεία των βιβλίων, των συγγραφέων και των αναγνωστών της

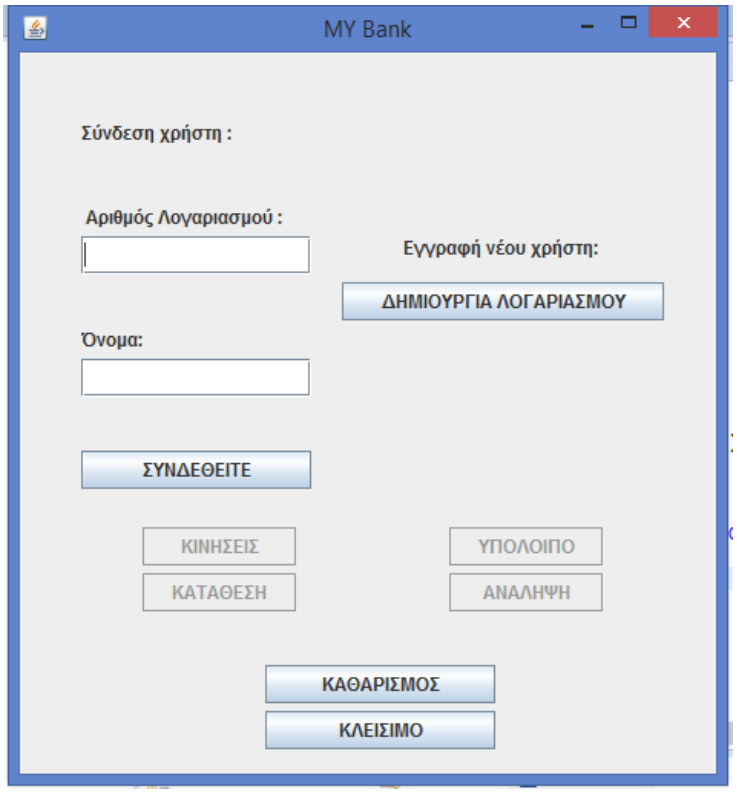
βιβλιοθήκης. Εσείς θα υλοποιήσετε την όψη του αναγνώστη μόνο η οποία περιλαμβάνει τις εξής ενέργειες:

- Δανεισμό βιβλίου
- Επιστροφή βιβλίου

Κάθε μια τέτοια ενέργεια θα έχει το δικό της πλαίσιο διαλόγου το οποίο θα δέχεται τα στοιχεία του χρήστη και του βιβλίου που θέλει να δανειστεί. Αν δεν υπάρχει άλλο αντίτυπο του βιβλίου διαθέσιμο ή ο χρήστης έχει ήδη δανειστεί 4 βιβλία θα επιστρέφει ειδοποίηση στον χρήστη ότι η συναλλαγή με τη βάση δεν είναι δυνατή και θα εξηγεί γιατί.

Δραστηριότητα 4

Θέλουμε να αναπτύξουμε μια παραλλαγή του συστήματος ATM των τραπεζών. Ο χρήστης εγγράφεται στο σύστημα δίνοντας το όνομα και τον αριθμό λογαριασμού του. Μετά τη σύνδεση του, ο χρήστης μπορεί να καταθέσει ένα ποσό στο λογαριασμό του, να κάνει ανάληψη, να δει το υπόλοιπο του λογαριασμού του και τις κινήσεις του. Η είσοδος και έξοδος δεδομένων προς και από την βάση, μπορεί να γίνει με τη χρήση των διαγνωστικών μηνυμάτων της κλάσης JOptionPane (Ενότητα 2.6). Η γραφική διεπαφή (GUI) της εφαρμογής μπορεί να υλοποιηθεί όπως φαίνεται στην παρακάτω εικόνα.



Κεφάλαιο 7

Δικτυακός Προγραμματισμός

7. Δικτυακός Προγραμματισμός

Εισαγωγή

Ο προγραμματισμός σε επίπεδο δικτύου ή ιστού είναι ένα από τα πιο δυνατά σημεία της Java. Για την Java η διαχείριση των δεδομένων είτε αυτά προέρχονται από κάποιο αρχείο είτε από μιας ιστοσελίδα στο δίκτυο γίνεται με τον ίδιο ακριβώς τρόπο, με τη χρήση ρευμάτων (streams). Αυτό διευκολύνει σε μεγάλο βαθμό την ανάπτυξη δικτυακών εφαρμογών. Το πακέτο java.net προσφέρει επικοινωνία μέσω των πρωτοκόλλων TCP/IP και UDP παρέχοντας για το σκοπό αυτό κλάσεις όπως URL, URLConnection, Socket, και ServerSocket για το πρωτόκολλο TCP, και DatagramPacket, DatagramSocket, και MulticastSocket για το πρωτόκολλο UDP.

Όλη η επικοινωνία στις δικτυακές εφαρμογές βασίζεται στο μοντέλο πελάτη – εξυπηρετητή (client – server) (ή εξυπηρέτη ή διακομιστή) σύμφωνα με το οποίο ο πελάτης απαιτεί μια υπηρεσία και ο διακομιστής του την παρέχει. Όταν για παράδειγμα εκτελούμε μια αναζήτηση στον ιστό, η μηχανή αναζήτησης είναι ο εξυπηρέτης και εμείς (για την ακρίβεια ο browser που χρησιμοποιούμε) ο πελάτης.

Στόχοι

Οι μαθητές να μπορούν να:

- Χρησιμοποιούν αντικείμενα URLs και Sockets για την ανάπτυξη δικτυακών εφαρμογών
- Διακρίνουν τη διαφορά μεταξύ πελάτη και εξυπηρετητή σε μια δικτυακή εφαρμογή
- Να σχεδιάζουν εφαρμογές βασισμένες στην αρχιτεκτονική πελάτη εξυπηρετητή

Ενότητες Κεφαλαίου

- Χρήσιμα αντικείμενα
- Το μοντέλο πελάτη – εξυπηρέτη
- Uniform Resource Locator
- Sockets
- Datagrams

7.0 Χρήσιμα αντικείμενα

Σε αυτή την παράγραφο θα αναφερθούμε σε δυο είδη αντικειμένων τα οποία καταφέραμε να αποφύγουμε μέχρι τώρα, είτε με τη χρήση της βιβλιοθήκης της acm είτε αγνοώντας τα. Τα αντικείμενα αυτά όμως είναι πολύ σημαντικά για τις εφαρμογές επικοινωνίας δεδομένων. Αναφερόμαστε στις εξαιρέσεις και στα ρεύματα εισόδου/εξόδου. Δεν είναι σκοπός μας να επικεντρωθούμε σε αυτά τα αντικείμενα αλλά μόνο να τα χρησιμοποιήσουμε σαν εργαλεία για να καταλάβουμε τη βασική αρχιτεκτονική μιας δικτυακής εφαρμογής σε Java, με απώτερο σκοπό να μπορούμε να υλοποιήσουμε και τη δική μας.

Ρεύματα Εισόδου – Εξόδου στη Java

Στην Java όλες οι εργασίες που έχουν να κάνουμε με είσοδο-έξοδο ή μετάδοση-λήψη δεδομένων υλοποιούνται μέσα από ρεύματα (streams) τα οποία είναι αντικείμενα της Java που ελέγχουν τη ροή δεδομένων μεταξύ της εφαρμογής και ενός αρχείου ή μιας συσκευής εισόδου/εξόδου. Τα αντικείμενα αυτά ορίζονται στο πακέτο *java.io* .

Τα ρεύματα χωρίζονται σε δυο κατηγορίες τα ρεύματα bytes (byte streams) για δυαδικά δεδομένα και τα ρεύματα χαρακτήρων (character streams) για χαρακτήρες UNICODE. Τα δεύτερα βολεύουν όταν έχουμε κείμενο.

Στο προκαθορισμένο πακέτο java.lang ορίζεται η πολύ σημαντική κλάση System η οποία περιέχει και τρία στατικά αντικείμενα το in για είσοδο, το out για έξοδο και το err για μηνύματα λάθους. Ήδη έχουμε χρησιμοποιήσει σε κάποιες περιπτώσεις την System.out.println( ) για έξοδο στην οθόνη. Το System.in (ρεύμα εισόδου) είναι συνδεδεμένο με το πληκτρολόγιο. Για να διαβάσουμε από το πληκτρολόγιο πρέπει να δημιουργήσουμε το παρακάτω αντικείμενο:

```
BufferedReader bReader =
        new BufferedReader( new InputStreamReader( System.in ) );
```

Με την παραπάνω εντολή συνδέουμε το πληκτρολόγιο με ένα buffer χαρακτήρων τύπου BufferedReader. Το αντικειμένο bReader χειρίζεται πλέον το ρεύμα εισόδου. Ωστόσο επειδή η είσοδος επιστρέφει bytes χρειαζόμαστε ένα φιλτράρισμα της εισόδου που να μετατρέπει τα bytes σε χαρακτήρες. Για αυτό δημιουργούμε ένα αντικείμενο InputStreamReader το οποίο παίρνει τα bytes από το System.in τα μετατρέπει σε χαρακτήρες και τα στέλνει στο BufferedReader. Θα μπορούσαμε να πούμε ότι το InputStreamReader δρα ως μεσάζων μεταξύ των δυο αντικειμένων.

Για να διαβάσουμε τώρα τους χαρακτήρες αρκεί να μελετήσουμε την τεκμηρίωση του BufferedReader, όπου θα βρούμε την μέθοδο read. Η read επιστρέφει έναν ακέραιο τον οποίο μετατρέπουμε σε χαρακτήρα όπως φαίνεται παρακάτω:

```
char letter;
letter = ( char ) bReader.read();
```

Μπορούμε να διαβάσουμε κατευθείαν και μια ολόκληρη γραμμή και να την αποθηκεύουμε σε ένα αντικείμενο τύπου String:

```
String sentence = bReader.readLine( );
```

Ομοίως για την έξοδο υπάρχει το ρεύμα PrintStream που είναι υποκλάση της OutputStream. Πολλές φορές μπορεί να χρησιμοποιήσουμε και το PrintWriter που θεωρείται πιο χρηστικό και παρέχει τις μεθόδους print(), println() για έξοδο όλων των τύπων αντικειμένων. Δίνουμε ένα παράδειγμα παρακάτω:

```
PrintWriter pw = new PrintWriter( System.out, true );
pw.println("Είμαι αλφαρηθμητικό");
```

```
int number = 28;  float real = 3.14;
pw.println( number );  pw.println( real );
```

Εξαιρέσεις

Οι εξαιρέσεις εμφανίζονται όταν κάτι μπορεί να πάει στραβά (σχεδόν πάντα δηλαδή σύμφωνα με τον νόμο του Murphy). Αποτελούν μια δικλείδα ασφαλείας σε περίπτωση που τα πράγματα δεν πάνε όπως τα περιμένουμε και συμβεί κάποιο σφάλμα το οποίο δεν επιτρέπει την συνέχιση της εκτέλεσης του προγράμματος. Για παράδειγμα το άνοιγμα ενός αρχείου που δεν υπάρχει, η σύνδεση με έναν υπολογιστή ο οποίος δεν αποκρίνεται. Μπορεί να είναι επίσης μια διαίρεση με το 0 ή η προσπέλαση πέρα από τα όρια ενός πίνακα. Ωστόσο η συντριπτική πλειοψηφία των εξαιρέσεων αναφέρεται σε προβλήματα κατά την είσοδο / έξοδο ή μετάδοση δεδομένων. Επίσης έχουμε τη δυνατότητα να ορίσουμε και εμείς τις δικές μας εξαιρέσεις κάτι που δεν θα χρειαστεί να κάνουμε εδώ.

Όταν λέμε εξαίρεση εννοούμε συνήθως ένα συμβάν το οποίο διακόπτει τη ροή εκτέλεσης του προγράμματος. Ο μηχανισμός λειτουργίας των εξαιρέσεων είναι ο εξής: Όταν συμβεί ένα τέτοιο συμβάν η Java δημιουργεί αυτόματα ένα αντικείμενο – εξαίρεση και το εξαπολύει (throws). Αν το πρόγραμμα δεν “πιάσει” (catch), δηλαδή δεν το χειριστεί, τότε θα διακοπεί η εκτέλεση και θα εμφανιστεί κάποιο μήνυμα λάθους. Βλέπουμε τον τύπο του λάθους και ένα λεπτομερές μήνυμα. Συνήθως στον ορισμό μιας μεθόδου η πρώτη γραμμή μας πληροφορεί για τα είδη των εξαιρέσεων που μπορεί να προκύψουν κατά την εκτέλεση. π.χ.

```
public int myread(  ) throws IOException, ArithmeticException {
```

Για να χειριστούμε μια εξαίρεση χρησιμοποιούμε τρεις εντολές τις **try, catch και finally**. Εντός της **try** θέτουμε τις εντολές οι οποίες μπορεί να κάνουν throw κάποια εξαίρεση. Για κάθε είδος εξαίρεσης ορίζουμε έναν χειριστή της εξαίρεσης (exception handler) με το **catch**. Τέλος το μπλοκ εντολών του **finally** εκτελείται πάντα είτε έχουμε ρίψη εξαίρεσης είτε όχι.

Παρακάτω δίνουμε ένα παράδειγμα κώδικα που χειρίζεται μια εξαίρεση αριθμητικού τύπου (διαίρεση με το μηδέν):

```
try {
    System.out.println(“Πριν”);
    int division = 1 / 0;
    System.out.println(“Μετά”);
}
catch ( ArithmeticException e ) {
    System.out.println(“Χειρισμός εξαίρεσης: Διαίρεση με μηδέν”);
    System.out.println( e.getMessage() );
}
finally {
    System.out.println(“Ο κώδικας στο finally εκτελείται πάντα”);
}
```

7.1. Το μοντέλο πελάτη – εξυπηρετητή

Το πιο διαδεδομένο μοντέλο ανάπτυξης δικτυακών εφαρμογών είναι το μοντέλο πελάτη – εξυπηρετητή (client–server). Ο εξυπηρετητής είναι μια διεργασία που εκτελείται σε έναν ισχυρό υπολογιστή και περιμένει αιτήσεις από διεργασίες – πελάτες που ζητούν κάποια εξυπηρέτηση. Για παράδειγμα όταν θέτουμε μια ερώτηση σε μια μηχανή αναζήτησης ο πελάτης είναι ο browser που χρησιμοποιούμε και ο εξυπηρετητής η μηχανή αναζήτησης.

7.2. Uniform Resource Locator

Με το ακρωνύμιο URL (Uniform Resource Locator) περιγράφεται μια συγκεκριμένη διεύθυνση στο διαδίκτυο. Το URL περιέχει μια σειρά από στοιχεία όπως πρωτόκολλο επικοινωνίας, όνομα αρχείου, θύρα επικοινωνίας κλπ. Το πακέτο java.net παρέχει μια κλάση με όνομα URL, για τον χειρισμό URL διευθύνσεων.

Η δημιουργία και χρήση αντικειμένων URL είναι αρκετά απλή όπως φαίνεται παρακάτω:

URL wikipedia = new URL("https://en.wikipedia.org/");

Στη συνέχεια μπορούμε να ορίσουμε νέα αντικείμενα URL σε σχέση με το παραπάνω ως εξής:

URL freeSoftware = new URL(wikipedia, "Free_software");

Δηλαδή το τελευταίο URL έχει διεύθυνση:
https://en.wikipedia.org/Free_software

7.2.1. Ανάλυση

Η κλάση URL είναι εφοδιασμένη με μια σειρά μεθόδων για την εξαγωγή χρήσιμων πληροφοριών από μια διεύθυνση της μορφής URL, όπως είναι οι παρακάτω:

getProtocol( ): Επιστρέφει το όνομα του πρωτοκόλλου (http, ftp).
getHost( ): Επιστρέφει το όνομα του δικτυακού τόπου, π.χ. www.wikipedia.org
getFile( ): Επιστρέφει το όνομα της σελίδας, π.χ. “nash.html”.
getPort( ): Επιστρέφει τον αριθμό της θύρας (συνήθως 80 για http).
getRef( ): Επιστρέφει το όνομα του σελιδοδείκτη της σελίδας.

7.2.2. Σύνδεση – Ανάγνωση

Για τη δημιουργία μιας σύνδεσης με μία διεύθυνση URL, χρησιμοποιείται η κλάση URLConnection, κυρίως για το πρωτόκολλο http. Υποστηρίζει αμφίδρομη μεταφορά δεδομένων. Ας δούμε το επόμενο παράδειγμα όπου χρησιμοποιούμε ρεύματα εισόδου για τα δεδομένα που λαμβάνουμε από την σύνδεση URL που ανοίξαμε και τα εμφανίζουμε στην οθόνη γραμμή – γραμμή:

```
URL wikipedia = new URL("http://www.wikipedia.org/");
URLConnection wk = wikipedia.openConnection();
BufferedReader in = new BufferedReader(
      new InputStreamReader( wk.getInputStream()));
String inputLine;
while ((inputLine = in.readLine()) != null)
System.out.println(inputLine);
in.close();
```

7.3. Sockets

Όταν αναφέρουμε τον όρο πρωτόκολλο επικοινωνίας εννοούμε τους κανόνες που διέπουν την επικοινωνία μεταξύ δυο ή περισσότερων πλευρών. Για παράδειγμα ο κώδικας Μορς διέπεται από κάποιους κανόνες. Αυτό είναι το δικό του πρωτόκολλο. Στον δικτυακό προγραμματισμό θα μιλήσουμε για δυο βασικά πρωτόκολλα το TCP και το UDP.

Το TCP (Transmission Control Protocol) είναι ένα πρωτόκολλο που βασίζεται στη σύνδεση. Δηλαδή πρώτα δημιουργείται μια σύνδεση και μέσω αυτής μεταδίδονται τα δεδομένα. Η επικοινωνία αυτή θα μπορούσε να παραλληλιστεί με μια τηλεφωνική συνομιλία. Μόλις σηκώσουμε το ακουστικό γίνεται η σύνδεση και η επικοινωνία διενεργείται σε πραγματικό χρόνο.

Το UDP (Unreliable Datagram Protocol) είναι βασισμένο στην ασύγχρονη επικοινωνία. Δηλαδή δεν υπάρχει σύνδεση ώστε να ξεκινήσει η επικοινωνία μεταξύ των δυο πλευρών. Εδώ το

ανάλογο είναι το ταχυδρομείο. Μπορούμε να στείλουμε διάφορα γράμματα με διάφορους τρόπους (εξπρές, κούριερ) έτσι ώστε να μην φτάσουν όλα στην ίδια σειρά με την οποία ξεκίνησαν.

Ενώ και στα δυο πρωτόκολλα χρησιμοποιούνται sockets (υποδοχές διαύλου επικοινωνίας) έχει επικρατήσει όταν αναφερόμαστε σε sockets να εννοούμε επικοινωνία με TCP ενώ όταν μιλάμε για datagram να εννοούμε το πρωτόκολλο UDP.

Η Java υποστηρίζει τον προγραμματισμό με sockets με τις κλάσεις Socket για την πλευρά του πελάτη (Client) και ServerSocket, για την δημιουργία των sockets από τη πλευρά του εξυπηρετητή/διακομιστή (server).

Ο ΕΞΥΠΗΡΕΤΗΣ (SERVER)

Για να αποκατασταθεί μια σύνδεση μεταξύ δυο υπολογιστών δημιουργούμε ένα αντικείμενο ServerSocket το οποίο χειρίζεται αυτή τη σύνδεση από την πλευρά του διακομιστή (server). Η δημιουργία μιας υποδοχής (socket) η οποία θα ακούει στη θύρα *port=8128* από την πλευρά του διακομιστή γίνεται με την παρακάτω εντολή:

ServerSocket server = **new ServerSocket**(port);

η οποία χρησιμοποιεί την κλάση *ServerSocket* για να αναπαραστήσει την μια άκρη της υποδοχής με το αντικείμενο *server* που αφορά τον διακομιστή. Το αντικείμενο αυτό ακούει στη θύρα *port* και περιμένει μήνυμα από κάποιον πελάτη.

Επεξήγηση: Για να καταλάβετε πως δουλεύουν οι πόρτες φανταστείτε τις σαν ταχυδρομικές θυρίδες σε ένα μεγάλο οικοδομικό συγκρότημα το οποίο είναι ο υπολογιστής. Οι κάτοικοι του συγκροτήματος είναι τα προγράμματα εξυπηρετητές κάθε ένα από τα οποία έχει μια δική του θυρίδα την οποία ελέγχει συνέχεια για μηνύματα.

Η μέθοδος που υλοποιεί αυτή τη συμπεριφορά είναι η *accept*:

Socket client = mySocket.accept();

Μόλις κάποιος πελάτης στείλει μήνυμα, η accept επιστρέφει το αντικείμενο *client* το οποίο παριστάνει την υποδοχή που επιτρέπει την επικοινωνία με τον πελάτη, ώστε να μπορεί ο διακομιστής να διαβάζει με τις μεθόδους εισόδου/εξόδου της γλώσσας τα μηνύματα που στέλνει ο πελάτης μέσω της υποδοχής. Εδώ πρέπει να σημειώσουμε μια παρανόηση που γίνεται λόγω του ονόματος του αντικειμένου *ServerSocket*. Το αντικείμενο αυτό δεν χρησιμοποιείται για την επικοινωνία με τον πελάτη, απλά ακούει στη δεδομένη θύρα. Μόλις κάποιος πελάτης αιτηθεί σύνδεση τότε δημιουργείται το αντικείμενο client τύπου *Socket* για την επικοινωνία μεταξύ του πελάτη και του εξυπηρέτη. Το αντικείμενο *server* συνεχίζει να ακούει στη θύρα για τυχόν επόμενες αιτήσεις.

Θα πρέπει να τονίσουμε τη διαφορά μεταξύ των δυο τύπων αντικειμένων:

- Ο ***ServerSocket*** ακούει ("διαβάζει") αιτήσεις για σύνδεση και επιστρέφει μέσω της *accept* ένα αντικείμενο *Socket* που παριστάνει τη νέα σύνδεση.
- Ο ***Socket*** ακούει ("διαβάζει") πακέτα δεδομένων και όχι συνδέσεις

Στη συνέχεια δίνουμε το βασικό τμήμα του κώδικα από την υλοποίηση του διακομιστή. Η σύνδεση τερματίζεται όταν λάβουμε τη λέξη "JAVA". Έχουμε δημιουργήσει τις μεθόδους CreateReader και CreateWriter ώστε να "κρύψουμε" από τον βασικό αλγόριθμο λεπτομέρειες των ρευμάτων εισόδου-εξόδου, ώστε να επικεντρωθούμε στην επικοινωνία και όχι στις λεπτομέρειες των ρευμάτων εισόδου – εξόδου. (Θυμίζουμε ότι port = 8128)

```
ServerSocket mySocket = new ServerSocket( port );
  System.out.println("Ο εξυπηρέτης ξεκίνησε " + mySocket);

  Socket client = mySocket.accept();
  System.out.println("Η σύνδεση έγινε "+ socket);
  BufferedReader in = CreateReader(socket);
```

```
String message = in.readLine();
while (!message.equals("JAVA")) {
    System.out.println(message);
    message = in.readLine();
}
```

Παρακάτω δίνουμε και τις δυο μεθόδους που υλοποιήσαμε, για να απλοποιήσουμε τα πράγματα:

```
BufferedReader CreateReader(Socket socket)
{
 return new BufferedReader(new InputStreamReader(socket.getInputStream()));
}
PrintWriter CreateWriter(Socket socket){
  return new PrintWriter( new BufferedWriter( new OutputStreamWriter (
                    socket.getOutputStream())),true);
}
```

Ο ΠΕΛΑΤΗΣ (CLIENT)

Με αντίστοιχο τρόπο γίνεται και η δημιουργία της υποδοχής από την πλευρά του πελάτη, μόνο που εδώ ο πελάτης θα πρέπει να δώσει και τη διεύθυνση του διακομιστή στο διαδίκτυο, για παράδειγμα:

```
Socket server = new Socket("www.wikipedia.org", 80);
```

Το αντικείμενο *server* παριστάνει τη σύνδεση στη θύρα *80* του διακομιστή που βρίσκεται στη διεύθυνση *www.db-net.aueb.gr* από την πλευρά του πελάτη.

Στη συνέχεια όπου *IPAddress* είναι η διεύθυνση του υπολογιστή στον οποίο τρέχει ο διακομιστής. Έχουμε συνεννοηθεί όλοι να χρησιμοποιούμε την ίδια θύρα (8128). Παρακάτω δίνεται και το τμήμα κώδικα του πελάτη. Όπως μπορεί να διακρίνει κάποιος είναι σχεδόν ίδιο με αυτό του διακομιστή όσον αφορά την κύρια λειτουργία με τη διαφορά, ότι ενώ ο διακομιστής διαβάζει από την υποδοχή και αντιγράφει το μήνυμα στην προκαθορισμένη έξοδο (οθόνη), ο πελάτης διαβάζει από την προκαθορισμένη είσοδο αυτά που πληκτρολογεί ο χρήστης και τα αντιγράφει στην υποδοχή για να τα λάβει ο διακομιστής.

Για την σύνδεση της προκαθορισμένης εισόδου με ένα ρεύμα τύπου BufferedReader δίνουμε την παρακάτω εντολή:

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

Και συνεχίζουμε με τα τετριμμένα πλέον της σύνδεσης όπως κάναμε και προηγουμένως:

```
Socket socket = new Socket(IPaddress, 8128);
PrintWriter out = CreateWriter(socket);

String message =  in.readLine();
while (!message.equals("JAVA"));
    out.println(message);
    message = in.readLine();
}
```

7.4. Datagrams

Στα UDP Datagrams η επικοινωνία που αναπτύσσεται είναι ασύγχρονη, δηλαδή δεν υπάρχει κάποια σταθερή σύνδεση μεταξύ των αποστολέα και του παραλήπτη. Από την στιγμή που δεν υφίσταται σύνδεση μεταξύ των δυο πλευρών κάθε πακέτο που στέλνουμε πρέπει να περιέχει την διεύθυνση IP και την πόρτα προορισμού.

Θα εργαστούμε με δυο τύπους αντικειμένων:
DatagramSocket: είναι η υποδοχή (socket) που υλοποιεί την επικοινωνία με datagrams
DatagramPacket: είναι ο τύπος των πακέτων που στέλνουμε. Για να φτιάξουμε ένα νέο πακέτο αρκεί να κατασκευάσουμε ένα νέο αντικείμενο αυτής της κλάσης και να το γεμίσουμε με τα δεδομένα προς αποστολή, μαζί με τα στοιχεία αποστολής (όπως ακριβώς κάνουμε με ένα γράμμα).

Αρχικά δημιουργούμε ένα αντικείμενο DatagramSocket μέσω του οποίου θα έχουμε αμφίδρομη επικοινωνία:

```
DatagramSocket server = new DatagramSocket( );
```

Σειρά έχει η δημιουργία ενός πακέτου που θέλουμε να στείλουμε. Για να κατασκευάσουμε το πακέτο χρειάζεται να δώσουμε τα δεδομένα σε έναν πίνακα από bytes (message) το μέγεθός τους (size), τη διεύθυνση του παραλήπτη (address) και την πόρτα στην οποία περιμένει το μήνυμα (port).

```
int size = 512;
DatagramPacket packet;
packet = new DatagramPacket( message, size, address, port );
server.send( packet );
```

Αυτό χρειάζεται να το κάνουμε για κάθε πακέτο που θα στείλουμε, δηλαδή πρώτα δημιουργούμε το πακέτο και στη συνέχεια το στέλνουμε με τη send. Στη συνέχεια με την εντολή receive αναμένουμε το πακέτο – απάντηση.

```
server.receive( packet );
```

Με χρήση της μεθόδου getData( ) της κλάσης DatagramPacket εξάγουμε από το πακέτο το μήνυμα που θέλουμε.

Όπως βλέπετε το βάρος τώρα δεν δίνεται στην σύνδεση αλλά στη δημιουργία του πακέτου.

7.5. Δραστηριότητες

Δραστηριότητα 1

Να αναφέρετε μερικά παραδείγματα εφαρμογής του μοντέλου πελάτη-διακομιστή από την καθημερινή ζωή και μερικά παραδείγματα από τον κόσμο της πληροφορικής.

Δραστηριότητα 2

Θέμα για Συζήτηση: Στην περίπτωση του ταχυδρόμου που φέρνει τα γράμματα στην πόρτα μας ποιος είναι ο πελάτης και ποιος ο διακομιστής. Μια συζήτηση πάνω σε αυτό το παράδειγμα θα είχε ενδιαφέρον.

Δραστηριότητα 3

Να υλοποιήσετε ένα λογισμικό σύγχρονης συνομιλίας chat με χρήση sockets. Τα λογισμικό θα αποτελείται από ένα μόνο παράθυρο το οποίο θα έχει δυο αντικείμενα σε στυλ JTextArea. Στο ένα θα γράφει ο ένας συνομιλητής και στο άλλο θα δέχεται τα μηνύματα από τον άλλο συνομιλητή. Επίσης θα περιέχει και ένα κουμπί JButton για την αποστολή μηνυμάτων. Να το δοκιμάσετε στην τάξη σε διαφορετικούς υπολογιστές. Θα χρειαστεί να μάθετε την IP του υπολογιστή του συμμαθητή σας με τον οποίο θα θελήσετε να συνδεθείτε.

Κεφάλαιο 8

Ανάπτυξη Ολοκληρωμένης Εφαρμογής

8. Ανάπτυξη Ολοκληρωμένης Εφαρμογής

Εισαγωγή

Σε αυτή την ενότητα θα παρουσιάσουμε συνοπτικά τα στάδια ανάπτυξης μιας ολοκληρωμένης εφαρμογής με βάση τις αρχές του αντικειμενοστρεφούς προγραμματισμού. Δεν θα επικεντρωθούμε στα αρχικά στάδια της ανάλυσης διότι αυτά αναλύονται πολύ καλά στο μάθημα "Σχεδιασμός και Ανάπτυξη Δικτυακών Εφαρμογών". Ο σκοπός της ενότητας είναι να μελετήσουμε κάποια εργαλεία όπως το Ant για την ολοκλήρωση και τη συντήρηση εφαρμογών σε Java και τα JUnit / TestNG για τον συστηματικό έλεγχο της εφαρμογής. Θα πούμε επίσης λίγα λόγια για το μοντέλο σχεδιασμού Model – View – Controller (MVC) για τον διαχωρισμό της γραφικής διεπαφής με τη λογική και τον έλεγχο της εφαρμογής. Τέλος θα δείξουμε πως μπορούμε να παράγουμε αυτόματα την τεκμηρίωση για το έργο μας μέσω του εργαλείου javadoc.

Στόχοι

Οι μαθητές να μπορούν να:

- Διαχωρίζουν τον κώδικα της γραφικής διεπαφής από τον υπόλοιπο κώδικα μέσω τεχνικών αντικειμενοστρεφούς σχεδίασης
- Υλοποιούν μια εφαρμογή αν δίνεται το διάγραμμα κλάσεων και οι περιγραφές των μεθόδων
- Παράγουν την τεκμηρίωση της εφαρμογής μέσω του javadoc
- Κάνουν δοκιμές του τελικού λογισμικού και να διορθώνουν πιθανά λάθη και παραλείψεις

Ενότητες Κεφαλαίου

- Ανάλυση Απαιτήσεων
- Αντικειμενοστρεφής σχεδίαση
- Διάγραμμα κλάσεων
- Το μοντέλο σχεδιασμού Model View Controller
- Ορισμός των διεπαφών των κλάσεων
- Δημιουργία της βάσης δεδομένων
- Υλοποίηση της γραφικής διεπαφής
- Υλοποίηση των μεθόδων διασύνδεσης με τη βάση
- Ολοκλήρωση της εφαρμογής
- Έλεγχος της εφαρμογής
- Τεκμηρίωση της εφαρμογής

8.1. Ανάλυση Απαιτήσεων

Όταν ξεκινάμε την ανάπτυξη μιας εφαρμογής το πρώτο πράγμα που κάνουμε λέγεται ανάλυση απαιτήσεων και συνίσταται στην περιγραφή του **τι** θέλουμε να φτιάξουμε. Για την ακρίβεια με την ανάλυση απαιτήσεων φιλοδοξούμε να καταγράψουμε σε κάποια γλώσσα την επιθυμητή συμπεριφορά του συστήματος. Για παράδειγμα αν θέλουμε να υλοποιήσουμε ένα σύστημα δανεισμού/κρατήσεων βιβλίων μια βιβλιοθήκης το σύστημά μας θα πρέπει να επιτρέπει:

- Δανεισμό βιβλίου
- Κράτηση βιβλίου
- Επιστροφή βιβλίου
- Εισαγωγή νέου βιβλίου
- Απόσυρση βιβλίου
- Εγγραφή νέου χρήστη
- Παρακολούθηση των βιβλίων που έχουν καθυστερήσει να επιστραφούν και αποστολή ειδοποιήσεων στους αναγνώστες που τα έχουν δανειστεί.

Όλες οι παραπάνω ενέργειες φαίνονται αρκετά σαφείς εκτός ίσως από την τελευταία η οποία αποτελεί μια δέσμη ενεργειών που εκτελείται σε τακτά χρονικά διαστήματα. Εδώ χρειάζονται περισσότερες λεπτομέρειες.

Περισσότερες λεπτομέρειες χρειάζονται και στις άλλες λειτουργίες, όπως είναι για παράδειγμα οι περιορισμοί στους οποίους υπόκειται ο δανεισμός των βιβλίων, π.χ. μέχρι πόσα βιβλία μπορεί να δανειστεί στο ίδιο χρονικό διάστημα ένας χρήστης;

Όταν σε ένα τόσο απλό σύστημα όπως το παραπάνω μας εμφανίζονται τέτοια ερωτήματα, είναι προφανές ότι σε ένα πιο σύνθετο σύστημα η ανάλυση απαιτήσεων είναι μια πολύ δύσκολη υπόθεση η οποία χρειάζεται πολύ μεγάλη προσοχή. Ενδεχόμενα λάθη ή παρανοήσεις που μπορεί να γίνουν κατά τη φάση της ανάλυσης μπορεί να δημιουργήσουν μεγάλες χρονικές καθυστερήσεις στην ανάπτυξη του έργου αν ανιχνευτούν σε προχωρημένο στάδιο. Κατά την καταγραφή των απαιτήσεων για ένα σύστημα δεν καταγράφουμε μόνο την επιθυμητή λειτουργικότητά του, αλλά και τις απαιτήσεις και τους περιορισμούς σε υπολογιστικούς πόρους. Γενικότερα πρόκειται για μια αρκετά σύνθετη διαδικασία.

8.2. Αντικειμενοστρεφής Σχεδίαση

Στην αντικειμενοστρεφή σχεδίαση τα δεδομένα παίζουν κεντρικό ρόλο και όχι η λειτουργικότητα των συστημάτων. Πρώτα αναγνωρίζουμε τις οντότητες που εμφανίζονται κατά την ανάλυση του συστήματος και στη συνέχεια αναγνωρίζουμε τη λειτουργικότητά τους. Το πλεονέκτημα αυτής της προσέγγισης είναι ο πολύ υψηλός βαθμός επεκτασιμότητας των συστημάτων και η ευελιξία και προσαρμοστικότητα που έχουν στις αλλαγές.

Κάποιες από τις βασικές κατευθύνσεις που πρέπει να ακολουθήσουμε κατά την αντικειμενοστρεφή σχεδίαση είναι:

- Αναγνώριση των αντικειμένων και των κλάσεων που θα υλοποιήσουμε
- Αναγνώριση των συνδέσεων μεταξύ των αντικειμένων
- Αναγνώριση των ιδιοτήτων των αντικειμένων
- Αναγνώριση της λειτουργικότητας / συμπεριφοράς των αντικειμένων

Αφού αναγνωρίσουμε τις βασικές οντότητες που εμπλέκονται στον σχεδιασμό του συστήματος τις οποίες θα υλοποιήσουμε ως κλάσεις αντικειμένων, προσπαθούμε να δούμε μήπως μπορούμε να απλοποιήσουμε το διάγραμμα των κλάσεων με τη χρήση κληρονομικότητας. Κάτι τέτοιο έγινε όταν μιλήσαμε για κληρονομικότητα και δώσαμε το παράδειγμα με τις κλάσεις Student και Teacher που κληρονομούν από την κλάση Person. Η ανάλυση απαιτήσεων σε αυτό το απλό το παράδειγμα διέκρινε ότι το σύστημά μας περιέχει δυο είδη αντικειμένων τους καθηγητές και τους

μαθητές. Οι δυο αυτές οντότητες έχουν αρκετά κοινά στοιχεία για τα οποία θα χρειαστεί να υλοποιηθούν οι ίδιες ενέργειες δυο φορές. Για να αποφύγουμε τον πλεονασμό ορίσαμε μια γονική κλάση Person η οποία συγκεντρώνει τα κοινά τους στοιχεία.

8.3. Διάγραμμα Κλάσεων

Στα προηγούμενα κεφάλαια που μιλήσαμε για την κληρονομικότητα δώσαμε ένα διάγραμμα κλάσεων (στην ενότητα 4.4) στο οποίο φαινόταν η γονική κλάση Person από την οποία κληρονομούσαν οι Student και Teacher. Κατά τον σχεδιασμό μιας εφαρμογής με την αντικειμενοστρεφή φιλοσοφία το διάγραμμα αυτό είναι το βασικότερο δεδομένου ότι η αντικειμενοστρεφής σχεδίαση έχει ως κέντρο τα δεδομένα και όχι τις λειτουργίες του συστήματος. Η γλώσσα UML (Unified Markup Language) είναι η πιο γνωστή γλώσσα μοντελοποίησης για τη σχεδίαση συστημάτων λογισμικού, η οποία χρησιμοποιείται κατά κόρον στην αντικειμενοστρεφή σχεδίαση.

Το διάγραμμα κλάσεων στη γλώσσα UML είναι ένα σύνολο διασυνδεδεμένων γεωμετρικών σχημάτων το οποίο με διάφορους συμβολισμούς υποστηρίζει έννοιες όπως

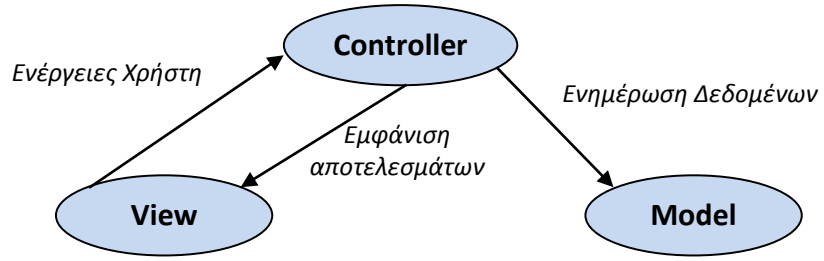
- κλάσεις
- αντικείμενα
- τα γνωρίσματά τους
- τις εξαρτήσεις μεταξύ τους

8.4. Το μοντέλο σχεδιασμού Model-View-Controller

Μια στρατηγική σχεδιασμού ενός συστήματος είναι η κατάτμησή του σε υποσυστήματα τα οποία μπορούν να σχεδιαστούν και να υλοποιηθούν ανεξάρτητα ή με ελάχιστες εξαρτήσεις μεταξύ τους. Τα υποσυστήματα αυτά πρέπει να είναι έτσι σχεδιασμένα ώστε να μπορούν να υλοποιηθούν όλα από διαφορετικούς προγραμματιστές οι οποίοι θα έχουν στα χέρια τους μόνο τις προδιαγραφές τους.

Ένα μοντέλο σχεδιασμού προς αυτή την κατεύθυνση είναι το μοντέλο Model–View–Controller που διαχωρίζει τη γραφική διεπαφή και την επικοινωνία της εφαρμογής με τον χρήστη σε σχέση με το έργο που υλοποιεί. Πρόκειται για ένα μοντέλο αρχιτεκτονικής λογισμικού που χρησιμοποιείται για τον σχεδιασμό και την ανάπτυξη περιβαλλόντων αλληλεπίδρασης με τον χρήστη. Το μοντέλο αυτό διαχωρίζει τον σχεδιασμό της εφαρμογής σε τρία μέρη:

- Το **μοντέλο** (model) που είναι η καρδιά του συστήματος και ασχολείται με τις διαδικασίες αναπαράστασης και αποθήκευσης των δεδομένων της εφαρμογής.
- Η **όψη** (view) περιγράφει αυτό που βλέπει ο χρήστης, δηλαδή είναι υπεύθυνη για την γραφική διεπαφή (user interface).
- Ο **ελεγκτής** (controller) δέχεται δεδομένα εισόδου και ενημερώνει το μοντέλο για τυχόν αλλαγές. Επίσης στέλνει στην όψη τα αποτελέσματα για εμφάνιση.



Ο βασικός στόχος της αρχιτεκτονικής MVC είναι να διαχωριστεί η παρουσίαση της πληροφορίας από τον τρόπο που είναι αποθηκευμένη εσωτερικά.

8.5. Ορισμός των διεπαφών των κλάσεων

Στο στάδιο αυτό οι κλάσεις που έχουμε ορίσει στο διάγραμμα κλάσεων αρχίζουν να γίνονται πιο συγκεκριμένες. Δεν ασχολούμαστε με την υλοποίηση αλλά πρέπει να ορίσουμε επακριβώς όλα τα πεδία (δεδομένα) και τη λειτουργικότητα (επικεφαλίδες μεθόδων) των κλάσεων. Αυτή είναι η πρώτη ακριβής περιγραφή του τι θέλουμε να κάνουμε. Θα πρέπει να διακρίνεται με τέτοια σαφήνεια ώστε αν τη δώσετε σε έναν συμμαθητή σας να μπορεί να υλοποιήσει χωρίς πολλές ερωτήσεις όλες τις μεθόδους.

Το τμήμα αυτό θα λέγαμε ότι αντιστοιχεί κατά το μεγαλύτερο μέρος στον controller του μοντέλου MVC.

8.6. Δημιουργία της Βάσης Δεδομένων

Συνήθως το Model στο μοντέλο MVC μεταφράζεται σε μια σχεσιακή βάση δεδομένων στην οποία αποθηκεύονται όλες οι πληροφορίες τις εφαρμογής. Η βάση αυτή είναι ανεξάρτητη σε κάποιο βαθμό από την υπόλοιπη εφαρμογή. Κάθε όμως εγγραφή ενός πίνακα της βάσης αντιστοιχεί στο αντικείμενο μιας κλάσης και έχει την δική του οπτικοποίηση.

8.7. Υλοποίηση των μεθόδων διασύνδεσης με τη βάση

Η επικοινωνία της βάσης δεδομένων στην Java γίνεται με το πακέτο jdbc το οποίο παρουσιάστηκε στην αντίστοιχη ενότητα. Το ιδανικό είναι σε κάθε βασικό πίνακα (οντότητα) της βάσης δεδομένων να αντιστοιχεί μια κλάση του αντικειμενοστρεφούς μοντέλου.

8.8. Υλοποίηση της γραφικής διεπαφής

Αυτό που μένει για να ολοκληρωθεί η εφαρμογή μας είναι ο σχεδιασμός της γραφικής διεπαφής και της αλληλεπίδρασης με τον χρήστη. Το τμήμα αυτό θα πρέπει να μπορεί να υλοποιηθεί ανεξάρτητα από τις άλλα τμήματα. Δηλαδή ο προγραμματιστής θα πρέπει να έχει διαθέσιμες τις διεπαφές όλων των κλάσεων και το σχήμα της βάσης δεδομένων αλλά να μην γνωρίζει τις λεπτομέρειες υλοποίησής τους.

8.9. Ολοκλήρωση της εφαρμογής

Κάποια στιγμή θα φτάσουμε σε σημείο να πρέπει να διαχειριστούμε μια αρκετά σύνθετη εφαρμογή που θα αποτελείται από δεκάδες ή εκατοντάδες αρχεία κώδικα. Η συντήρηση ενός τέτοιου συστήματος δεν είναι μια απλή υπόθεση. Αν κάποια στιγμή αλλάξει η υλοποίηση σε ένα αρχείο επειδή επινοήθηκε ένας καλύτερος αλγόριθμος ή επειδή διορθώθηκε ένα λάθος ίσως να χρειαστεί να ξέρουμε ποια αρχεία εξαρτώνται από αυτό στο οποίο έγινε η αλλαγή.

Υπάρχουν ειδικά εργαλεία αυτοματοποίησης της διαδικασίας του χτισίματος της εφαρμογής για διάφορες γλώσσες. Στην Java δυο από τα πιο δημοφιλή εργαλεία είναι το Ant το οποίο είναι ενσωματωμένο στο περιβάλλον προγραμματισμού Eclipse και το Maven. Τα εργαλεία αυτά εκτός από το χτίσιμο της εφαρμογής είναι υπεύθυνα για την διαχείριση βιβλιοθηκών και άλλων αρθρωμάτων που χρησιμοποιεί η εφαρμογή.

Και τα δυο εργαλεία περιγράφουν τις παραμέτρους και τη διαδικασία σε αρχεία XML (build.xml για το Ant και pom.xml για το Maven). Τα εργαλεία αυτά καθιστούν ταχύτατη την ανάπτυξη και εγκατάσταση μιας Java εφαρμογής αλλά και την προσθήκη νέων βιβλιοθηκών. Κατά το «χτίσιμο» της εφαρμογής, αναλαμβάνουν να συγκεντρώσουν όλες τις εξαρτήσεις, να μεταγλωττίσουν τον κώδικά μας και να «πακετάρουν» όλα τα αρχεία που χρειάζονται σε ένα τελικό αρχείο.

Το Ant (Another Neat Tool) είναι ένα δωρεάν εργαλείο ανοικτού κώδικα το οποίο μπορεί να εκτελέσει τις παρακάτω εργασίες:

- Μεταγλώττιση όλου του έργου
- Διαχείριση των αρχείων και των φακέλων του έργου

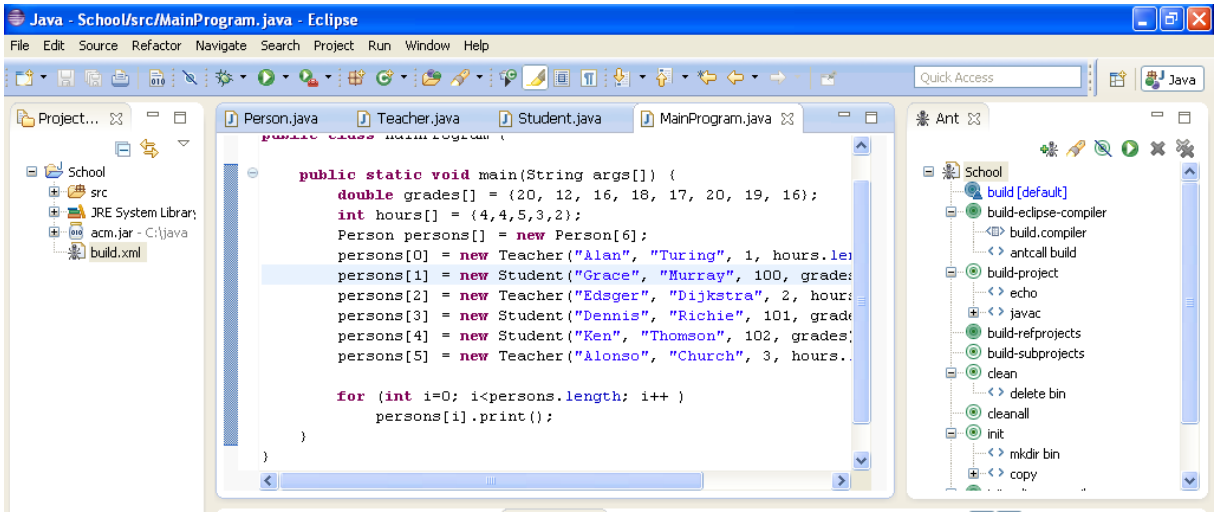
- Αυτόματη κλήση του JUnit για διενέργεια των τεστ που έχουμε ορίσει
- Πακετάρισμα κλάσεων σε αρχεία jar, war.
- Ανέβασμα και προετοιμασία της εφαρμογής για εκτέλεση σε web server

Η συνηθέστερη λειτουργία του Ant είναι ο ορισμός μιας δέσμης εργασιών που εκτελούμε συχνά με στόχο την αυτοματοποίησή της.

Το Ant είναι επεκτάσιμο ως εργαλείο ανοικτού κώδικα αφού επιτρέπει την ενσωμάτωση νέων δυνατοτήτων που οι προγραμματιστές θεωρούν αναγκαίες, μέσω κώδικα. Αν θέλουμε μέσα από το Eclipse να ξεκινήσουμε να δουλεύουμε με το Ant επιλέγουμε από το

Project → Properties → Builders → New → Ant Builder

Παρακάτω φαίνεται μια οθόνη του Eclipse για την εφαρμογή του σχολείου που έχουμε σχεδιάσει. Αριστερά φαίνεται το αρχείο build.xml που έχουμε δημιουργήσει και δεξιά το μενού ελέγχου του Ant.



Παρακάτω δίνουμε και ένα αρκετά απλό build.xml του ant για το γνωστό πρόγραμμα Hello World:

```
<?xml version="1.0"?>
<project name="Hello" default="compile">
  <target name="clean" description="remove intermediate files">
    <delete dir="classes"/>
  </target>
  <target name="clobber" depends="clean" description="remove all artifact files">
    <delete file="hello.jar"/>
  </target>
  <target name="compile" description="compile the Java source code">
    <mkdir dir="classes"/>
    <javac srcdir="." destdir="classes"/>
  </target>
  <target name="jar" depends="compile" description="create a Jar file">
    <jar destfile="hello.jar">
      <fileset dir="classes" includes="**/*.class"/>
      <manifest>
        <attribute name="Main-Class" value="HelloProgram"/>
```

```
      </manifest>
    </jar>
  </target>
</project>
```

8.10. Έλεγχος της Εφαρμογής

Κατά τη διάρκεια ανάπτυξης μιας εφαρμογής αλλά και μετά την ολοκλήρωσή της πρέπει να ελέγχουμε συνεχώς τον κώδικά μας για λάθη ώστε να είμαστε σίγουροι για την ορθότητά του. Ο έλεγχος συνήθως γίνεται δοκιμάζοντας την εφαρμογή μας με διάφορες κατηγορίες δεδομένων για να διαπιστώσουμε αν παρουσιάζει την επιθυμητή συμπεριφορά. Ο έλεγχος μιας εφαρμογής κατά μια έννοια είναι και τέχνη και θέμα εμπειρίας. Κάποιες δοκιμές όμως μπορούν να αυτοματοποιηθούν. Για την αυτοματοποίηση των δοκιμών υπάρχουν διάφορα εργαλεία με πιο γνωστά από αυτά να είναι το JUnit και το TestNG.

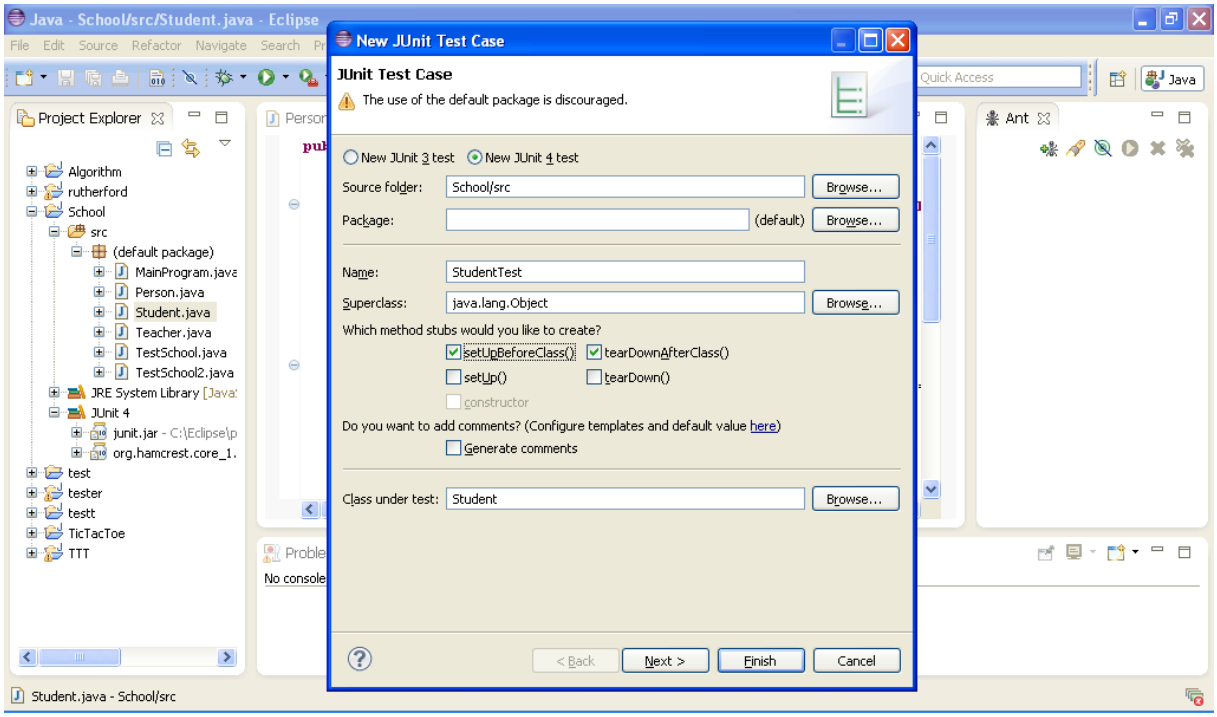
Το Junit είναι μέχρι στιγμής το πιο δημοφιλές εργαλείο το οποίο υποστηρίζεται και αυτό από το Eclipse. Για να δημιουργήσουμε ένα σύνολο δοκιμών για την εφαρμογή μας επιλέγουμε

File → New →JUnit Test Case

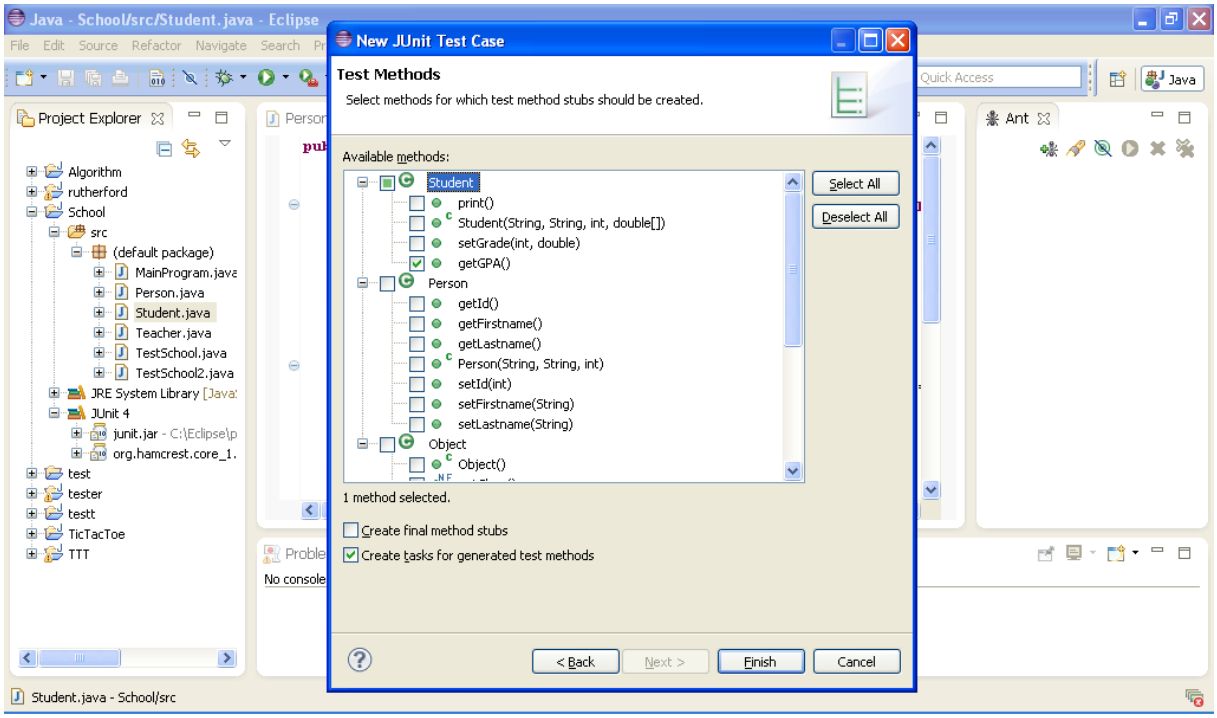
και ορίζουμε ποιες μεθόδους θα χρησιμοποιήσουμε και πάνω σε ποια κλάση της εφαρμογής μας θα εκτελεστούν τα τεστ. Μπορούμε να ορίσουμε διάφορα είδη δοκιμών για κάθε κλάση ξεχωριστά ώστε να εκτελέσουμε και τμηματικούς ελέγχους για κάθε ενότητα της εφαρμογής ξεχωριστά.

Για παράδειγμα αν θέλουμε να τρέξουμε κάποια τεστ πάνω στον υπολογισμό του μέσου όρου της κλάσης Student εκτελούμε τις εξής ενέργειες.

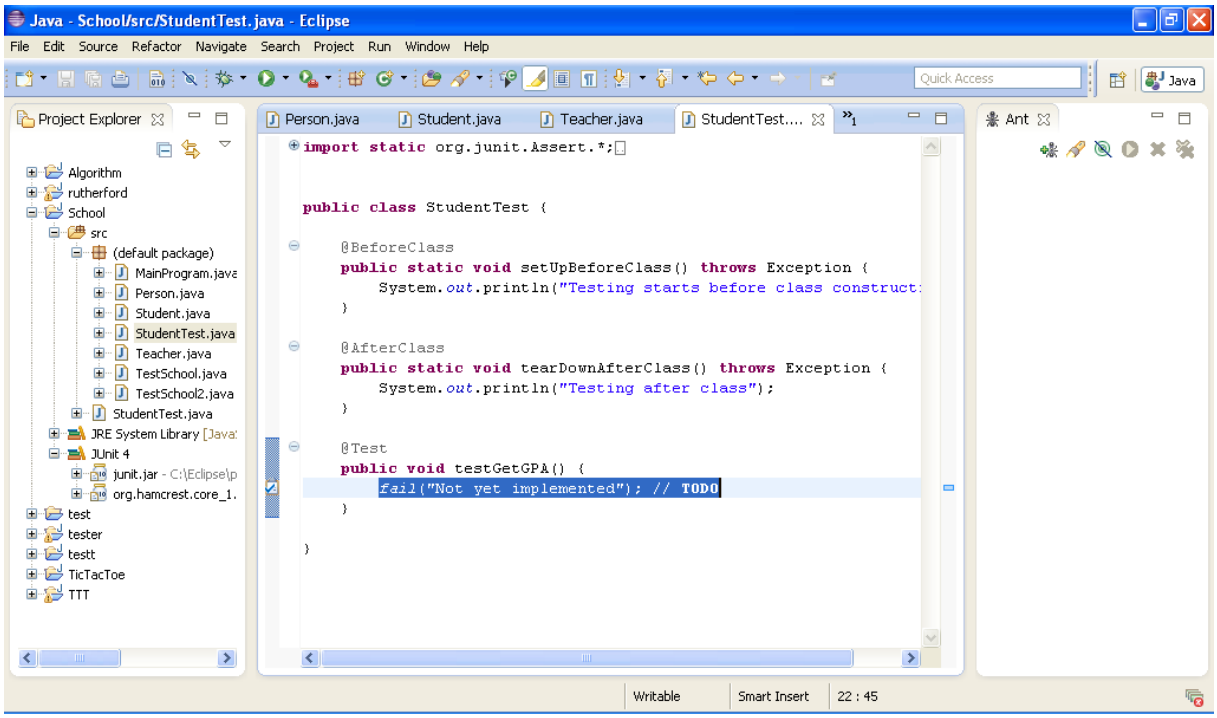
Καταρχήν δημιουργούμε ένα JUnit Test Case



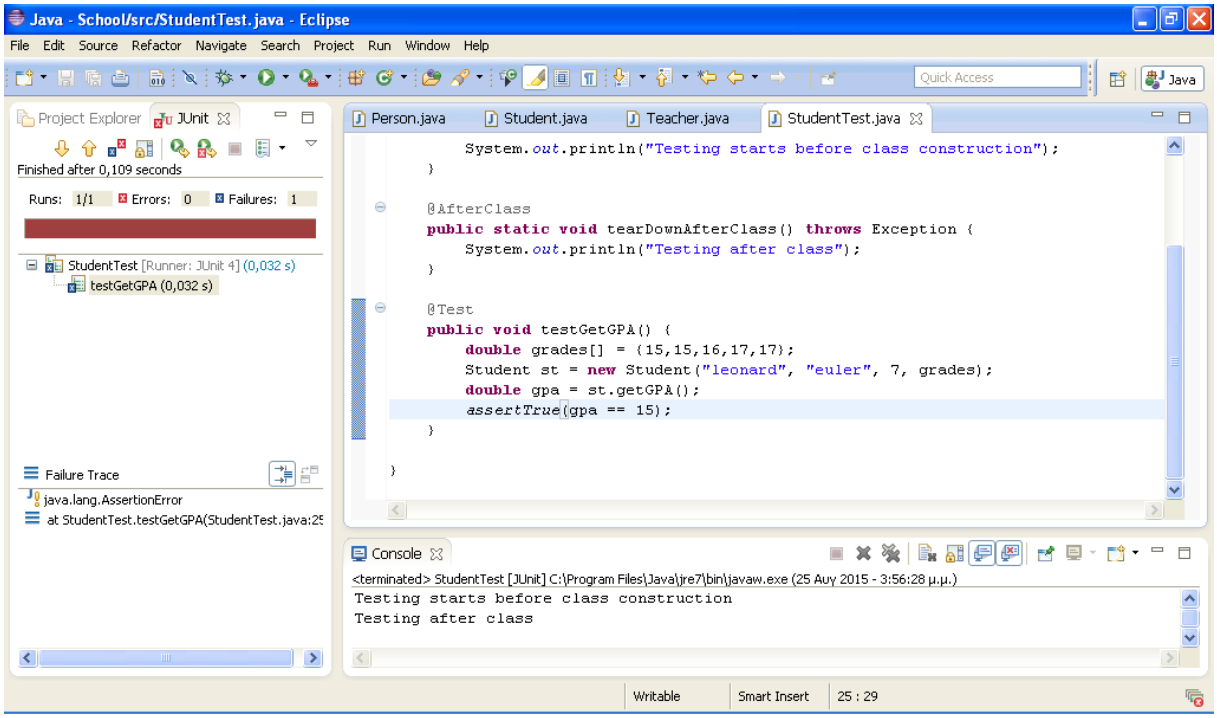
Στη συνέχεια επιλέγουμε την κλάση Student. Έχουμε επιλέξει να υλοποιήσουμε τις μεθόδους setUpBeforeClass() και tearDownAfterClass() που εκτελούνται πριν και μετά την κλάση. Στη συνέχεια επιλέγουμε ποιες μεθόδους από την κλάση Student θέλουμε να συμπεριλάβουμε στις δοκιμές που θα τρέξουμε.



Πλέον έχουμε δημιουργήσει το δικό μας JUnit Test Case, το οποίο όπως βλέπετε παρακάτω είναι μια νέα κλάση η οποία περιέχει τις δυο μεθόδους που δηλώσαμε μαζί με μια νέα μέθοδο testGetGPA για τον έλεγχο ορθότητας της getGPA:



Για να ελέγξουμε τα αποτελέσματα σε κάποιο ενδιάμεσο στάδιο της εκτέλεσης του προγράμματος μπορούμε να χρησιμοποιήσουμε μια ποικιλία από κατηγορήματα assertions τα οποία προσθέτουμε στην αντίστοιχη μέθοδο. Στην προκειμένη περίπτωση χρησιμοποιούμε την assertTrue για να ελέγξουμε αν ισχύει η συνθήκη που ελέγχει την ορθότητα του αποτελέσματος:

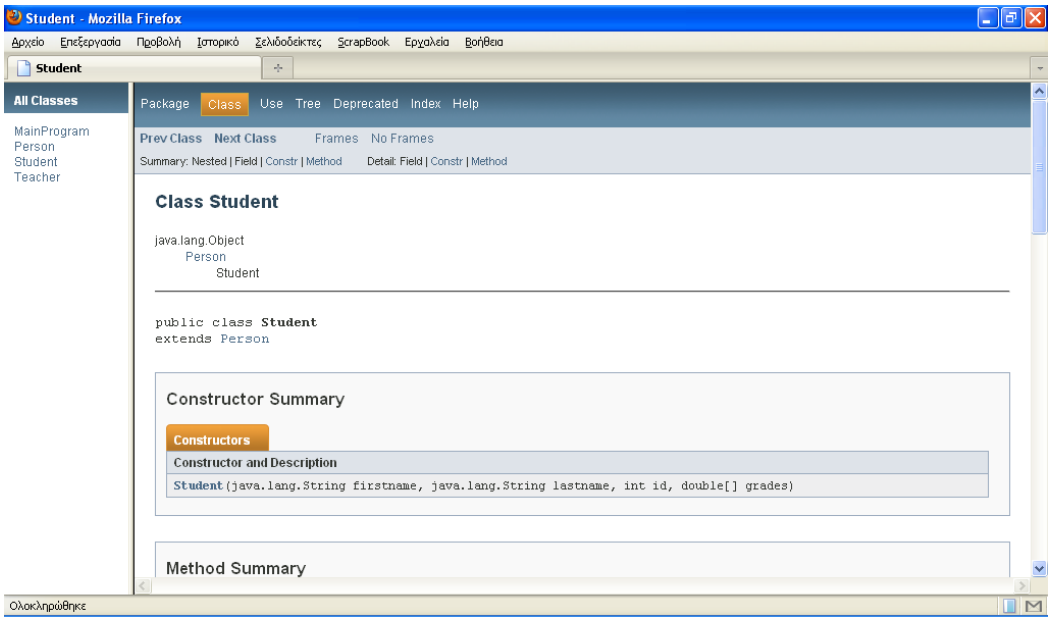


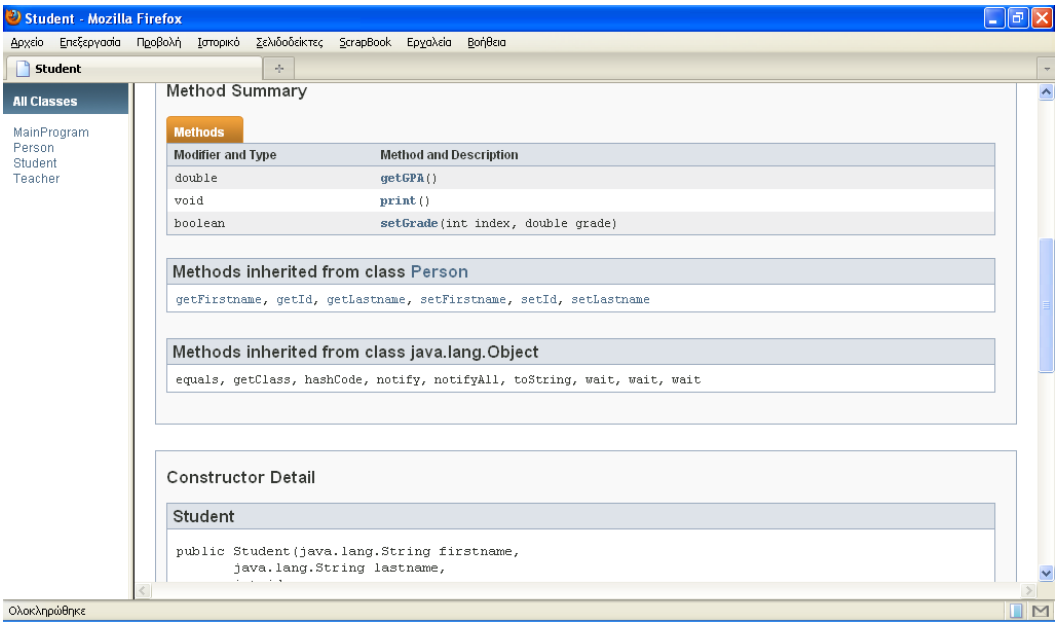
Όπως βλέπετε παραπάνω ο έλεγχος απέτυχε αφού ο μέσος όρος είναι 16 και όχι 15. Με αυτόν τον τρόπο μπορούμε να καταστρώσουμε ένα σχέδιο δοκιμών των κρίσιμων τμημάτων της εφαρμογής οι οποίες θα εκτελούνται κάθε φορά που θα κάνουμε αλλαγές στον κώδικά μας.

8.11. Τεκμηρίωση της Εφαρμογής

Η τεκμηρίωση της εφαρμογής μας αποτελεί την εικόνα της προς τον έξω κόσμο. Ένας προγραμματιστής που θέλει να μελετήσει την εφαρμογή μας, να χρησιμοποιήσει κάποιο τμήμα της ή να κάνει αλλαγές, το πρώτο πράγμα που θα κοιτάξει είναι η τεκμηρίωση. Στην περίπτωση της Java ένα σημαντικό τμήμα της τεκμηρίωσης παράγεται αυτόματα από το εργαλείο javadoc, το οποίο μπορεί να εκτελεστεί μέσα από το Eclipse. Για να παράγουμε την τεκμηρίωση του έργου μας επιλέγουμε Project → Generate Javadoc. Θα χρειαστεί να ορίσουμε τα αρχεία java για τα οποία θέλουμε να παραχθεί η τεκμηρίωση. Ίσως να χρειαστεί να δώσουμε και το μονοπάτι στον δίσκο μας για την εντολή javadoc.

Η τεκμηρίωση εξάγεται σε μορφή ιστοσελίδων ώστε να μπορεί να αναρτηθεί και στον παγκόσμιο ιστό και η πλοήγηση ανάμεσα στις κλάσεις και τι μεθόδους να είναι εύκολη. Παρακάτω δίνουμε την τεκμηρίωση που πήραμε για το δικό μας έργο.


Student - Mozilla Firefox
Αρχείο Επεξεργασία Προβολή Ιστορικό Σελιδοδείκτες ScrapBook Εργαλεία Βοήθεια
Student
All Classes
MainProgram
Person
Student
Teacher
Package Class Use Tree Deprecated Index Help
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method
Class Student
java.lang.Object
Person
Student
public class Student
extends Person
Constructor Summary
Constructors
Constructor and Description
Student(java.lang.String firstname, java.lang.String lastname, int id, double[] grades)
Method Summary
Ολοκληρώθηκε


Student - Mozilla Firefox
Αρχείο Επεξεργασία Προβολή Ιστορικό Σελιδοδείκτες ScrapBook Εργαλεία Βοήθεια
Student
All Classes
MainProgram
Person
Student
Teacher
Method Summary
Methods
Modifier and Type
Method and Description
double
getGPA()
void
print()
boolean
setGrade(int index, double grade)
Methods inherited from class Person
getFirstname, getId, getLastname, setFirstname, setId, setLastname
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
Student
public Student(java.lang.String firstname,
java.lang.String lastname,
Ολοκληρώθηκε

Βιβλιογραφία - Πηγές

- http://jtf.acm.org
- http://www.eclipse.org
- https://jdk9.java.net/
- http://www.greenfoot.org/
- http://www.bluej.org/
- http://cs.joensuu.fi/jeliot/
- http://junit.org/
- https://maven.apache.org/what-is-maven.html
- Bruce Eckel (2006), Thinking in Java, Pearson Education, MindView.
- Eric S. Roberts (2008), The Art and Science of Java: An Introduction to Computer Science, Addison – Wesley.
- Kölling, M. (2010). The Greenfoot Programming Environment. ACM Transactions on Computing Education, 10:182-196.
- Kölling, M. et al. (2003). The BlueJ system and its pedagogy. Computer Science Education, 13:249-268
- Kölling, M. (2015). Introduction to Programming with Greenfoot Second edition. Pearson Education
- Mertz, A., Slough, W., and Van Cleave, N. (2008). *Using the ACM Java Libraries in CS 1*, Journal of Computing Sciences in Colleges, Vol 24(1), pp 16-26
- Εισαγωγή στην γλώσσα προγραμματισμού Java, Σημειώσεις από το Εργαστήριο Πολυμέσων, των ΗΜΜΥ του ΕΜΠ.

ΕΝΟΤΗΤΑ 2α

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ (APPS) ΓΙΑ ANDROID

Κεφάλαιο 1

Γνωριμία με προγραμματιστικά περιβάλλοντα για το Android

Αντί Προλόγου

Το δεύτερο αυτό μέρος στις παρούσες σημειώσεις έχει σκοπό να εξοικειώσει τους μαθητές με την ανάπτυξη ολοκληρωμένων εφαρμογών για την πλατφόρμα Android.Η πλατφόρμα αυτή αφορά κυρίως στην περίπτωση των κινητών συσκευών και ταμπλετών όπου ο ρόλος της στην αγορά είναι σημαντικός. Παρουσιάζει δε σημαντικές ιδιοτυπίες καθώς όχι απλά επιτρέπει αλλά διευκολύνει ιδιαίτερα την επικοινωνία μεταξύ εφαρμογών, την αξιοποίηση τμημάτων της μίας από την άλλη και την από κοινού χρήση πόρων.

Οι επί μέρους ενότητες 2Α και 2Β του δεύτερου μέρους αποτελούν ένα αρθρωτό σύνολο. Προβλέπονται για αυτές δύο εναλλακτικές διάρκειες με τη μία περίπτωση να δίνει το μεγαλύτερο χρόνο στην ενότητα 2Α που αφορά στην ανάπτυξη εφαρμογών χρησιμοποιώντας πακέτα ανάπτυξης όπως το Eclipse ή το Android Studio και αξιοποίηση java και XML και τη δεύτερη να παραχωρεί μεγαλύτερο ρόλο στην ανάπτυξη εφαρμογών στο περιβάλλον ανάπτυξης AppInventor που χρησιμοποιεί προγραμματισμό ψηφίδων.

Είναι φανερό ότι η ενότητα 2Α έχει περισσότερα να προσφέρει στην εκπαίδευση επαγγελματιών ως προς την αγορά εργασίας. Η δυνατότητα από την άλλη απόκτησης ολοκληρωμένης εικόνας της διαδικασίας ανάπτυξης εφαρμογών, με εύκολο τρόπο, που προσφέρει η ενότητα 2Β έχει και αυτή σημαντικά εκπαιδευτικά οφέλη. Αφήνουμε την επιλογή στους συναδέλφους αλλά και στα εργαστήρια που διαθέτουν τα σχολεία τους:

Σκοπός του δεύτερου μέρους είναι η εφαρμοσμένη προσέγγιση στην ανάπτυξη εφαρμογών Android. Αν οι συνθήκες επιτρέπουν την αξιοποίηση των γνώσεων που αποκτήθηκαν στο 1^{ο} μέρος και εδώ αυτό θα ήταν για τους συγγραφείς το ιδανικό.

1. Γνωριμία με προγραμματιστικά περιβάλλοντα για το Android

Εισαγωγή

Το μοντέλο ανάπτυξης εφαρμογών για Android στην αγορά χρησιμοποιεί το Android SDK.

Για την ώρα θα επιλέξουμε να δουλέψουμε εκμεταλλευόμενοι τη δυνατότητα του Android SDK να συνδυαστεί με το περιβάλλον Eclipse, που είδαμε στην ενότητα προγραμματισμού με Java, χρησιμοποιώντας τα Android Development Tools (ADT) . Θα δημιουργήσουμε τους απαραίτητους πόρους για την ανάπτυξη εφαρμογής του Android στον υπολογιστή σας και θα δημιουργήσουμε πλήρως λειτουργικές εφαρμογές.

Όμως πρέπει να λάβουμε υπόψη ότι η υποστήριξη της google για τα ADT στο Eclipse έχει διακοπεί και έμφαση δίνεται πλέον στη νέα πλατφόρμα ανάπτυξης Android Studio (AS). Τα παρακάτω θα πρέπει επομένως να προσαρμοστούν στην νέα πραγματικότητα όταν το νέο περιβάλλον γίνει αρκετά σταθερό και ομογενοποιημένο για να αντικαταστήσει το υπάρχον. Ευτυχώς το AS επίσης χρησιμοποιεί το Android SDK καθιστώντας τη μετάβαση σχετικά εύκολη.

Στην πορεία θα γίνει σαφές ότι το Android αποτελεί τελείως διαφορετική περίπτωση από τη Java και αυτό γιατί δεν αποτελεί, ούτε περιλαμβάνει, γλώσσα προγραμματισμού. Αυτό σημαίνει ότι η ανάπτυξη εφαρμογών για αυτό θα απαιτήσει και κάποιο προγραμματισμό σε γλώσσα προγραμματισμού. Ευτυχώς ένα μεγάλο μέρος από τον προγραμματισμό μπορεί να γίνει σε Java. Ένα άλλο μέρος λειτουργεί καλά με μια δηλωτική γλώσσα σήμανσης - περιγραφής σελίδας, την XML, της οποίας τα απαραίτητα στοιχεία θα παρουσιαστούν όπου χρειάζονται. Εναλλακτικά η υλοποίηση μπορεί να γίνει εξολοκλήρου με την γλώσσα ψηφίδων του περιβάλλοντος AppInventor, με τους περιορισμούς που μια τέτοια γλώσσα βάζει.

Στόχοι

Οι μαθητές να μπορούν να:

- Επιλέγουν περιβάλλον ανάπτυξης εφαρμογής με βάση τις ανάγκες και τη διαθέσιμη υποδομή
- Αναγνωρίζουν τα βασικά στοιχεία του περιβάλλοντος ανάπτυξης που θα χρησιμοποιήσουν
- Να μπορούν να πλοηγηθούν σε αυτό
- Να δημιουργήσουν ένα έργο στο περιβάλλον αυτό

Ενότητες Κεφαλαίου

- Το έργο (project) και οι διαφορές του στα διάφορα περιβάλλοντα
- Βασικά στοιχεία περιβαλλόντων ανάπτυξης για Android
- Διαμόρφωση και πλοήγηση στο περιβάλλον ανάπτυξης
- Τύποι έργων (Built types)
- Δημιουργία project σε ένα από τα περιβάλλοντα

1.1. Το έργο (project) και οι διαφορές του στα διάφορα περιβάλλοντα

Ένα στοιχείο που προκαλεί σύγχυση στη μετάβαση ανάμεσα στα διάφορα περιβάλλοντα ανάπτυξης εφαρμογών είναι η ονοματολογία. Όμως μπορεί να είναι και η ίδια η δομή του περιβάλλοντος διαφορετική. Τέλος, όταν δουλεύεις σε διαφορετικά λειτουργικά ή πλατφόρμες (π.χ. Windows, Linux, Android στις διάφορες εκδόσεις τους), υπάρχουν διαφορές στους τρόπους που υλοποιούνται διάφορα στοιχεία της γλώσσας (π.χ. Java) που επέλεξες να χρησιμοποιείς .

Όταν ξεκινάμε να χτίσουμε μια εφαρμογή σε ένα περιβάλλον, στην πραγματικότητα ξεκινάμε ένα έργο ανάπτυξης. Αυτό το έργο στο περιβάλλον Eclipse υλοποιείται σε έναν ενιαίο «χώρο δουλειάς» που λέγεται workspace και περιλαμβάνει και στοιχεία που δεν αφορούν αποκλειστικά το συγκεκριμένο έργο. Στο Android Studio το έργο ονομάζεται main project και είναι λίγο πιο στενά ορισμένο.

Στοιχεία που προσδιορίζουν το είδος του έργου και τον τρόπο που προσεγγίζει το λειτουργικό το έργο μας τα λέμε μεταδεδομένα ή ιδιότητες. Στο Eclipse οι σχετικές πληροφορίες βρίσκονται στο αρχείο project.properties. Στο Android Studio αποθηκεύονται σε μια περιοχή (.idea) με λίγο διαφορετικό τρόπο.

Όταν χτίζουμε εφαρμογές σπανίως ανακαλύπτουμε ξανά τον τροχό. Κυρίως γιατί αυτό θέλει χρόνο που δεν μας περισσεύει. Για να μπορέσουμε να χρησιμοποιήσουμε την εμπειρία άλλων προγραμματιστών υπάρχουν πολλοί τρόποι, ένας από τους πιο χρήσιμους είναι η χρήση εξωτερικών βιβλιοθηκών. Αυτές στο Eclipse ονομάζονται Reference Libraries ενώ στο AS External Libraries. Σε κάθε περίπτωση περιέχουν έτοιμα τμήματα κώδικα, αντικείμενα και διαδικασίες που κάνουν τη ζωή μας ευκολότερη.

Προφανώς σε ένα περιβάλλον μπορεί να υπάρχουν παραπάνω από ένα έργα. Κάθε φορά δουλεύουμε σε ένα από αυτά και συγκεκριμένα στο φάκελό του (Project Folder / Project module). Σε αυτόν περιλαμβάνονται όλα τα στοιχεία του. Για την ώρα το Eclipse μόνο δίνει το πλεονέκτημα να έχεις σε ένα έργο πολλαπλές εφαρμογές (apps), πράγμα πολύ χρήσιμο αν θες να υλοποιήσεις εκδοχές του έργου για διαφορετικές χρήσεις ή διαφορετικά κοινά. Αντίστοιχα το AS αντιμετωπίζει το ίδιο θέμα με την επιλογή rebuild the project που μας επιτρέπει μεταβολές στις ιδιότητες / μεταδεδομένα της εφαρμογής μας. Τα επί μέρους έργα του Eclipse αντιστοιχούν στα modules του AS. Για κάθε module μπορούν να αντιστοιχούν πολλαπλές βιβλιοθήκες. Επίσης οι παραλλαγές ενός έργου/ εφαρμογής στο AS, για τις οποίες θα μιλήσουμε αργότερα, διαχειρίζονται το θέμα των εκδόσεων με ένα ιδιαίτερα πρωτότυπο τρόπο.

Τα παραπάνω ανταποκρίνονται στη βασική δομή των φακέλων του έργου στα δύο περιβάλλοντα ανάπτυξης. Μέσα στο φάκελο του κυρίως έργου σας, τον οποίο θα ξεχωρίσετε γιατί έχει το όνομα που εσείς δώσατε, υπάρχουν κάποιοι υποφάκελοι. Ανάμεσα σε αυτούς στην περίπτωση του Eclipse θα βρείτε τους υποφακέλους build, libs, src και res. Στην περίπτωση του AS στον src περιλαμβάνονται τόσο ένας υποφάκελος java που αντιστοιχεί στον src του Eclipse όσο και ένας res που αντιστοιχεί στον αντίστοιχο του Eclipse.

1.2. Βασικά στοιχεία περιβαλλόντων ανάπτυξης για Android

1.2.1. Android SDK with Eclipse

Το Eclipse, όπως έχουμε δει, είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για Java. Μαζί με την προσθήκη (plugin) ADT το Eclipse παρέχει ένα περιβάλλον για το σχεδιασμό, τον προγραμματισμό, τη δόμηση και τελικά το τρέξιμο Android εφαρμογών. Εδώ παρουσιάζονται τα βασικά στοιχεία του περιβάλλοντος ή αν θέλετε της διεπαφής χρήστη του.

1.2.2. Δημιουργία ενός έργου για Android

Αν ανοίξουμε το Eclipse, έχοντας εγκαταστήσει προηγούμενα το ADT-plugin, μπορούμε να προχωρήσουμε στη δημιουργία ενός έργου για Android το οποίο θα επεκτείνουμε αργότερα. Χωρίς πολλές λεπτομέρειες, θα δημιουργήσουμε ένα βασικό έργο. Το Eclipse, όπως γνωρίζουμε, διαμορφώνει κάθε εφαρμογή που δημιουργείτε ως ένα έργο, με την κωδικοποίηση και άλλους πόρους που σχετίζονται με την εφαρμογή να αποθηκεύονται και διαχειρίζονται μέσα από έναν κατάλογο για το έργο στο χώρο εργασίας σας. Η δημιουργία ενός νέου έργου για το Android περιλαμβάνει μερικά βήματα, ανάλογα με όσα έχετε εφαρμόσει γενικά για την Java. Αξίζει να δούμε πως διαμορφώνεται το περιβάλλον όταν εργαζόμαστε πάνω σε μια εφαρμογή (app) για Android.

Δραστηριότητα 1.2

Δημιουργείστε στο περιβάλλον που χρησιμοποιείτε ένα έργο. Δώστε του ένα όνομα που να έχει νόημα και να θυμίζει αυτό που σκοπεύετε να κάνει η εφαρμογή που θα φτιάξετε. Δείτε τους φακέλους του και επιβεβαιώστε ότι υπάρχουν τα στοιχεία που αναφέραμε στην παράγραφο 1.1.

1.3. Διαμόρφωση και πλοήγηση στο περιβάλλον ανάπτυξης

Το περιβάλλον ανάπτυξης τους Eclipse ονομάζεται πάγκος εργασίας (Workbench). Το όνομα είναι χαρακτηριστικό του σκοπού της πλατφόρμας, που είναι η ανάπτυξη εφαρμογών. Ανοίγοντας το περιβάλλον διακρίνουμε διάφορες αυτόνομες περιοχές, σε ξεχωριστά παράθυρα, μαρκαρισμένες και με ετικέτες, που συνθέτουν την εικόνα της τρέχουσας διεπαφής του πάγκου εργασίας μας. Οι περιοχές αυτές ονομάζονται όψεις (Views).

Μπορούμε να ορίσουμε μια όψη ως ένα οπτικό συστατικό εντός του πάγκου εργασίας. Συνήθως χρησιμοποιείται για την πλοήγηση σε μια λίστα ή δέντρο πληροφοριών (όπως οι πόροι του πάγκου εργασίας), ή για την ανάρτηση ιδιοτήτων που αφορούν τον ενεργό επιμελητή (editor). Ο επιμελητής δεν είναι ένας απλός επεξεργαστής κειμένου ενώ συχνά μπορεί να περιλαμβάνει σημαντικούς αυτοματισμούς. Οι τροποποιήσεις που γίνονται σε μια όψη αποθηκεύονται άμεσα.

Ένας επιμελητής είναι επίσης ένα οπτικό συστατικό του πάγκου εργασίας. Συνήθως χρησιμοποιείται για να επεξεργαστείτε ή να περιηγηθείτε σε έναν πόρο. Η οπτική παρουσίαση μπορεί να είναι κείμενο ή διάγραμμα. Οι επιμελητές ξεκινούν όταν επιλέξετε (κλικ) έναν πόρο σε μια όψη. Τροποποιήσεις που γίνονται σε έναν επιμελητή ακολουθούν τον παραδοσιακό κύκλο ζωής άνοιγμα-σώσιμο-κλείσιμο.

Αν έχετε ανοιχτό ένα έργο και δεν έχετε προβεί σε κάποιες τροποποιητικές ενέργειες θα πρέπει να βλέπετε την όψη Package Explorer στα αριστερά και την όψη Outline στα δεξιά, στη μέση θα υπάρχει ένας επιμελητής (editor) και από κάτω του μια σειρά από θέσεις στις οποίες εμφανίζονται συμπληρωματικές πληροφορίες. Μπορείτε να προσθέσετε όψεις επιλέγοντας διαδοχικά Window→Show View→ την επιλογή σας. Για να καταστήσετε την όψη μη ορατή αρκεί να την κλείσετε όπως οποιοδήποτε παράθυρο. Το ίδιο ισχύει για ελαχιστοποίηση και μεγιστοποίηση.

Μπορείτε να δημιουργήσετε και δικές σας όψεις αν το κρίνετε απαραίτητο. Πληροφορίες για τον τρόπο που μπορείτε να κατασκευάσετε μια όψη μπορείτε να βρείτε στη θέση: https://eclipse.org/articles/viewArticle/ViewArticle2.html.

Κάποιες ιδιότητες τόσο των όψεων όσο και των επιμελητών είναι κοινές. Άλλωστε λειτουργούν ως παράθυρα μέσα στο παράθυρο του πάγκου εργασίας. Από κοινού αποτελούν τα μέρη του πάγκου εργασίας και της εκάστοτε οπτικής του. Τα μέρη μπορεί να είναι ενεργά ή ανενεργά, ενώ μόνο ένα από αυτά είναι ενεργό κάθε φορά. Το ενεργό μέρος είναι φωτισμένο (highlighted) και αυτό αποτελεί τον στόχο συνήθων λειτουργιών όπως αντιγραφή, διαγραφή και

επικόλληση. Όταν δεν είναι ενεργός ένας επιμελητής, οι ορατές όψεις μπορεί να περιλαμβάνουν στοιχεία που βασίζονται στον προηγούμενο ενεργό επιμελητή.

Μια οπτική (perspective) είναι μια ομαδοποίηση από όψεις και επιμελητές στο παράθυρο του πάγκου εργασίας. Μπορεί να υπάρχουν ταυτόχρονα παραπάνω από μια μέσα στον πάγκο εργασίας. Κάθε μία οπτική περιλαμβάνει ένα ή περισσότερα τμήματα. Μέσα στο ίδιο παράθυρο κάθε οπτική μπορεί να έχει διαφορετικές προβολές αλλά όλες οι οπτικές μοιράζονται το ίδιο σύνολο επιμελητών.

Στην πλατφόρμα Eclipse υπάρχουν δύο κύρια επίπεδα : το επίπεδο του μοντέλου και το επίπεδο της διεπαφής του χρήστη. Το μοντέλο , γνωστό ως το Workspace , είναι μια συλλογή των πόρων (έργα , φάκελοι και αρχεία). Η διεπαφή χρήστη , ή Workbench , ορίζει την παρουσίαση αυτών των πόρων . Μέσα στον πάγκο εργασίας η οπτική χρησιμοποιείται για τον έλεγχο της ορατότητας των διάφορων στοιχείων στο μοντέλο και τη διεπαφή χρήστη .

Μια οπτική ελέγχει τί βλέπετε εκείνη τη στιγμή από το μοντέλο (ποιο έργο, φάκελο και αρχεία) ενώ ταυτόχρονα καθορίζει τις ορατές ενέργειες και όψεις μέσα σε ένα παράθυρο. Οι οπτικές επίσης παρέχουν μηχανισμούς για εργασία εστιασμένη σε συγκεκριμένου τύπου αλληλεπίδραση με τους πόρους της πλατφόρμας Eclipse, στο φιλτράρισμα της πληροφορίας και στο multi-tasking.

Οι οπτικές αφορούν όπως είπαμε συγκεκριμένες εργασίες. Επειδή τον περισσότερο χρόνο σας θα τον καταναλώνετε στην ανάπτυξη των εφαρμογών σας η προεπιλεγμένη οπτική είναι η λεγόμενη «Java». Εναλλακτικά, για την εκσφαλμάτωση υπάρχει η οπτική DDMS. Από την πρώτη στη δεύτερη μπορείτε να μεταφερθείτε επιλέγοντας διαδοχικά Window→ Open Perspective → DDMS.

Υπάρχουν διαθέσιμες οπτικές για όλες τις εργασίες και καλό είναι να τις χρησιμοποιείτε. Αν θέλετε να αλλάξετε κάτι μέσα στην οπτική που δουλεύεται αυτό γίνεται τροποποιώντας τα μέρη που περιλαμβάνει. Μπορείτε όμως να κατασκευάσετε και δική σας οπτική (πράγμα που δεν προτείνεται σε αυτή τη φάση). Πληροφορίες για τη δημιουργία οπτικών μπορείτε να βρείτε στη θέση: https://eclipse.org/articles/using-perspectives/PerspectiveArticle.html.

1.4. Τύποι έργων (BuildTypes)

Τα αρχεία APK είναι τα αντίστοιχα των exe αρχείων για το Android. Η κύρια διαφορά τους είναι ότι περιλαμβάνουν και όλα τα αρχεία πόρων στο ίδιο πακέτο . Το όνομά τους σημαίνει ακριβώς αυτό: Application Package (πακέτο εφαρμογής).

Οι τύποι κατασκευής (BuildTypes) στο Eclipse επιτρέπουν την παραγωγή από τον ίδιο κώδικα εναλλακτικών αρχείων APK. Αυτό είναι ιδιαίτερα χρήσιμο κατά την ανάπτυξη της εφαρμογής οπότε και η εκσφαλμάτωση καθορίζει τον επιθυμητό τρόπο λειτουργίας της εφαρμογής. Για αυτόν άλλωστε το λόγο, οι προϋπάρχουσες επιλογές είναι δύο: debug και release. Φυσικά μπορούμε να ορίσουμε τις δικές μας.

Μια έκδοση για γενική χρήση, σαν αυτή που θα βάζαμε στο Play store, έχει απαιτήσεις ασφάλειας κατά την εγγραφή, υπογραφής από τον κατασκευαστή, ενώ η καταγραφή των συνδέσεων (σε έναν κατάλογο logcat) μπορεί να αντιμετωπιστεί διαφορετικά από ότι σε μια έκδοση για εκσφαλμάτωση. Αντίστοιχα η έκδοση εκσφαλμάτωσης λειτουργεί καλύτερα και επιτρέπει γρηγορότερο εντοπισμό λαθών χωρίς τις αντίστοιχες προβλέψεις. Οι δυο εκδόσεις μπορεί να λειτουργούν ταυτόχρονα με ελαφρά διαφορετικά τμήματα κώδικα (packages) με κατάλληλα διαφοροποιημένα ονόματα (packageNames).

Διαφοροποιήσεις πάνω σε μια εφαρμογή υλοποιούν και οι ρυθμίσεις των λεγόμενων γεύσεων (Product Flavors). Εδώ υπάρχει μια μόνο προϋπάρχουσα ρύθμιση χωρίς όνομα. Συνήθως με αυτές τις ρυθμίσεις διαχωρίζονται οι δωρεάν με τις πληρωνόμενες εκδόσεις. Η διαφορετική ονομασία αφορά σε εκδόσεις που μοιράζονται τον ίδιο κύριο κώδικα, αλλά διαφορετικές εκδόσεις

μερικών αρχείων του πηγαίου κώδικα ή αρχείων πόρων. Σε κάποιες περιπτώσεις, όπως για τις δωρεάν γεύσεις, μπορεί απλώς να παραλείπονται τμήματα κώδικα.

Συνολικά λοιπόν ένα έργο μπορεί να αντιστοιχεί σε πολλαπλές εφαρμογές και επομένως αρχεία APK. Οι εφαρμογές αυτές αποτελούν τις λεγόμενες παραλλαγές του. Μία παραλλαγή (BuildVariant) καθορίζεται από τις ρυθμίσεις γεύσης (Product flavor) και τις ρυθμίσεις κατασκευής (BuildType) . Έτσι, μόνο με τους δύο προϋπάρχοντες τύπους κατασκευής και τις δύο συνήθεις γεύσεις, προκύπτουν τέσσερις παραλλαγές της εφαρμογής:

Δωρεάν για εκσφαλμάτωση (free –debug)

Δωρεάν γενικής χρήσης (free-release)

Πληρωνόμενη για εκσφαλμάτωση (paid-debug)

Πληρωνόμενη γενικής χρήσης (paid-release)

1.5. Δημιουργία project σε ένα από τα περιβάλλοντα

Αν έχετε ολοκληρώσει τη Δραστηριότητα 1 έχετε ήδη καλύψει την αντίστοιχη ύλη. Αν όχι, τώρα είναι η ώρα να την ξεκινήσετε.

Κεφάλαιο 2

Εισαγωγή στις Android Εφαρμογές (Apps)

2. Εισαγωγή στις Android Εφαρμογές (Apps)

Στόχοι

Οι μαθητές να μπορούν να:

- Αναγνωρίζουν με ποιο τρόπο περιγράφεται μια εφαρμογή και τον κατάλογό της
- Αναγνωρίζουν την αναγκαιότητα για ποικιλία πόρων ώστε να δημιουργηθεί μια θελκτική εφαρμογή
- Αναφέρουν τις βασικές κατηγορίες πόρων για μια Android εφαρμογή
- Αναγνωρίζουν τη σκοπιμότητα χρήσης διαφορετικών γλωσσών (με έμφαση στην Java και την XML) σε τμήματα του έργου.
- Εντοπίζουν μέσα σε ένα έργο τον πηγαίο κώδικα
- Εντοπίζουν και διαβάζουν το δηλωτικό έργου
- Αναγνωρίζουν την υποκείμενη γλώσσα προγραμματισμού σε διάφορα τμήματα κώδικα

Ενότητες Κεφαλαίου

- Δομή Εφαρμογής
- Πηγαίος κώδικας και ο φάκελος του
- Πόροι (σχεδιαστικοί, γραφικοί, δεδομένων)
- Δηλωτικό Έργου (Manifest)
- Είσοδος / Έξοδος δεδομένων
- Το Android δεν είναι γλώσσα

2.1 Δομή Εφαρμογής

Στο κεφάλαιο αυτό θα δούμε περιληπτικά τη δομή μιας Android εφαρμογής και τα πιο σημαντικά στοιχεία της. Θα τα ξαναδούμε αναλυτικά σε επόμενα κεφάλαια αλλά είναι σκόπιμο να ρίξουμε εδώ μια γενικότερη ματιά για να έχουμε μια ενιαία εικόνα των τμημάτων πάνω στα οποία θα δουλέψουμε στη συνέχεια αλλά και μια αντίληψη του πού πηγαίνει τι κατά την ανάπτυξη.

Η κυρίως εφαρμογή μας υλοποιείται με Java στον αντίστοιχο φάκελο (src στον φάκελο Package Explorer) αλλά ταυτόχρονα ρυθμίζονται διάφορες επιλογές της διεπαφής, των πόρων, του τρόπου λειτουργίας στα αντίστοιχα σημεία. Οι σχετικές ρυθμίσεις συνήθως κωδικοποιούνται σε κώδικα XML και αποθηκεύονται σε σχετικά αντικείμενα.

Στις επόμενες ενότητες θα δούμε τα σημεία αυτά ένα προς ένα.

2.2 Πηγαίος κώδικας και ο φάκελος του

Επειδή δουλεύουμε με αντικειμενοστρεφή τρόπο ένα μεγάλο μέρος του κώδικά μας θα βρίσκεται σε αρχεία κλάσεων (class files). Ιδιαίτερα αυτό αφορά το τμήμα του κώδικα που υλοποιεί τη διεπαφή με το χρήστη – την παρουσίαση της εφαρμογής, την ανταπόκριση στις ενέργειές του, τις σχετικές επεξεργασίες.

Κάθε φορά που δημιουργούμε ένα έργο, δημιουργούμε ταυτόχρονα και ένα πακέτο με τα σχετικά αρχεία Java κλάσεων. Μια εφαρμογή μπορεί να περιλαμβάνει περισσότερα από ένα πακέτα και το καθένα από αυτά μπορεί να περιλαμβάνει πολλά αρχεία κλάσεων. Αυτά είναι που θα μας επιτρέψουν τη δημιουργία των κατάλληλων αντικειμένων που θα αναλάβουν τις διάφορες διεργασίες που θα υλοποιήσουμε.

Στο Eclipse, αν ανοίξουμε το φάκελο Package Explorer για την εφαρμογή στην οποία εργαζόμαστε, μέσα στο φάκελο src θα μας εμφανίσει λογικά το πακέτο που δημιουργήσαμε κατά τη δημιουργία του project. Μέσα στο πακέτο αυτό θα πρέπει να βρίσκεται το αρχείο για το Activity class. Το αρχείο αυτό θα πρέπει να είναι επίσης ανοιχτό στον διορθωτή (editor). Πρόκειται για τη βασική διαδικασία της εφαρμογής μας που εκκινεί με το άνοιγμα της εφαρμογής. Θεωρείται η κύρια κλάση της εφαρμογής μας και αναφέρεται και στο δηλωτικό της όπως θα δούμε παρακάτω.

Για κάθε οθόνη της εφαρμογής μπορούμε να υλοποιήσουμε επιπλέον Activity κλάσεις. Προφανώς, όσο προχωράμε στην υλοποίηση, πρέπει να προσθέσουμε κι άλλες κλάσεις, άρα και τα αρχεία τους, στο πακέτο ή τα πακέτα μας.

Αν κοιτάξουμε την Activity class στον διορθωτή θα δούμε ότι ήδη περιλαμβάνει κάποιον προϋπάρχοντα τυποποιημένο κώδικα. Για παράδειγμα υπάρχει η μέθοδος onCreate, που περιλαμβάνει τον κώδικα που θα δημιουργηθεί μαζί με την αρχική δραστηριότητα (activity). Μέσα στη μέθοδο αυτή βρίσκουμε τον ακόλουθο κώδικα:

```
setContentView(R.layout.activity_main);
```

Με τη γραμμή αυτή έχει προσδιοριστεί ότι η διάταξη (layout) που δημιουργείται με την έναρξη της εφαρμογής βρίσκεται στο «R.layout.activity_main». Εκεί δηλαδή προσδιορίζεται τι θα βλέπουν οι χρήστες. Η σύνταξη του ονόματος «R.type.name» μας λέει κάποια πράγματα:

Το R μας λέει ότι αναφερόμαστε σε πόρους της εφαρμογής. Ο τύπος –εδώ layout– μας προσδιορίζει τον τύπο του στοιχείου που στην περίπτωσή μας είναι μια διάταξη. Αθροιστικά μας λένε ότι το στοιχείο θα αποθηκευτεί στον κατάλογο res / layout και μάλιστα με όνομα «activity_main».

Αν επιστρέψουμε στον Package Explorer και αναπτύξουμε τον κατάλογο res θα πρέπει να βρούμε μέσα μια σειρά από υποκαταλόγους. Όλους αυτούς τους έχουν αυτόματα δημιουργήσει το Eclipse και το ADT με τη δημιουργία του Android έργου μας. Ανάμεσα στους υποκαταλόγους θα βρούμε και τον υποκατάλογο layout και σε αυτόν πάλι το στοιχείο «activity_main».

Φυσικά υπάρχουν και άλλοι κατάλογοι που μπορούμε να χρησιμοποιήσουμε τους οποίους θα δούμε αργότερα. Αργότερα επίσης θα προσθέσουμε στο «activity_main» κώδικα για να διαχειριστούμε την αλληλεπίδραση με το χρήστη.

2.3 Πόροι (σχεδιαστικοί, γραφικοί, δεδομένων)

Υπάρχουν πολλές κατηγορίες πόρων που χρησιμοποιούμε στις εφαρμογές μας. Ήδη στην προηγούμενη ενότητα είδαμε έναν πόρο διάταξης (layout resource), που δημιουργήθηκε με τη δημιουργία του έργου. Ο κατάλογος στον οποίο δημιουργήθηκε ήταν ο res / layout . Σε αυτόν τον κατάλογο μπορούν να αποθηκεύονται αρχεία διάταξης είτε για τις τυχόν πολλαπλές οθόνες δραστηριοτήτων είτε και για επιμέρους στοιχεία της διεπαφής χρήστη όπως κουμπιά, εικόνες και κείμενο.

Η δημιουργία κλάσης δραστηριότητας (Activity) με τον τρόπο που είδαμε είναι η μία επιλογή για ρυθμίσεις αυτού του είδους. Η άλλη απαιτεί τον ορισμό της διάταξης στα πλαίσια του κώδικα Java. Η πρώτη λύση έχει το πλεονέκτημα ότι μπορούμε να έχουμε μια οπτική αναπαράσταση κατά το σχεδιασμό ενώ η δεύτερη ότι η κλάση δημιουργείται δυναμικά κατά την εκτέλεση.

Στο κυρίως αρχείο διάταξης υπάρχουν δομές XML. Η XML είναι, όπως έχουμε πει και αλλού, μια δηλωτική γλώσσα σήμανσης αντίστοιχη με την γλώσσα περιγραφής σελίδας HTML που έχετε δει στην Β΄ τάξη στον Σχεδιασμό ιστότοπων.

Στον κατάλογο πόρων (res) υπάρχουν και υποκατάλογοι με τον χαρακτηρισμό «σχεδιάσιμα» (drawable) στο όνομά τους. Αυτός είναι ο τρόπος για να αποθηκευτούν τα αρχεία εικόνας που χρησιμοποιεί η εφαρμογή. Μπορεί να οριστούν με κώδικα XML περιγράφοντας χαρακτηριστικά όπως το σχήμα, το χρώμα, η υφή κλπ. Μπορεί όμως και να είναι αρχεία εικόνας κάποιου συμβατικού τύπου π.χ. JPEG.

Με την εισαγωγή ενός αρχείου εικόνας στους υποκαταλόγους σχεδιάσιμων αυτό μπορεί να χρησιμοποιηθεί από τα αρχεία διάταξης ή μέσα στον κώδικα της εφαρμογής σας, επιτρέποντας την εισαγωγή γραφικών στοιχείων στη διεπαφή χρήστη. Οι εικόνες που χρησιμοποιούμε στα έργα μας μπορούν να έχουν διαφορετικές εκδόσεις για κάθε πυκνότητα ανάλυσης, τοποθετημένες στους αντίστοιχους υποκαταλόγους.

Αυτός είναι και ο λόγος που υπάρχουν πολλαπλοί υποκατάλογοι για τα γραφικά στοιχεία: η ανάγκη για εξυπηρέτηση δηλαδή πολλαπλών συσκευών Android και επομένωςπολλαπλών χαρακτηριστικών οθόνης. Η οργάνωση γίνεται με βάση την πυκνότητα(density) της ανάλυσης γενικά, με τα γραφικά στοιχεία να κατανέμονται στους υποκαταλόγους με βάση την κατάταξη σε χαμηλής έως εξαιρετικά υψηλής συχνότητας (5 κατηγορίες).

Πάντα στον κατάλογο πόρων (res), υπάρχουν και ορισμένοι υποκατάλογοι με τον χαρακτηρισμό «τιμές» (values) στον τίτλο τους. Αυτά είναι για τις τιμές των δεδομένων που χρησιμοποιούνται στο πλαίσιο της εφαρμογής σας, είτε αφορούν σε συμβολοσειρές είτε σε αριθμούς είτε σε διαστάσεις και στυλ. Οι υποκατάλογοι τιμών περιέχουν αρχεία XML στα οποία καταλογογραφούνται μία ή περισσότερες τιμές. Κάθε εγγραφή περιλαμβάνει το όνομα και μια τιμή και μπορεί να χρησιμοποιηθεί για αναφορά σε συγκεκριμένες τιμές μέσω του ονόματος. Μπορούμε έτσι να αποθηκεύσουμε τα κείμενα που αφορούν σε στοιχεία της διεπαφής χρήστη, π.χ. τί αναγράφεται σε ένα κουμπί, με όνομα το οποίο μπορούμε να χρησιμοποιούμε σε πολλά σημεία και αν χρειαστεί να το τροποποιήσουμε μόνο εδώ.

Για άλλη μια φορά οι πολλαπλοί υποκατάλογοι έχουν σκοπό να επιτρέπουν την προσαρμογή των τιμών σε συγκεκριμένα μεγέθη οθόνης και επίπεδα API. Μια τιμή που μπορεί να χρησιμοποιηθεί σε διάφορες συσκευές αποθηκεύεται συνήθως στον απλό φάκελο τιμών (values).

2.4 Δηλωτικό Έργου (Manifest)

Στον κύριο φάκελο για την εφαρμογή σας, θα δείτε το αρχείο δήλωσης έργου (Manifest). Αν το επιλέξουμε με διπλό κλικ μπορούμε να επιλέξουμε από τη γραφική διεπαφή κάποιες καρτέλες στο κάτω μέρος του παραθύρου. Ανάμεσα σε αυτές υπάρχει η καρτέλα του XML κώδικα του δηλωτικού. Σε αυτή (AndroidManifest.xml) ρυθμίζονται πολλά χαρακτηριστικά της εφαρμογής.

Και στο δηλωτικό, υπάρχουν ήδη κάποια τμήματα κώδικα, που έχουν δημιουργηθεί από το περιβάλλον με την δημιουργία της εφαρμογής. Για παράδειγμα το στοιχείο χρήσης του SDK (uses-SDK element) βρίσκεται εδώ και περιλαμβάνει τις ρυθμίσεις σχετικά με το ελάχιστο επίπεδο API και το επίπεδο στόχου/ μετάφρασης API. Φυσικά υπάρχει επίσης το στοιχείο εφαρμογής (application element) με το οποίο προσδιορίζονται ο launcher, το όνομα της εφαρμογής μας και η αναφορά στο στοιχείο της βασικής ή κύριας δραστηριότητας (Activity). Επίσης, εδώ καταγράφονται οι υποστηριζόμενες συσκευές αλλά και τα τμήματά της εφαρμογής που λειτουργούν ως υπηρεσίες παρασκηνίου (background services).

Υπάρχει επίσης η δυνατότητα να προσθέσουμε με δικό μας κώδικα επιπλέον στοιχεία. Εδώ είναι που μπορούν να εισαχθούν εναλλακτικές δραστηριότητες (activities). Επίσης εδώ μπορούμε να περιλάβουμε το στοιχείο προσβάσεων (uses-permission element) όπου καταλογογραφούνται τα δικαιώματα πρόσβασης της εφαρμογής μας π.χ. στο διαδίκτυο ή στην κάμερα της συσκευής.

2.5 Είσοδος / Έξοδος δεδομένων

Σε όλες τις εφαρμογές, επομένως και στις εφαρμογές στο Android, το κύριο μέσο επικοινωνίας της εφαρμογής είναι η διεπαφή με το χρήστη. Από αυτήν εισάγονται και σε αυτήν εξάγονται δεδομένα. Η διαχείριση αυτών των δεδομένων γίνεται στην περίπτωσή μας μέσω κατάλληλων αντικειμένων. Τα δεδομένα όμως μπορεί να φυλάσσονται σε αρχεία και αυτά να είναι δημόσια ή ιδιωτικά. Από τα αρχεία μπορεί να δέχεται δεδομένα η εφαρμογή μας ή να τα αποθηκεύει σε αυτά.

Ακόμα η εφαρμογή μας μπορεί να ανταλλάσσει δεδομένα με βάσεις δεδομένων. Για το σκοπό αυτό υπάρχουν κατάλληλες έτοιμες κλάσεις μέσα από τις οποίες μια Android εφαρμογή διαχειρίζεται την επικοινωνία με τη βάση δεδομένων και εμφανίζει τα σχετικά αποτελέσματα. Η επικοινωνία με το διαδίκτυο είναι επίσης διαδεδομένη για εφαρμογές Android. Σε αυτή την περίπτωση εκτός από τις κατάλληλες κλάσεις χρησιμοποιούμε και τυποποιημένες προσβάσεις (permissions).

Με τις περισσότερες από αυτές τις μορφές εισόδου/ εξόδου θα ασχοληθούμε σε βάθος στο κεφάλαιο 7.

Συχνά όμως οι εφαρμογές επικοινωνούν και με το σύστημα. Αυτό συμβαίνει σε όλα τα λειτουργικά συστήματα και τις πλατφόρμες. Στο Android συμβαίνει συνήθως μέσα από εκπομπές που μπορεί να περιλαμβάνουν οδηγίες για ενέργειες τις οποίες ονομάζουμε προθέσεις (Intents) και ίσως κάποια δεδομένα. Το ενδιαφέρον βρίσκεται στο γεγονός ότι στην περίπτωση του Android αυτού του είδους η επικοινωνία επεκτείνεται και μεταξύ εφαρμογών επιτρέποντας την συνεργασία των εφαρμογών ή και τμημάτων τους. Με αυτές τις επικοινωνίες και τις περιπλοκές τους θα ασχοληθούμε στα κεφάλαια 6, 9 και 10.

Το είδος των δεδομένων που χρησιμοποιούμε και η διαχείρισή τους έχει σημαντική επίδραση στα επίπεδα ασφαλείας των δεδομένων αυτών. Και για αυτό το ζήτημα θα επεκταθούμε αργότερα.

2.6 Το Android δεν είναι γλώσσα

Είδαμε στις προηγούμενες ενότητες τον πηγαίο κώδικα, τους πόρους και το δηλωτικό μιας Android εφαρμογής. Στη συγκεκριμένη περίπτωση ο κύριος κώδικας ήταν όπως είδαμε γραμμένος σε Java ενώ ο κώδικας που προέκυψε από τις ρυθμίσεις των πόρων και του δηλωτικού ήταν γραμμένος σε XML. Οι δυο αυτές διαφορετικές γλώσσες αρκούν για να περιγραφεί μια εφαρμογή για περιβάλλον Android.

Το Android, όπως είπαμε και αλλού, είναι μια πλατφόρμα για την οποία υπάρχουν διάφορες υλοποιήσεις γλωσσών προγραμματισμού. Οι δύο που χρησιμοποιούμε εδώ αρκούν για να περιγράψουν την ανάπτυξη μιας εφαρμογής για το συγκεκριμένο περιβάλλον. Προφανώς υπάρχουν και άλλες επιλογές όπως θα δούμε σε επόμενη ενότητα του μαθήματος.

Εκείνο που είναι σκόπιμο εδώ είναι να αντιληφθούμε τον τρόπο με τον οποίο οι εφαρμογές μας επικοινωνούν με το λειτουργικό σύστημα και τη δομή που αυτό απαιτεί να έχουν. Εφόσον γνωρίζουμε αυτά τα στοιχεία, είναι φανερό ότι αρκεί να γνωρίζουμε επίσης το συντακτικό μιας άλλης γλώσσας για να υλοποιήσουμε και με εκείνη. Πιο συχνά βέβαια θα έχουμε ένα συνδυασμό, καθώς οι ρυθμίσεις που εδώ εκφράστηκαν σε XML, γενικά απαιτούν μια γλώσσα σήμανσης για να επικοινωνηθούν εύκολα στο λειτουργικό.

Κεφάλαιο 3

Εισαγωγή στη Σχεδίαση διεπαφής Χρήστη και αλληλεπίδρασή της με το χρήστη

3. Εισαγωγή στη Σχεδίαση διεπαφής Χρήστη και αλληλεπίδρασή της με το χρήστη

Το κεφάλαιο 3 μας εισάγει στη σχεδίαση διεπαφών χρήστη εξοικειώνοντας τους μαθητές με τα δομικά τους στοιχεία και τον τρόπο που αλληλοεπιδρούν με το χρήστη και την εφαρμογή.

Στόχοι

Οι μαθητές να μπορούν να:

- Αναγνωρίζουν τα βήματα σχεδίασης μιας διεπαφής χρήσης
- Αναγνωρίζουν τα στοιχεία της διεπαφής χρήστη
- Δημιουργούν και εισάγουν οπτικά στοιχεία στο σχεδιασμό της εφαρμογής τους
- Επιλέγουν, δημιουργούν και εισάγουν γραφικά στοιχεία στο σχεδιασμό της εφαρμογής τους
- Αναγνωρίζουν τα βασικά στοιχεία της γλώσσας XML
- Τροποποιούν κώδικα XML για τις ανάγκες της εφαρμογής τους
- Αναγνωρίζουν τις ενέργειες του χρήστη – και αν είναι εφικτό τις εισόδους από αισθητήρες – ως γεγονότα και ενεργοποιητές μεθόδων

Ενότητες Κεφαλαίου

- Σχεδίαση (Layout) και οπτικά στοιχεία (Views)
- Γραφικά στοιχεία και εισαγωγή τους
- Εισαγωγή στην αλληλεπίδραση χρήστη
- Ταυτοποίηση στοιχείων διεπαφής χρήστη
- Γεγονότα, αναμονές, ανταπόκριση
- Βασικά στοιχεία XML και χρήση της στη σχεδίαση διεπαφής

3.1 Σχεδίαση (Layout) και οπτικά στοιχεία (Views)

Όταν εργαζόμαστε σε πόρους, και ιδιαίτερα στη διεπαφή χρήστη, εργαζόμαστε συνήθως με XML. Αν ανοίξουμε το κύριο αρχείο σχεδίασης της εφαρμογής μας, το Eclipse έχει ήδη εισάγει ένα βασικό πρώτο σχεδιασμό για να δουλέψουμε. Επιλέγοντας την ετικέτα επεξεργασίας XML (XML editing tab) μπορούμε να επεξεργαστούμε τον κώδικα.

Αυτό είναι ευκολότερο από ότι φαίνεται στο συγκεκριμένο περιβάλλον (Eclipse IDE και ADT) καθώς υπάρχει η διευκόλυνση των προτεινόμενων εγγραφών (εισόδων κειμένου) ενόσω πληκτρολογούμε. Ο υπάρχον σχεδιασμός θα είναι κάπως έτσι:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
     android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Όπως είναι φανερό, εδώ έχουμε μια δενδροειδή ή εμφωλευμένη δομή, με το ριζικό στοιχείο να είναι ένα στοιχείο διάταξης που περιέχει μια όψη που ονομάζεται TextView και μας επιτρέπει να δώσουμε μια συμβολοσειρά η οποία θα φαίνεται πάνω στο στοιχείο σχεδίασης. Θεωρητικά πολλαπλές διατάξεις μπορούν επίσης να εμφωλευτούν η μία μέσα στην άλλη, δημιουργώντας μια συνθετότερη ιεραρχία γονέων-παιδιών..

Εδώ το ριζικό στοιχείο είναι του τύπου RelativeLayout και περιλαμβάνει κάποια χαρακτηριστικά (attributes) της σχεδίασης του βασικού στοιχείου διάταξης, ή αν θέλετε της αρχικής οθόνης, όπως το ύψος, το πλάτος και τα περιθώρια. Συχνά τα σχεδιαστικά αντικείμενα στο Android αναφέρονται ως αντικείμενα ομαδοποίησης όψεων (GroupView objects) και φυσικά περιλαμβάνουν τουλάχιστον μία και συνήθως περισσότερες όψεις. Το TextView, όπως όλα τα Views ή όψεις είναι ένα ορατό και δυνητικά διαδραστικό στοιχείο της εφαρμογής μας.

Από το Outline View στα δεξιά του επιμελητή (editor) μπορούμε να ανοίξουμε μια άλλη διεπαφή με τα στοιχεία του αρχείου μας. Με διπλό κλικ στη λίστα μας πηγαίνει στη θέση του στοιχείου στον κώδικα, ενώ τα στοιχεία γονείς μπορούν να εκταθούν ή να συρρικνωθούν, πράγμα ιδιαίτερα χρήσιμο σε σχεδιαστικά στοιχεία αυξημένης πολυπλοκότητας.

Δραστηριότητα 3.1

Ας αντικαταστήσουμε τον αυτόματο κώδικα που βλέπουμε εδώ με δικό μας. Για να γίνει αυτό πρέπει να διαγράψουμε τον υπάρχοντα. Θα δουλέψουμε με XML και πάλι ασχολούμενοι για την ώρα μόνο με στοιχεία παρουσίασης και επικοινωνίας με το χρήστη. Αν δεν θέλετε να λειτουργήσετε πρακτικά και προτιμάτε να καταλαβαίνεται γιατί κάνετε κάτι, μπορείτε να ρίξετε τώρα μια ματιά στην ενότητα 3.6.

3.2 Γραφικά στοιχεία και εισαγωγή τους

Τα γραφικά στοιχεία τα διαχειριζόμαστε μέσω της προβολής Γραφική Διάταξη (Graphical Layout), στην οποία μεταφερόμαστε από την αντίστοιχη ετικέτα. Εδώ μπορείτε να δείτε μια οπτική αναπαράσταση της διάταξης σας όσο τη σχεδιάζετε . Καλό είναι να έχουμε υπόψη, πάντως, ότι η εικόνα που παίρνουμε μέσα από τη Γραφική Διάταξη δεν είναι πάντοτε ακριβής. Για να αποκτήσουμε καλύτερη εικόνα πως πραγματικά δείχνει η διάταξή μας σε μία συσκευή όταν τρέχει πρέπει να τη φορτώσουμε είτε σε πραγματικές είτε σε εικονικές συσκευές.

Στο πάνω μέρος της προβολής Γραφική Διάταξη υπάρχει μια αναπτυσσόμενη λίστα από την οποία μπορείτε να επιλέξετε διάφορες συσκευές για να δείτε πως δείχνει η διάταξή σας επάνω τους, καθώς και τα εργαλεία για να αλλάξετε προσανατολισμό και ζουμ . Η παλέτα στα αριστερά σας επιτρέπει να επιλέξετε στοιχεία UI και να τα σύρετε επάνω στη διάταξη. Ακόμη και όταν χρησιμοποιείτε τέτοια γραφικά εργαλεία , είναι πιθανό να χρειαστεί να επεξεργαστείτε, κάποια στιγμή, τον κώδικα XML που προκύπτει .

Πειραματιστείτε με τους ελέγχους της Διάταξης γραφικών όσο σχεδιάζετε τις διατάξεις σας . Υπάρχουν πολλά άλλα στοιχεία διάταξης και ρυθμίσεις.

Δραστηριότητα 3.2

Με τη μέχρι τώρα διαμόρφωση της εφαρμογής θα παρατηρήσετε ότι τα ορατά στοιχεία στη διάταξή μας εμφανίζονται πολύ κοντά στην άνω άκρη της οθόνης. Για να διορθώσετε αυτό το πρόβλημα προσθέστε το χαρακτηριστικό περιθώριο (margin) στο LinearLayout με το γνωστό τρόπο. Δώστε του τιμή «15dp». Οι τιμές σε dp προσαρμόζονται στην πυκνότητα (density) της οθόνης του χρήστη αυτόματα. Αποθηκεύστε και επιστρέψτε στη Γραφική Διάταξη για να ελέγξετε το αποτέλεσμα.

3.3 Εισαγωγή στην αλληλεπίδραση χρήστη

Η αλληλεπίδραση του χρήστη με την εφαρμογή μπορεί να γίνει με πολλούς τρόπους. Σε κάθε περίπτωση η αλληλεπίδραση συνδέεται με κάποιο συγκεκριμένο αντικείμενο στην οθόνη του. Τα ορατά αντικείμενα όπως έχουμε δει ήδη λέγονται όψεις (Views) και περιλαμβάνουν πολλούς τύπους όπως π.χ. τα κουμπιά (buttons). Ο χρήστης μπορεί, ανάλογα με τη συσκευή που χρησιμοποιεί, να χρησιμοποιήσει το ποντίκι για να κάνει κλικ ή διπλό κλικ ή να περάσει από πάνω τους ή στην περίπτωση οθόνης αφής να ακουμπήσει με το δάχτυλο την οθόνη αφής στη θέση τους, σύντομα ή με διάρκεια, ή να γλιστρήσει επάνω τους .

Η εφαρμογή πρέπει να μπορεί να παρακολουθήσει τις ενέργειες αυτές, τις οποίες ονομάζουμε γεγονότα χρήστη, για να υπάρχει αλληλεπίδραση. Και ο μόνος τρόπος για αυτό είναι να προγραμματίσουμε την αναμονή για και την αντίδραση στο καθένα από αυτά. Αυτό θα γίνει στην κλάση δραστηριότητα (Activity) της Java όπου θα προστεθεί κατάλληλος κώδικας για κάθε περίπτωση. Ο κώδικας που θα προσθέσουμε πρέπει να κάνει τρία πράγματα: να αναγνωρίζει διαφορετικά στοιχεία της διεπαφής χρήστη και να διακρίνει μεταξύ τους, να παρακολουθεί για την εμφάνιση γεγονότων χρήστη και, τέλος, να προσδιορίζει την ανταπόκριση σε αυτά.

3.4 Ταυτοποίηση στοιχείων διεπαφής χρήστη

Για να μπορεί να γίνει ταυτοποίηση των στοιχείων της διεπαφής χρήστη θα δώσουμε στο κάθε στοιχείο τη δική του μοναδική ταυτότητα και το δικό του μοναδικό όνομα. Αφού γίνει αυτό θα μπορούμε να αναφερόμαστε στην αντίστοιχη όψη είτε από κώδικα XML είτε από τη Java με το όνομα αυτό.

Για να διασφαλίσουμε τη μοναδικότητα της ταυτότητας, ειδικά σε μεγάλες εφαρμογές, έχουμε στη διάθεσή μας τη σύνταξη «@+id» που κατάλληλα τοποθετημένη στα χαρακτηριστικά ενός στοιχείου προκαλεί το Android να δημιουργήσει μια νέα ταυτότητα (ID) πόρου στο αρχείο πόρων του έργου R.Java και να τη συνδέσει με ένα επίσης μοναδικό για την εφαρμογή όνομα για το ίδιο στοιχείο.

Για να χρησιμοποιηθεί το στοιχείο πρέπει να προσδιοριστεί και ο τύπος του. Αυτό θα γίνει μέσα στην κλάση της κύριας δραστηριότητας (MainActivity) ορίζοντας το στοιχείο σαν μεταβλητή του συγκεκριμένου τύπου και μετά, αν ο τύπος χρησιμοποιείται για πρώτη φορά, εισάγοντας τον τύπο από αυτούς που διαθέτει το Android στα android.widget. Ας δούμε ένα παράδειγμα.

Δραστηριότητα 3.4

Τροποποιείστε το πρόγραμμά σας ώστε να είναι εφικτή η ταυτοποίηση ενός στοιχείου του τύπου κουμπί (Button).

3.5 Γεγονότα και χειρισμός τους

Τα γεγονότα χρήστη ανήκουν σε κατηγορίες ανάλογα με τον τύπο τους. Το Android παρακολουθεί και εντοπίζει τέτοια γεγονότα πάνω σε όψεις (Views) μόνο εφόσον του το ζητήσουμε. Υπάρχουν για το σκοπό αυτό κατάλληλες διεπαφές που αναμένουν (ή ακούν για) γεγονότα. Οι διεπαφές αυτές ονομάζονται στα αγγλικά Listeners και ακούν για συγκεκριμένες κατηγορίες γεγονότων η κάθε μία.

Υπάρχουν διάφοροι τρόποι για να υλοποιηθούν οι διεπαφές αυτές με τον απλούστερο να είναι η υλοποίηση από την ίδια την κλάση δραστηριότητας (Activity class). Για περισσότερα στοιχεία σχετικά με τους listeners δες στη θέση http://developer.android.com/guide/topics/ui/ui-events.html#EventListeners.

Δραστηριότητα 3.5

Ενσωματώστε τη δυνατότητα αναμονής και διαχείρισης για γεγονότα της κατηγορίας «κλικ» από την κλάση Activity, χρησιμοποιώντας τη διεπαφή OnClickListener και ελέγχοντας από την ID την όψη που «πατήθηκε».

3.6 Βασικά στοιχεία XML και χρήση της στη σχεδίαση διεπαφής

Όπως έχει γίνει φανερό όλα τα αρχεία πόρων που χρησιμοποιούμε στο Android, περιλαμβανομένου και του δηλωτικού. χρησιμοποιούν μια σήμανση XML. Χρειάζεται επομένως μια στοιχειώδης γνώση της δομής της γλώσσας αυτής για να προστεθούν ή τροποποιηθούν στοιχεία πόρων.

Η XML είναι μια δηλωτική γλώσσα προγραμματισμού, της ειδικής κατηγορίας των γλωσσών σήμανσης. Είναι δηλαδή μια γλώσσα που αποθηκεύει περιγραφές δεδομένων καταγράφοντας σε μορφή δέντρου τη δομή τους καταλήγοντας σε τιμές δεδομένων που χαρακτηρίζουν τα αντικείμενα.

Η XML μοιάζει πολύ στη λειτουργία της με την HTML που είναι επίσης μια γλώσσα σήμανσης. Παρότι χρησιμοποιούνται με πολλούς διαφορετικούς τρόπους συχνότερα λειτουργούν

σαν βάσεις δεδομένων συχνά για την μοντελοποίηση της εμφάνισης ιστοσελίδων αν και στην περίπτωση της XML η χρήση είναι πολύ ευρύτερη, περιλαμβάνοντας και τον ορισμό αντικειμένων.

Εφόσον στην XML, οι τιμές δεδομένων αποθηκεύονται σε στοιχεία, ένα στοιχείο θα περιέχει τουλάχιστον μια ετικέτα ανοίγματος και μία ετικέτα κλεισίματος, όπως σε αυτό το παράδειγμα:

```
<product>Onion</product>
```

Κάθε χαρακτηριστικό έχει ένα όνομα και μια τιμή, με την τιμή να περικλείεται σε εισαγωγικά. Τα στοιχεία μπορούν επίσης να περιέχουν και άλλα στοιχεία:

```
<section name="food">
<product type="vegetable">Onion</product>
<product type="fruit">Banana</product>
</section>
```

Εδώ το στοιχείο τμήμα (section) ονομάζεται γονέας και τα δύο προϊόντα (product) παιδιά. Μεταξύ τους τα δύο προϊόντα λέμε ότι είναι αδέρφια (siblings). Σε ένα έγγραφο XML, πρέπει να υπάρχει ένα στοιχείο ρίζα, η οποία ενεργεί ως γονέας σε όλα τα στοιχεία που περιέχονται ή είναι «φωλιασμένα», μέσα σε αυτό. Αυτό δημιουργεί τη δενδροειδή μορφή που έχουμε αναφέρει νωρίτερα με τα στοιχεία των παιδιών κάθε στοιχείου να διακλαδώνονται από αυτό. Ένα στοιχείο παιδί μπορεί να είναι επίσης γονέας αν έχει δικά του στοιχεία παιδιά.

Μια ενδιαφέρουσα XML δομή που θα χρησιμοποιήσουμε είναι το στοιχείο αυτόματο κλείσιμο. Η δομή αυτή δεν έχει διακριτές ετικέτες ανοίγματος και κλεισίματος:

```
<order number="12345" customerID="a4d45s"/>
```

Εδώ είναι ο χαρακτήρας «/» στο τέλος της δήλωσης που την κλείνει.

Κεφάλαιο 4

Προγραμματίζοντας σε Java για το Android

4. Προγραμματίζοντας σε Java για το Android

Στόχοι

Οι μαθητές να μπορούν να:

.• Οργανώνουν την εφαρμογή τους σε κλάσεις και αντικείμενα που επηρεάζονται από γεγονότα

- Επεκτείνουν ήδη υπάρχουσες κλάσεις
- Αναγνωρίζουν και απαριθμούν τα κύρια γεγονότα που σχετίζονται με την εφαρμογή τους
- Αναλύουν την εφαρμογή τους σαν ένα σύνολο μεθόδων κάθε μια από τις οποίες ενεργοποιείται από ένα γεγονός
- Σχεδιάζουν κατάλληλες μεθόδους για την διαχείριση αυτών των γεγονότων
- Εφαρμόζουν στην εφαρμογή τους την επαναχρησιμοποίηση κώδικα μέσα από την κληρονομικότητα
- Δημιουργούν τα δικά τους πακέτα ως βιβλιοθήκες λογισμικού

Ενότητες Κεφαλαίου

- Προαπαιτούμενα από Java για το Android
- Συντακτικό και δομές ελέγχου
- Κλάσεις και αντικείμενα
- Κληρονομικότητα και διεπαφές

4.1 Προαπαιτούμενα από Java για το Android

Για να δημιουργήσουμε μια εφαρμογή Android θα πρέπει να έχουμε σίγουρα κάποιες βασικές γνώσεις της γλώσσας προγραμματισμού Java. Με αυτόν τον τρόπο θα έχουμε καλύτερη κατανόηση της ανάπτυξης των εφαρμογών στην πλατφόρμα Android. Επίσης κάποιες βασικές γνώσεις XML θα ήταν χρήσιμες γιατί οι Android εφαρμογές στηρίζονται πολύ στην XML για να περιγράψουν την βασική λειτουργία της εφαρμογής και κυρίως τα γραφικά στοιχεία.

Σ' αυτό το κεφάλαιο θα γίνει μια σύντομη επανάληψη των βασικών εννοιών της γλώσσας προγραμματισμού Java. που είναι απαραίτητες για την υλοποίηση εφαρμογών Android

4.2 Συντακτικό και δομές ελέγχου

Οι γενικοί συντακτικοί κανόνες της java που θα πρέπει να γνωρίζει κάποιος είναι:

Η γλώσσα Java είναι case sensitive. Αυτό σημαίνει ότι κάνει διάκριση ανάμεσα σε πεζά και κεφαλαία γράμματα. Άρα το Μ δεν είναι το ίδιο με το m.

Όλες οι εντολές και δεσμευμένες (πιασμένες) λέξεις που χρησιμοποιεί η Java γράφονται με πεζά γράμματα (π.χ.: while, if, class, public).

Στο τέλος κάθε εντολής προστίθεται το ελληνικό ερωτηματικό (;).

Το όνομα των κλάσεων αρχίζει με κεφαλαίο γράμμα π.χ. class Test .

Αν το όνομα της κλάσης αποτελείται από περισσότερες της μιας λέξεις, τότε κάθε λέξη αρχίζει με κεφαλαίο γράμμα π.χ. class MyFirstJavaClass.

Το όνομα των μεθόδων αρχίζει με πεζό γράμμα. Αν το όνομα της μεθόδου αποτελείται από περισσότερες της μιας λέξεις, τότε κάθε λέξη, εκτός της πρώτης, αρχίζει με κεφαλαίο γράμμα π.χ. myMethodName().

Τα ονόματα των αρχείων του προγράμματος πρέπει να έχουν το ίδιο όνομα με αυτό της κλάσης. Όταν αποθηκεύεται το αρχείο προστίθεται στο τέλος η κατάληξη «.java». Αν το όνομα του αρχείου και της κλάσης δεν ταιριάζουν, τότε το πρόγραμμα δεν μεταγλωττίζεται. Π.χ. αν το όνομα της κλάση είναι 'MyFirstJavaProgram' , το αρχείο πρέπει να έχει το όνομα 'MyFirstJavaProgram.java'.

Όλα τα ονόματα κλάσεων, μεταβλητών και μεθόδων πρέπει ν' αρχίζουν με γράμμα ή το σύμβολο του δολαρίου ($) ή μια κάτω παύλα (_).

Η java χρησιμοποιεί τους προσδιοριστές πρόσβασης : public , protected, private

Οι δομές ελέγχου είναι:

```
while (συνθήκη)
{
    ...
}
```

```
do
{
    ...
}
```

```
while (συνθήκη);
```

```
for (αρχική τιμή αριθμητή; συνθήκη; βήμα αρίθμησης)
{
      ...
}
```

```
if(συνθήκη)
{
      ...
}
```

```
if(συνθήκη)
{
      ...
}
     else
      {
      ...
      }
```

```
if (συνθήκη1)
{
      ...
}
    else if (συνθήκη2)
    {
      ...
    }
else
{
      ...
}
```

```
switch(n)
{
case 1:
    ...
    break;
case 2:
    ...
    break;
case 3:
    ...
    break;
default:
    ...
}
```

4.3 Κλάσεις και αντικείμενα

Μια κλάση (class) είναι ένα γενικευμένο πρότυπο για μια ομάδα αντικειμένων τα οποία έχουν παρόμοια χαρακτηριστικά. Όλα τα στοιχεία από τα οποία αποτελείται ο κώδικας μας πρέπει βρίσκονται μέσα σε μια κλάση.

Κάθε κλάση έχει τη μορφή: class ονομα_κλάσης { ... }

Για παράδειγμα: class Person { ... }

Μια κλάση μπορεί να έχει έναν προσδιοριστή πρόσβασης όπως public ή private

Για παράδειγμα: public class Person { ... }

Μια κλάση περιέχει έναν συνδυασμό από πεδία (fields), μεθόδους (methods) και κατασκευαστές (constructors)

Ένα πεδίο στη δήλωσή του περιέχει:

- τον προσδιοριστή πρόσβασης (public, private, protected)
- τον τύπο δεδομένων (int, String κλπ)
- το όνομα του πεδίου

Για παράδειγμα: public int myVar;

Μια μέθοδος περιέχει:

- τον προσδιοριστή πρόσβασης (public, private, protected)
- τον τύπο δεδομένων που επιστρέφει (int, String κλπ) ή void αν δεν επιστρέφει τιμή
- το όνομα της μεθόδου

- καμία ή περισσότερες παραμέτρους
- το σώμα της μεθόδου που ορίζεται από τα άγκιστρα: { ... }
- μια μέθοδος μπορεί να περιέχει τις δικές της τοπικές μεταβλητές

Ένα αντικείμενο (instance) είναι σαφής αναπαράσταση μιας κλάσης. Μια εφαρμογή σε java χρησιμοποιεί μια σειρά από αντικείμενα για να τρέξει. Τα αντικείμενα αλληλοεπιδρούν μεταξύ τους καλώντας τις μεθόδους τους έτσι ώστε να έχουμε το επιθυμητό αποτέλεσμα.

Τα τρία βήματα για τη δημιουργία ενός αντικειμένου από μια κλάση, είναι:

- Δήλωση μιας μεταβλητής με τύπο δεδομένων το αντικείμενο που θα δημιουργηθεί
- Η χρήση της λέξης new για τη δημιουργία του αντικειμένου
- Η κλήση του κατασκευαστή του αντικειμένου.

Π.χ. House h= new House() ;

4.4 Κληρονομικότητα και διεπαφές

Μια κλάση που παράγεται από μια άλλη κλάση καλείται υποκλάση (subclass).

Η κλάση από την οποία παράγεται μια κλάση λέγεται υπερκλάση (superclass) ή γονική κλάση (parent).

Μια υποκλάση κληρονομεί (inherit) τις public και protected μεθόδους, τα πεδία της υπερκλάσης αλλά όχι τους κατασκευαστές, μπορεί όμως να έχει πρόσβαση σε αυτούς.

Μια υποκλάση για να έχει πρόσβαση στα private μέλη της υπερκλάσης χρησιμοποιεί τις public μεθόδους της υπερκλάσης.

Για να δηλώσουμε ότι μια κλάση, έστω SportsCar είναι υποκλάση ή κληρονομεί την κλάση Vehicle, γράφουμε: class SportsCar **extends** Vehicle.

Μια κλάση μπορεί να κληρονομεί μόνο μία άλλη κλάση η οποία μπορεί να κληρονομεί μια άλλη κλάση ... κ.λπ.

Μια διεπαφή (interface) είναι μια κλάση που περιέχει αποκλειστικά ορισμούς μεθόδων (δεν περιλαμβάνουν υλοποίηση μεθόδων) ή abstract μεθόδους. Ουσιαστικά ένα interface αποτελεί ένα αφηρημένο σκελετό - σχέδιο μιας κλάσης και αποτελεί ένα είδος συμβολαίου - υπόσχεσης ότι η κλάση που το υλοποιεί παρέχει την υλοποίηση του σχεδίου του δηλαδή των μεθόδων.

Μια διεπαφή δηλώνεται με τον προσδιοριστή interface .

Μια διεπαφή μπορεί να περιέχει μεταβλητές οι οποίες είναι υποχρεωτικά public, static και final.

Δεν μπορούμε να δημιουργήσουμε αντικείμενα από μια διεπαφή.

Μια διεπαφή μπορεί να κάνει extend μία ή περισσότερες διεπαφές.

Μια κλάση μπορεί να υλοποιεί (implements) μία ή περισσότερες διεπαφές. Μια κλάση που κάνει implement μια διεπαφή, πρέπει υποχρεωτικά να υλοποιεί όλες τις μεθόδους της διεπαφής.

Όλες οι μέθοδοι μιας διεπαφής θεωρούνται αυτόματα public και abstract.

Κεφάλαιο 5

Πόροι εφαρμογής (App)

5. Πόροι εφαρμογής (App)

Το κεφάλαιο 5 εξειδικεύει τις γνώσεις των μαθητών σε σχέση με τους πόρους που χρειάζεται μια εφαρμογή και τη μεταξύ τους αλληλεπίδραση.

Στόχοι

Οι μαθητές να μπορούν να:

- Αναγνωρίζουν βασικές κατηγορίες πόρων μιας εφαρμογής
- Σχεδιάζουν και βελτιστοποιούν μια διεπαφή χρήστη με χρήση αντικειμένων
- Σχεδιάζουν την ανταπόκριση σε γεγονότα με μεθόδους των κατάλληλων αντικειμένων
- Υλοποιούν κατάλληλες μεθόδους για την διαχείριση αυτών των γεγονότων
- Διαμορφώνουν κατάλληλα πλαίσια διαλόγου για επικοινωνία με τον χρήστη
- Υλοποιούν μεθόδους για τον χειρισμό γεγονότων

Ενότητες Κεφαλαίου

- Οικουμενικοί και προσανατολισμένοι στη συσκευή πόροι
- Γραφικοί πόροι
- Πόροι διάταξης και αλληλεπίδραση με γραφικούς πόρους και στοιχεία
- Άλλοι πόροι

5.1 Οικουμενικοί και προσανατολισμένοι στη συσκευή πόροι

Όπως έχει γίνει φανερό, πέρα από τις επιμέρους κατηγοριοποιήσεις των πόρων ανάλογα με το είδος, υπάρχει και ένας ουσιαστικός διαχωρισμός μεταξύ οικουμενικά χρησιμοποιούμενων και προσανατολισμένων στη συσκευή πόρων. Ήδη στη δομή του έργου και ειδικά στον κατάλογο των πόρων είδαμε πώς τα σχεδιάσιμα (drawable) αντικείμενα είναι συνήθως προσαρμοσμένα σε συγκεκριμένες πυκνότητες ανάλυσης, ενώ τοποθετούνται και σε αντίστοιχους υποκαταλόγους. Υπάρχουν και σχεδιάσιμα αρχεία που δεν είναι προσαρμοσμένα.

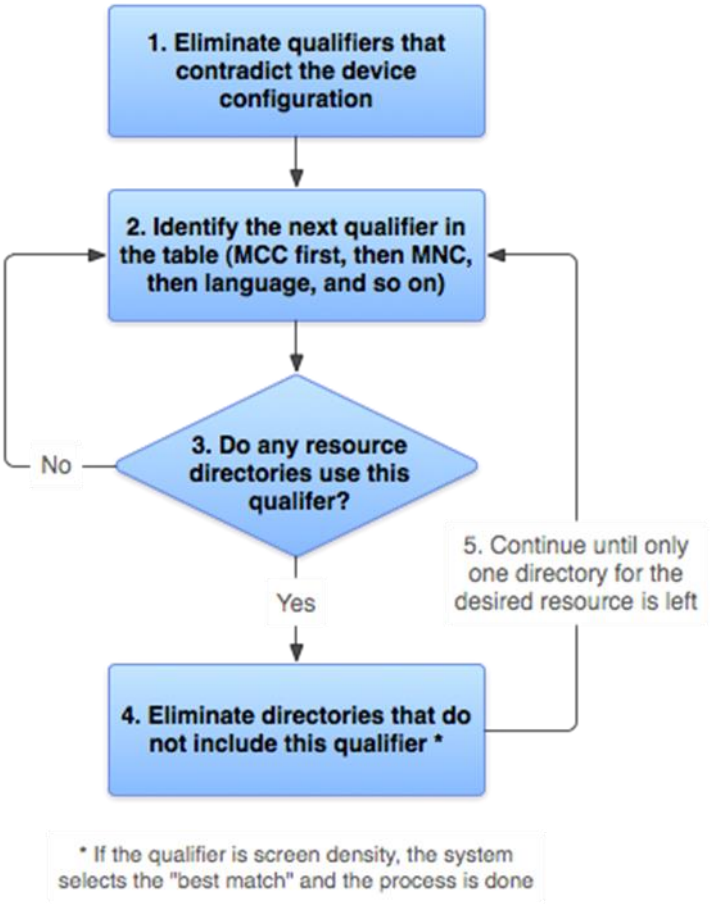
Τα ονόματα των υποκαταλόγων που περιλαμβάνουν προσαρμοσμένους πόρους περιλαμβάνουν προσδιορισμούς για να αναγνωρίζονται από το Android το οποίο αναγνωρίζει μια ποικιλία προσδιορισμούς για διάφορα ποιοτικά χαρακτηριστικά της συσκευής του χρήστη όπως η διάσταση και η διαδραστικότητα της οθόνης, η πυκνότητα ανάλυσης, το επίπεδο του API, η γλώσσα, η περιοχή κλπ. Οι προσδιορισμοί πρέπει να περιλαμβάνονται στο όνομα του υποκαταλόγου, γραμμένοι με τον τυποποιημένο τρόπο. Όταν αυτό συμβαίνει το Android χρησιμοποιεί τον αλγόριθμο του Σχήματος 5.1.1 για να επιλέξει πόρους από τον καταλληλότερο υποκατάλογο.

Μπορεί κανείς να προσθέσει υποκαταλόγους για συσκευές με συγκεκριμένα χαρακτηριστικά αρκεί να δώσει κατάλληλα ονόματα στους υποκαταλόγους αυτούς. Τα κατάλληλα ονόματα περιλαμβάνουν τον τύπο των πόρων που μπαίνει στην αρχή και μια σειρά προσδιορισμών που ακολουθούν διαχωρισμένοι από παύλες:

Όνομα πόρων – προσδιορισμός χαρακτηριστικών-προσδιορισμός χαρακτηριστικών…

Τόσο το όνομα πόρων όσο και οι προσδιορισμοί χαρακτηριστικών που μπορούν να είναι πολλαπλοί, πρέπει να είναι τυποποιημένοι για να αναγνωριστούν. Η σχετική τυποποίηση φαίνεται στους Πίνακες του Παραρτήματος Α και Παραρτήματος Β αντίστοιχα. Επίσης η σειρά με την οποία τοποθετούμε τους προσδιορισμούς είναι απαραίτητο να ακολουθεί την σειρά τους στον Πίνακα του Παραρτήματος Β για να λειτουργεί σωστά ο αλγόριθμος αναζήτησης.

Δυστυχώς οι εναλλακτικοί πόροι της ίδιας κατηγορίας δεν μπορούν να φωλιαστούν μέσα στον κατάλογο της κατηγορίας. Επίσης αν χρησιμοποιούμε τους ίδιους πόρους σε κάποιες από τις διαφορετικές κατηγοριοποιήσεις πρέπει να φτιάξουμε πολλαπλούς υποκαταλόγους. Για κάθε είδος προσδιορισμού ένας υποκατάλογος μπορεί να έχει έναν ή κανένα προσδιορισμό αλλά όχι δύο ή περισσότερους. Τέλος το Android αναγνωρίζει μόνο πηγές που είναι τοποθετημένες σε υποκαταλόγους και όχι κατευθείαν στον κατάλογο res.



Εικόνα 5.1.1 Αλγόριθμος προσδιορισμού υποκαταλόγου πηγών

5.2 Γραφικοί πόροι

Έχουμε ήδη αναφέρει τη δυνατότητα του περιβάλλοντος να εισάγει εικόνες των διάφορων γενικά διαθέσιμων τύπων όπως GIF, JPEG και PNG. Είναι απλά θέμα εισαγωγής τους στον κατάλληλο υποκατάλογο πόρων (κάποιον από τους υποκαταλόγους σχεδιάσιμων με κατάλληλους προσδιορισμούς) και αξιοποίησής τους από τον κώδικα των πόρων διάταξης.

Γενικά χρησιμοποιούμε διαφορετικούς και πάντως προσαρμοσμένους πόρους για κάθε κατηγορία πυκνότητας ανάλυσης. Σε κάποιες περιπτώσεις όμως μπορούν να χρησιμοποιηθούν οι ίδιοι πόροι για όλες τις συσκευές. Είδαμε νωρίτερα ένα παράδειγμα όπου ορίζοντας την απόσταση σε dp μπορέσαμε να απομακρύνουμε αντικείμενα από την κορυφή της οθόνης χωρίς να προσδιορίσουμε τις διαστάσεις της. Ο τρόπος για να το κάνουμε αυτό είναι ορίζοντας τα σχεδιάσιμα μέσω κώδικα (π.χ. XML).

Τέτοια γενικής χρήσης σχεδιάσιμα περιλαμβάνονται στον υποκατάλογο drawables, χωρίς προσδιορισμούς μετά το όνομα. Επειδή το Eclipse προτιμά να κατασκευάζει υποκαταλόγους εξειδικευμένους τουλάχιστον ως προς την πυκνότητα ανάλυσης μπορεί να χρειαστεί να κατασκευάσετε τον υποκατάλογο αυτό μέσα στο βασικό κατάλογο πηγών res. Ο τρόπος κατασκευής στο Eclipse δεν διαφέρει ιδιαίτερα από τον τρόπο κατασκευής υποκαταλόγου στα windows.

Δραστηριότητα 5.2

Σκοπός της δραστηριότητας αυτής είναι να δημιουργήσουμε γενικής χρήσης σχεδιάσιμα στον υποκατάλογο drawables χρησιμοποιώντας κώδικα XML. Συγκεκριμένα θα δημιουργήσουμε ένα γενικής χρήσης παραλληλόγραμμο.

Ένας σχεδιάσιμος πόρος είναι στην πραγματικότητα μια γενική έννοια για κάποιο γραφικό που μπορεί είτε να σχεδιαστεί στην οθόνη είτε να ενσωματωθεί σε έναν άλλο πόρο χρησιμοποιώντας χαρακτηριστικά όπως android: drawable στον κώδικα του πόρου που θα το περιλάβει. Ακολουθεί μια λίστα επιλογών δημιουργίας διαφορετικών τύπων σχεδιάσιμων όπως:

Bitmap File

Ένα σχεδιάσιμο αρχείο τύπου bitmap (.png, .jpg, ή.gif). Δημιουργεί ένα BitmapDrawable.

Nine-Patch File

Ένα σχεδιάσιμο αρχείο τύπου PNG με εκτάσιμες περιοχές για να επιτρέπει αλλαγή μεγέθους εικόνας με βάση το περιεχόμενο (.9.png). Δημιουργεί ένα NinePatchDrawable.

Layer List

Ένα σχεδιάσιμο που διαχειρίζεται μια σειρά άλλων σχεδιάσιμων. Αυτά σχεδιάζονται σε σειρά ώστε το αντικείμενο με το μεγαλύτερο δείκτη στον πίνακα (array) να σχεδιάζεται πάνω από τα προηγούμενα. Δημιουργεί ένα LayerDrawable.

State List

Ένα αρχείο XML που παραπέμπει σε διαφορετικά γραφικά bitmap για διαφορετικές καταστάσεις (για παράδειγμα, για να χρησιμοποιήσετε μια διαφορετική εικόνα όταν πατηθεί κάποιο κουμπί). Δημιουργεί ένα StateListDrawable.

Level List

Ένα αρχείο XML που καθορίζει ένα σχεδιάσιμα που διαχειρίζεται μια σειρά εναλλακτικών σχεδιάσιμων, στο καθένα από τα οποία αποδίδεται μια μέγιστη αριθμητική τιμή. Δημιουργεί ένα LevelListDrawable.

Transition Drawable

Ένα αρχείο XML που καθορίζει ένα σχεδιάσιμο που μπορεί να μεταβαίνει (cross-fade) από ένα σχεδιάσιμο σε ένα άλλο. Δημιουργεί ένα TransitionDrawable.

Inset Drawable

Ένα αρχείο XML που καθορίζει ένα σχεδιάσιμο που ενθέτει ένα άλλο σχεδιάσιμο από μια συγκεκριμένη απόσταση. Αυτό είναι ιδιαίτερα χρήσιμο όταν μία όψη (View) χρειάζεται ένα σχεδιάσιμο φόντου που είναι μικρότερο από τα πραγματικά όρια της όψης αυτής.

Clip Drawable

Ένα αρχείο XML που καθορίζει ένα σχεδιάσιμο που αποκόπτει ένα άλλο σχεδιάσιμο βάσει της τρέχουσας τιμής επιπέδου (current level value). Δημιουργεί ένα ClipDrawable.

Scale Drawable

Ένα αρχείο XML που καθορίζει ένα σχεδιάσιμο που αλλάζει το μέγεθος ενός άλλου σχεδιάσιμου βάσει της τρέχουσας τιμής επιπέδου (current level value). Δημιουργεί ένα ScaleDrawable .

Shape Drawable

Ένα αρχείο XML που καθορίζει ένα γεωμετρικό σχήμα, συμπεριλαμβανομένων των χρωμάτων και της διαβάθμισής τους. Δημιουργεί ένα ShapeDrawable.

Τέλος υπάρχει η κατηγορία AnimationDrawable για τη δημιουργία κινούμενων απεικονίσεων όπως καρτούν αλλά δεν θα επεκταθούμε στο συγκεκριμένο θέμα εδώ.

5.3 Πόροι διάταξης και αλληλεπίδραση με γραφικούς πόρους και στοιχεία

Έχουμε δει, όταν ξεκινήσαμε να φτιάξουμε τη διεπαφή χρήστη, τη μορφή των πόρων διάταξης. Αξίζει εδώ να εξετάσουμε πώς οι πόροι διάταξης αλληλοεπιδρούν με τους γραφικούς πόρους για να δομήσουν τη διεπαφή χρήστη μας.

Οι γραφικοί πόροι παρουσιάζουν μεγάλη ποικιλία, αρκετοί όμως από αυτούς μπορούν και να χρησιμοποιηθούν με πολλαπλούς τρόπους. Χωρίς αυτούς η εικόνα που θα έδινε ένας πόρος διάταξης θα ήταν κάπως γυμνή και πάντως αδιάφορη. Ας δούμε πως συνδυάζονται οι δύο κατηγορίες πόρων.

Δραστηριότητα 5.3

Σκοπός της δραστηριότητας αυτής είναι η αξιοποίηση του σχεδιάσιμου (παραλληλόγραμμου) που σχεδιάσατε στην προηγούμενη ενότητα στους πόρους διάταξης που ήδη έχετε σχεδιάσει για τη διεπαφή χρήστη.

Ανοίξτε τον κύριο πόρο διάταξης της εφαρμογής σας και ορίστε το παραλληλόγραμμο σχεδιάσιμο ως φόντο για το κουμπί που έχετε σχεδιάσει. Αυτό γίνεται προσθέτοντας στο στοιχείο Button το ακόλουθο χαρακτηριστικό (χρησιμοποιούμε το όνομα που δώσαμε στο σχεδιάσιμό μας):

```
android:background="@drawable/my_shape"
```

Αν αποθηκεύσετε και γυρίσετε στην ετικέτα (tab) Graphical Layout θα δείτε το αποτέλεσμα. Για να γίνει πιο φυσικό το κουμπί μπορείτε να προσθέσετε ένα πλαίσιο για όγκο. Αυτό γίνεται προσθέτοντας το χαρακτηριστικό εύρος πλαισίου (padding) στο στοιχείο Button, ως εξής:

```
android:padding="5dp"
```

Μπορείτε ακόμη να γεμίσετε την κύρια όψη χρησιμοποιώντας και πάλι το σχεδιάσιμο παραλληλόγραμμο. Ρυθμίστε το ImageView που θα χρησιμοποιήσετε ώστε να γεμίζει όλο το διαθέσιμο χώρο στην όψη σας εκτός από ένα πλαίσιο. Για να μη διαμαρτυρηθεί το Eclipse για την έλλειψη ενός χαρακτηριστικού περιγραφής επισκεφτείτε πρώτα το φάκελο strings στον υποκατάλογο res/values ώστε να ορίσουμε μια συμβολοσειρά για την περιγραφή:

```
<string name="pic">Picture</string>
```

Τώρα, επιστρέφοντας στην κύρια όψη, μπορείτε να ορίσετε το ImageView, αμέσως μετά το Button. Προσέξτε πώς, στην τελευταία γραμμή, αξιοποιείται το string που ορίσατε:

```
<ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_margin="10dp"
        android:src="@drawable/my_shape"
        android:contentDescription="@string/pic" />
```

Επιστρέψτε στην ετικέτα (tab) Graphical Layout για να δείτε το αποτέλεσμα.

Κατά την εργασία με τις πηγές υπάρχει πολύ πήγαινε έλα ανάμεσα στις όψεις που επιτρέπουν εργασία στον κώδικα XML και εκτίμηση του αποτελέσματος μέσω του Graphical Layout. Όμως και κατά τη γραφή κώδικα πηγαινοερχόμαστε ανάμεσα σε διάφορα XML αρχεία, χρησιμοποιώντας τυποποιημένα συντακτικά υποδείγματα για να κάνουμε αναφορές από το ένα στοιχείο πόρων στο άλλο. Αυτό είναι ένα χαρακτηριστικό που θα διατρέχει τη δουλειά πάνω στο Android και στη συνέχεια.

5.4 Άλλοι πόροι

Χρησιμοποιήσαμε στις προηγούμενες ενότητες πόρους διάταξης, σχεδιάσιμους πόρους κι πόρους τιμών. Από τους τελευταίους στην πράξη χρησιμοποιήσαμε μόνο συμβολοσειρές, ενώ από τους σχεδιάσιμους κυρίως κουμπιά και σχήματα. Υπάρχουν πολλές ακόμα επιλογές πόρων. Αναφέραμε σε προηγούμενη παράγραφο τους διάφορους τύπους σχεδιάσιμων πόρων. Ακόμα, αν ανοίξουμε τον κατάλογο res/values στον Package Explorer θα δούμε ότι εκτός από τον υποκατάλογο strings υπάρχουν οι υποκατάλογοι dimens και styles . Στον πρώτο καθορίζουμε διαστάσεις που θέλουμε να χρησιμοποιήσουμε στις εφαρμογές μας ενώ στον δεύτερο καθορίζουμε συγκροτημένες ιδιότητες εμφάνισης για τις διεπαφές χρήστη μας.

Υπάρχει όμως μια μεγάλη σειρά πόρων διαφορετικών τύπων που επίσης μπορούμε να χρησιμοποιήσουμε στις εφαρμογές μας. Άλλοι τύποι πόρων που μπορείτε να βρείτε χρήσιμους περιλαμβάνουν εκείνους για τους αριθμούς, τα μενού, κινούμενα σχέδια, και τις χρωματικές αξίες. Το Eclipse συνήθως δημιουργεί ένα φάκελο μενού όταν δημιουργείτε μια εφαρμογή. Για να ορίσετε κινούμενα σχέδια σε XML, μπορείτε να προσθέσετε έναν υποκατάλογο «anim» ή «animator» στον κατάλογο res, ή να προσθέσετε αρχεία κινουμένων σχεδίων σας στους υποκαταλόγους σχεδιάσιμων. Ενδεικτικά θα αναφέρουμε τις εξής κατηγορίες:

Animation Resources

Ορίζουν προκαθορισμένα κινούμενα σχέδια.

Κινούμενα σχέδια μετασχηματισμών (τύπου tween) αποθηκεύονται στον υποκατάλογο res/anim/ και μπορείτε να τα προσπελάσετε μέσω της κλάσης R.anim.

Κινούμενα σχέδια με frames αποθηκεύονται στον υποκατάλογο res/drawable/ και μπορείτε να τα προσπελάσετε μέσω της κλάσης R.drawable.

Color State List Resource

Ορίζει έναν χρωματικό πόρο που αλλάζει βάσει της κατάστασης της όψης (View).

Αποθηκεύονται στον υποκατάλογο res/color/ / και μπορείτε να τα προσπελάσετε μέσω της κλάσης R.color.

Menu Resource

Ορίζει τα περιεχόμενα των μενού της εφαρμογής σας.

Αποθηκεύεται στον υποκατάλογο res/menu/ και μπορείτε να την προσπελάσετε μέσω της κλάσης R.menu.

Style Resource

Ορίζει την εμφάνιση και διαμόρφωση των στοιχείων διεπαφής χρήστη (UI)

Αποθηκεύεται στον υποκατάλογο res/values/ και μπορείτε να την προσπελάσετε μέσω της κλάσης R.style.

More Resource Types

Ορίζει τύπους τιμών όπως δυαδικούς, ακεραίους, διαστάσεις, χρώματα και πίνακες.

Αποθηκεύονται στον υποκατάλογο res/values/ αλλά προσπελάζονται από διαφορετικές R υποκλάσεις (όπως R.bool, R.integer, R.dimen, κλπ.).

Επιπλέον, αναφέραμε ήδη ότι μπορείτε να δημιουργήσετε υποκαταλόγους εναλλακτικών τύπων πόρων χρησιμοποιώντας προσδιορισμούς για συγκεκριμένες ιδιότητες της συσκευής. Επίσης το Eclipse δημιουργεί υποκαταλόγους τιμών που απευθύνονται σε συγκεκριμένα επίπεδα API, αλλά μπορείτε να επιλέξετε συσκευές χρησιμοποιώντας άλλους προσδιορισμούς. Για παράδειγμα, μπορεί να θέλετε να χρησιμοποιήσετε σταθερό πλάτος και ύψος για την ImageView που προσθέσαμε στην προηγούμενη ενότητα, προσαρμόζοντας το μέγεθος που εμφανίζεται στο μέγεθος οθόνης της συσκευής.

Για να επιτευχθεί αυτό, θα μπορούσατε να προσθέσετε υποκαταλόγους τιμών για κάθε μέγεθος ή πυκνότητα ανάλυσης ("-small", "-large", "-hdpi", "-mdpi", κλπ), με ένα αρχείο διαστάσεων για κάθε μία. Συμπεριλαμβάνοντας μια τιμή διάστασης σε κάθε ένα από αυτά τα αρχεία, δίνοντας στην τιμή το ίδιο όνομα αλλά διαφορετικό αριθμό, το Android επιλέγει αυτόματα το σωστό για τη συσκευή του χρήστη.

Κεφάλαιο 6

Δηλωτικό Έργου

6. Δηλωτικό Έργου

Το δηλωτικό έργου εξυπηρετεί πολλαπλούς σκοπούς: από τον προσδιορισμό της συσκευασίας της εφαρμογής, την περιγραφή με τυπικό τρόπο των τμημάτων της εφαρμογής, δηλώνει τα δικαιώματα χρήσης, τα απαραίτητα επίπεδα API και τις συνδεδεμένες βιβλιοθήκες. Υπάρχουν απαραίτητα και προαιρετικά τμήματα του που περιλαμβάνουν μια ποικιλία στοιχείων και χαρακτηριστικών.

Στόχοι

Οι μαθητές να μπορούν να:

- Αναγνωρίζουν την ανάγκη επιλογών χειρισμού και την σύνδεσή τους με τις προδιαγραφές της εφαρμογής τους
- Αναγνωρίζουν την αναγκαιότητα ενσωμάτωσης των επιλογών αυτών σε μια αυτόνομη εφαρμογή και το ρόλο του δηλωτικού στη διαδικασία αυτή στην περίπτωση Android εφαρμογής
- Περιγράφουν την διαχείριση της πολυμορφικότητας σε σχέση με την εκάστοτε χρήση
- Αναγνωρίζουν συνήθη στοιχεία ενός δηλωτικού
- Προσαρμόζουν έτοιμες εγγραφές στις ανάγκες των έργων τους

Ενότητες Κεφαλαίου

- Το στοιχείο του Δηλωτικού
- Το στοιχείο Uses-SDK
- Το στοιχείο Εφαρμογής (Application)
- Στοιχεία Δραστηριοτήτων (για τις κλάσεις δραστηριοτήτων)
- Στοιχεία Intent Filters
- Προσβάσεις (Uses Permission)
- Συσκευές χρήστη
- Άλλες συσκευές

6.1 Το στοιχείο του Δηλωτικού

Στον ριζικό κατάλογο του έργου μας υπάρχει το αρχείο δηλωτικού (manifest). Μπορούμε να το δούμε με διάφορους τρόπους αλλά να θέλουμε να το δούμε στην αυθεντική (XML) μορφή του πρέπει να επιλέξουμε την αντίστοιχη ετικέτα, που στο Eclipse ονομάζεται «AndroidManifest.xml».

Στο αρχείο δηλωτικού εμφανίζονται εγγραφές για διάφορα στοιχεία του έργου. Η πρώτη εγγραφή μετά τη δήλωση της έκδοσης της γλώσσας είναι αυτή που αφορά στο ίδιο το δηλωτικό και αποτελεί τη ριζική εγγραφή μιας δενδροειδούς μορφής. Τα επίπεδα των διακλαδώσεων είναι εμφανή γιατί τα παιδιά ενός στοιχείου ακολουθούν την ετικέτα ανοίγματος του στοιχείου, προηγούνται της ετικέτας κλεισίματός του και γράφονται πιο «μέσα» από το γονέα – στο ίδιο σημείο που ξεκινούν και τα χαρακτηριστικά του.

Τα βασικά χαρακτηριστικά που έχουν αυτόματα συμπληρωθεί είναι το όνομα του πακέτου, η εσωτερική (versionCode) και η εξωτερική (versionName) ονομασία της έκδοσης. Αν υποθέσουμε ότι έχουμε ονομάσει, όταν το ξεκινήσαμε, το έργο μας myfirstapp η σχετική εγγραφή θα πρέπει να μοιάζει κάπως έτσι, αν παραλείψουμε τα στοιχεία παιδιά του πριν από το κλείσιμο:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myfirstapp"
    android:versionCode="1"
    android:versionName="1.0" >
</manifest>
```

Το πακέτο (package) προκύπτει από τα όσα δηλώσατε κατά τη δημιουργία του έργου και είναι ουσιαστικά το πλήρες δέντρο καταλόγων που ορίστηκε για την εφαρμογή μας με όλα τα περιεχόμενά του. Η εξωτερική ονομασία του έργου είναι αυτή που βλέπει ο χρήστης όταν για παράδειγμα βρίσκει την εφαρμογή στο Play Store και μπορεί να ακολουθήσει όποια σύμβαση αποφασίσετε για να διακρίνονται οι εκδόσεις.

Η εσωτερική ονομασία αφορά στους προγραμματιστές αλλά είναι επίσης αυτή που χρησιμοποιεί το ίδιο το Play Store για να διακρίνει της εκδόσεις. Δεν ακολουθεί κάποια συγκεκριμένη σύμβαση όμως ο αριθμός που αποτελεί την εσωτερική ονομασία μιας επόμενης έκδοσης πρέπει να είναι μεγαλύτερος από κάθε προηγούμενης – ιδιαίτερα αν θέλετε να την αναρτήσετε στο Play Store.

6.2 Το στοιχείο Uses-SDK

Το επόμενο στοιχείο στο αρχείο του δηλωτικού είναι η δήλωση χρήσης της έκδοσης SDK (uses-sdk). Τα χαρακτηριστικά του προσδιορίζουν το ελάχιστο επίπεδο API αλλά και το επίπεδο στόχο για τα οποία σκοπεύετε να ελεγχθεί η λειτουργικότητα της εφαρμογής. Το πρώτο προσδιορίζει το ελάχιστο API στο οποίο παίζει η εφαρμογή ενώ το δεύτερο το API στο οποίο έχει πλήρη λειτουργικότητα. Και αυτά είναι τα ίδια που εσείς ορίσατε κατά τη δημιουργία του έργου. Η αντίστοιχη εγγραφή θα δείχνει είναι κάπως έτσι:

```
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="18" />
```

Είναι δυνατό να αλλαχθούν οι τιμές αυτές σε οποιοδήποτε χρονικό σημείο. Μόλις όμως αλλάξουν οι εκδόσεις του SDK πρέπει να ξαναδομηθεί (rebuild) το έργο. Το Eclipse το κάνει αυτό άμεσα. Και υπάρχει λόγος για να γίνει κάτι τέτοιο: το ελάχιστο επίπεδο API προσδιορίζει ποιοι

χρήστες θα μπορούν να χρησιμοποιήσουν την εφαρμογή σας - χρήστες που η συσκευή τους χρησιμοποιεί χαμηλότερη έκδοση δεν θα έχουν πρόσβαση σε αυτή.

Κανονικά η εφαρμογή πρέπει να δοκιμαστεί για όλα τα API μεταξύ ελάχιστου και στόχου ή πάντως για όσα περισσότερα είναι αυτό δυνατό. Για μια εφαρμογή, καλό είναι να έχει μεγάλο φάσμα εκδόσεων Android στις οποίες να λειτουργεί ειδικά αν πρόκειται για πωλούμενη έκδοση.

Η εγγραφή κλείνει μόνη της με το «/» πριν το «>» επειδή το στοιχείο αυτό δεν έχει παιδιά για να χρειάζεται ξεχωριστή ετικέτα για το κλείσιμο.

6.3 Το στοιχείο Εφαρμογής (Application)

Έχουμε και αλλού αναφερθεί στο πώς κατά τη γραφή κώδικα πηγαινοερχόμαστε ανάμεσα σε διάφορα XML αρχεία, χρησιμοποιώντας τυποποιημένα συντακτικά υποδείγματα για να κάνουμε αναφορές από το ένα στοιχείο πόρων στο άλλο. Το ίδιο μπορεί να συμβαίνει και μέσα στο ίδιο αρχείο με αναφορές δενδροειδούς χαρακτήρα σε άλλα στοιχεία.

Το επόμενο στοιχείο στο αρχείο δηλωτικού, το στοιχείο εφαρμογής, είναι ένα καλό παράδειγμα τέτοιας περίπτωσης παρότι έχει εισαχθεί από το περιβάλλον, στην περίπτωσή μας το Eclipse, και όχι από προγραμματιστή. Έτσι η εγγραφή για το στοιχείο Εφαρμογής θα ανοίγει με τα χαρακτηριστικά του αλλά θα περιλαμβάνει επίσης εγγραφές για τα στοιχεία παιδιά του. Μια ετικέτα ανοίγματος για το στοιχείο αυτό θα έχει τέτοια μορφή:

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```

Προσέξτε τις αναφορές τύπου «@<υποκατάλογος ή αρχείο πόρων>» στα χαρακτηριστικά του στοιχείου. Αυτές μας ενημερώνουν για τη θέση της αντίστοιχης τιμής:

Το icon έχει οριστεί η τυποποιημένη εικόνα launcher του android που υπάρχει ως αρχείο ic_launcher στα σχεδιάσιμα, από προεπιλογή.

Το label μας παραπέμπει σε ένα στοιχείο τύπου τιμής string που έχει αποθηκευτεί με το όνομα app_name στο αρχείο strings.xml (υποκατάλογος res/values) και περιέχει την αρχική μας δήλωση για το όνομα του έργου διατηρώντας τα κεφαλαία – για λόγους αναγνωσιμότητας ίσως πρέπει να αντικατασταθεί.

Το theme μας παραπέμπει σε ένα στοιχείο τύπου τιμής style που έχει αποθηκευτεί με το όνομα AppTheme στο αντίστοιχο αρχείο (styles.xml στον υποκατάλογο res/values). Πρόκειτε για προεπιλεγμένο στοιχείο για άλλη μια φορά.

6.4 Στοιχεία Δραστηριοτήτων (για τις κλάσεις δραστηριοτήτων)

Μέσα στο στοιχείο Εφαρμογής υπάρχουν τα στοιχεία Δραστηριοτήτων. Ήδη στην εφαρμογή σας πρέπει να υπάρχει μια εγγραφή για ένα τέτοιο στοιχείο που αντιπροσωπεύει την κύρια κλάση Δραστηριότητας (Activity) που δημιουργήσατε ξεκινώντας το έργο.

Το στοιχείο αυτό είναι παιδί του στοιχείου Εφαρμογής αλλά έχει και τα δικά του παιδιά όπως θα δούμε αμέσως μετά. Η ετικέτα ανοίγματος αυτού του στοιχείου θα έχει την μορφή:

```
<activity
    android:name="com.example.myfirstapp.MainActivity"
```

```
android:label="@string/app_name" >
```

Το χαρακτηριστικό name περιέχει και πάλι ένα πλήρες μονοπάτι στο πακέτο της εφαρμογής. Το χαρακτηριστικό label περιέχει τον τίτλο του παραθύρου όταν η δραστηριότητα είναι στην οθόνη. Από προεπιλογή είναι το ίδιο που συναντήσαμε και στο στοιχείο application.

Όταν έρθει η ώρα να προσθέσετε νέες δραστηριότητες μπορείτε να τις προσθέσετε μέσα στο στοιχείο Εφαρμογής και μετά την εγγραφή για την κύρια δραστηριότητα. Η εγγραφή για το νέο στοιχείο θα έχει τη μορφή:

```
<activity
    android:name=" com.example.myfirstapp.NewActivity" >
    // εδώ μπαίνει ο κώδικας για τη νέα δραστηριότητα
</activity>
```

Δραστηριότητα 6.4

Αναπτύξτε πλήρως μια νέα δραστηριότητα για την εφαρμογή σας. Εκμεταλευτείτε, όπου μπορείτε, τις δυνατότητες επαναχρησιμοποίησης κώδικα, σχεδιάσιμων και τιμών.

6.5 Στοιχεία Intent Filters

Πριν αναφερθούμε στα φίλτρα προθέσεων (Intent Filters) χρειάζεται να αναφερθούμε σύντομα στις ίδιες τις Προθέσεις (Intents). Μια Πρόθεση είναι ένα αντικείμενο ανταλλαγής μηνυμάτων που μπορεί να χρησιμοποιηθεί για να ζητήσει μία συνιστώσα (βασικό δομικό στοιχείο - component) μιας εφαρμογής την εκτέλεση μιας ενέργειας από μία άλλη. Οι συνιστώσες αυτές ανήκουν σε τέσσερις τύπους: τις Δραστηριότητες (Activities), τις Υπηρεσίες (Services), τους Προμηθευτές Περιεχομένου (Service Providers) και τους Δέκτες Εκπομπών (Broadcast Receivers).

Χωρίς να επεκταθούμε στις επιμέρους λειτουργίες κάθε κατηγορίας αρκεί να πούμε ότι οι συνιστώσες μιας εφαρμογής υφίστανται ανεξάρτητα η μία από την άλλη και η επικοινωνία τους στηρίζεται σχεδόν αποκλειστικά σε ανταλλαγές μηνυμάτων οι οποίες υλοποιούνται από αντικείμενα που ονομάζονται Προθέσεις (Intents). Ένα τέτοιο αντικείμενο μεταφέρει πληροφορία την οποία αξιοποιεί το Android για να εντοπίσει ποια συνιστώσα να εκκινήσει αλλά και πληροφορία που η συνιστώσα, η οποία θα επιλεγεί, θα χρησιμοποιήσει για να εκτελέσει την ενέργεια (όπως data specifications). Το αντικείμενο μεταφέρεται στην επιλεγμένη συνιστώσα με την κλήση της.

Άσχετα από τη δομή ενός τέτοιου μηνύματος είναι φανερό ότι μια συνιστώσα – στην περίπτωσή μας μια Δραστηριότητα (Activity) – δεν μπορεί να εκτελέσει τις σχετικές οδηγίες αν δεν μπορεί να τις εντοπίσει. Το αντικείμενο μεταφέρεται στην επιλεγμένη Δραστηριότητα αλλά εκείνη πρέπει να έχει ένα πρωτόκολλο χειρισμού του. Πρακτικά χρειάζεται ένα τέτοιο πρωτόκολλο, ή σύνολο οδηγιών, για κάθε πιθανή Πρόθεση που θα κληθεί να υλοποιήσει.

Ένα φίλτρο Προθέσεων περιγράφει τις Προθέσεις στις οποίες η μητρική του Δραστηριότητα μπορεί να ανταποκριθεί. Είτε λάβει μια Πρόθεση για να εκκινήσει τη λειτουργία της είτε για να εκτελέσει κάποια ενέργεια το φίλτρο θα κρίνει αν η Δραστηριότητα μπορεί να εκτελέσει και πώς.

Στην εφαρμογή σας, στην κύρια Δραστηριότητα υπάρχει ήδη ένα φίλτρο Προθέσεων. Αυτό έχει την ακόλουθη μορφή:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
```

Το συγκεκριμένο φίλτρο αφορά στην κύρια δραστηριότητα και προσδιορίζει ότι αυτή θα πρέπει να εκκινήσει όταν εκκινήσει η εφαρμογή. Για το σκοπό αυτό χρησιμοποιεί το στοιχείο action στο οποίο καταλογογραφείται η ενέργεια MAIN. Το στοιχείο category περιγράφει την κατηγορία του συγκεκριμένου φίλτρου (LAUNCHER). Χρησιμοποιώντας τους καταλόγους των δύο αυτών τυποποιημένων στοιχείων από κοινού προσδιορίζει τη μητρική του Δραστηριότητα ως το σημείο εισόδου για την εφαρμογή και κάνει σαφές ότι η κύρια Δραστηριότητα χρησιμοποιείται όποτε εκκινεί η εφαρμογή.

6.6 Προσβάσεις (Uses Permission)

Τα στοιχεία Προσβάσεως δεν είναι υποχρεωτικά για ένα δηλωτικό. Για αυτό το λόγο και το Eclipse δεν περιλαμβάνει τέτοια στοιχεία αυτόματα όταν δημιουργείται ένα έργο. Όμως τα στοιχεία Προσβάσεως, επιτρέπουν να ζητήσει ο προγραμματιστής την αποδοχή από το χρήστη της εκτέλεσης από την εφαρμογή ενεργειών που απαιτούν, με τον ένα ή τον άλλο τρόπο, την έγκρισή του.

Οι ενέργειες που απαιτούν την έγκριση του χρήστη περιλαμβάνουν την χρήση δεδομένων από το διαδίκτυο, την εγγραφή σε εξωτερικές μονάδες μνήμης, την πρόσβαση σε τεχνικά χαρακτηριστικά της συσκευής όπως η κάμερα ή το μικρόφωνο. Μια Πρόσβαση έχει τη μορφή αναφοράς σε μία καταλογογραφημένη ενέργεια του τυποποιημένου στοιχείου permission και έχει την εξής δομή:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Αν η εφαρμογή έχει αναρτηθεί στο Play Store οι προσβάσεις θα φαίνονται στην ανάρτηση ώστε οι χρήστες να τις γνωρίζουν πριν «κατεβάσουν» την εφαρμογή. Για να προχωρήσουν πρέπει να εγκρίνουν τις προσβάσεις που ζητάτε ώστε η εφαρμογή σας να μπορεί να τρέξει στη συσκευή τους.

6.7 Συσκευές χρήστη

Υπάρχουν επιπλέον κάποια μη υποχρεωτικά στοιχεία του δηλωτικού που επιτρέπουν να περιγραφούν στοιχεία και τεχνικά χαρακτηριστικά υλικού αλλά και λογισμικού που είναι απαραίτητα για τη λειτουργία μιας εφαρμογής. Και πάλι χρησιμοποιούν αναφορές σε κάποια τυποποιημένα στοιχεία τα οποία όμως μπορούν να επεκταθούν κατάλληλα.

Ενδεικτικά, μέσω του στοιχείου uses-configuration μπορούν να προσδιοριστούν απαιτήσεις για την πλοήγηση και επιλογές πληκτρολογίου και οθόνης αφής. Στο στοιχείο uses-feature μπορεί να καταλογογραφηθεί ένα απαιτούμενο τεχνικό χαρακτηριστικό είτε στο υλικό είτε στο λογισμικό με απλή αναφορά του ονόματος και μια σημαία δυαδικού (Boolean) τύπου. Αυτό μπορεί να αφορά στη χρήση επιλογών του Bluetooth ή της κάμερας, των αισθητήρων ή του εντοπισμού θέσης. Τέλος με το στοιχείο supports-screens προσδιορίζονται μεγέθη οθόνης που υποστηρίζονται από την εφαρμογή, προσδιορίζοντας χαρακτηριστικά που συνδέονται είτε με το μέγεθος είτε με την πυκνότητα της απεικόνισης.

Δραστηριότητα 6.7

Αναζητείστε τον τρόπο που συντάσεται το στοιχείο supports-screens και προσδιορίστε ως υποστηριζόμενο το μέγεθος οθόνης της συσκευής που χρησιμοποιείτε για τις δοκιμές σας.

6.8 Άλλες συσκευές

Αναφέραμε σε προηγούμενη παράγραφο τους τέσσερις τύπους συνιστωσών μιας εφαρμογής. Παρότι δεν επεκταθήκαμε στη χρήση τους είναι φανερό ότι μέχρι στιγμής έχουμε περιγράψει ένα δηλωτικό που περιλαμβάνει μόνο τον ένα από τους τύπους, συγκεκριμένα τις Δραστηριότητες.

Όταν όμως έχουμε ολοκληρωμένες εφαρμογές αυτές συχνά απαιτούν τη λειτουργία διαδικασιών στο βάθος (background), τη διαχείριση δεδομένων και της πρόσβασης σε αυτά και την αποδοχή μηνυμάτων/ ανακοινώσεων του συστήματος. Για την κάθε μία από αυτές τις λειτουργίες χρησιμοποιείτε ένας από τους άλλους τρεις τύπους:

Για να επιτρέψουμε τη λειτουργία διαδικασιών στο παρασκήνιο χρησιμοποιούμε τις Υπηρεσίες (Services). Για να το κάνουμε αυτό , απαιτείται η εγγραφή του στοιχείου service στο δηλωτικό και μάλιστα μια φορά για κάθε κλάση Service – ακριβώς όπως και με τις Δραστηριότητες.

Για τη διαχείριση της πρόσβασης σε πηγές δεδομένων στο Android χρησιμοποιούνται οι Προμηθευτές Περιεχομένου (Service Providers) που εγγράφονται χρησιμοποιώντας το στοιχείο provider.

Για να μπορεί μια εφαρμογή να δέχεται αιτήματα Προθέσεων (Intents)από άλλες εφαρμογές ή και το σύστημα χρησιμοποιούμε τους Δέκτες Εκπομπών (Broadcast Receivers) εγγράφοντας το στοιχείο receiver.

Οι λεπτομέρειες δεν είναι ιδιαίτερα σημαντικές σε αυτό το σημείο αλλά η αναφορά των τύπων είναι απαραίτητη για την πληρέστερη κατανόηση ενός δηλωτικού.

Κεφάλαιο 7

Δεδομένα εφαρμογής

7. Δεδομένα εφαρμογής

Τα δεδομένα μιας εφαρμογής αποτελούν το πλέον ευαίσθητο στοιχείο της. Η διαχείρισή τους παίζει σημαντικό ρόλο στην δυνατότητα να εξασφαλιστεί η ασφάλεια και η ακεραιότητά τους.

Στόχοι

Οι μαθητές να μπορούν να:

- Περιγράφουν τον τρόπο διαχείρισης των συνήθων τύπων δεδομένων
- Διακρίνουν τη διαφορά μεταξύ Private Internal files, Public External Files
- Σχεδιάζουν τη διαχείριση βάσης δεδομένων μέσω εφαρμογής
- Χρησιμοποιούν τους τύπους δεδομένων για να εξασφαλίζουν την ασφάλειά τους.

Ενότητες Κεφαλαίου

- Τύποι δεδομένων (shared preferences, Private Internal files, Public External Files)
- Ασφάλεια δεδομένων
- Βάσεις δεδομένων
- Διαδικτυακά δεδομένα

7.1 Τύποι δεδομένων (shared preferences, Private Internal files, Public External Files)

Υπάρχουν τρεις διαφορετικοί τρόποι να χειριστεί κανείς δεδομένα μέσα από την εφαρμογή χωρίς να εμπλακεί με βάσεις δεδομένων ή το διαδίκτυο. Μπορεί να χρησιμοποιήσει την κλάση Shared Preferences για να αποθηκεύσει δεδομένα των βασικών (primitive) τύπων , μπορεί να χρησιμοποιήσει εσωτερική αποθήκευση στη μνήμη της συσκευής και να χρησιμοποιήσει την εξωτερική αποθήκευση.

7.1.1 Χρησιμοποιώντας shared preferences

Η κλάση SharedPreferences είναι η πιο απλή μορφή αποθήκευσης αλλά και η πιο περιορισμένη. Επιτρέπει την διαχείριση τιμών οποιουδήποτε βασικού τύπου (boolean, float, int, long, ή string) δημιουργώντας ζευγάρια ονομάτων και τιμών. Η αντιστοίχιση θα παραμείνει σε όλες τις επόμενες περιπτώσεις ενεργοποίησης της εφαρμογής.

Συχνή χρήση της SharedPreferences είναι η αποθήκευση ρυθμίσεων του χρήστη για την εφαρμογή και από εκεί προέρχεται και το όνομα προτιμήσεις (preferences). Είναι ιδιαίτερα χρήσιμες σε σχέση με επιλογές εμφάνισης και συμπεριφοράς της εφαρμογής σας. Ας δούμε πως γίνεται αυτό στην πράξη:

Δραστηριότητα 7.1.1

Σκοπός της δραστηριότητας αυτής να εισάγουμε κάποιες προτιμήσεις στην εφαρμογή σας. Ορίστε τη δημόσια μεταβλητή MY_APP_PREFS και χρησιμοποιείστε την για να ανακτήσετε προτιμήσεις σας. Περιλάβετε τουλάχιστον την αναγραφή Pressed στο κουμπί thebutton. Θα χρειαστεί να εισάγετε την κλάση android.content. Shared Preferences.

7.1.2 Χρησιμοποιώντας internal files

Όταν σώνετε αρχεία στην εσωτερική μνήμη της συσκευής αυτά είναι από προεπιλογή ιδιωτικά για την εφαρμογή σας. Η συσκευή τα θεωρεί τμήμα της εφαρμογής, δεν προσφέρει απευθείας πρόσβαση σε αυτά έξω από την εφαρμογή, ενώ τα απομακρύνει αν απομακρυνθεί η εφαρμογή.

Η δημιουργία ενός φακέλου στην εσωτερική μνήμη (Internal storage) γίνεται χρησιμοποιώντας την κλάση FileOutputStream την οποία εισάγουμε με τον γνωστό τρόπο από τη θέση java.io.FileOutputStream. Για τη δημιουργία απαιτείται ένα όνομα και ένας τρόπος (mode) πρόσβασης. Εφόσον ο τρόπος οριστεί private ο φάκελος μπορεί να χρησιμοποιηθεί μόνο από την εφαρμογή:

FileOutputStream fileOut = openFileOutput(“my_file”, Context.MODE_PRIVATE);

Στην πράξη είναι καλό να ελέγχουμε πρώτα αν η διαδικασία εισόδου/ εξόδου είναι εφικτή για να μην κολλήσει η εφαρμογή. Για το λόγο αυτό χρησιμοποιούμε ένα μπλοκ δοκιμής (try). Αν για κάποιο λόγο προκληθεί ένα exception τότε εκτελείται ο κώδικας στο Catch μπλοκ καταγράφοντας απλά το συμβάν στο Log. Για να εισάγουμε την κλάση Log με το γνωστό τρόπο την εισάγουμε από τη θέση android.utility.Log. Το μήνυμα λάθους, αν προκύψει, θα φανεί επομένως στο Android LogCat.

Δραστηριότητα 7.1.2

Σκοπός της δραστηριότητας αυτής είναι η δημιουργία και επεξεργασία ενός internal file για την εφαρμογή σας. Χρησιμοποιείστε ένα Try block στο σημείο που θα δημιουργείτε το αρχείο. Ο σχετικός κώδικάς θα πρέπει να είναι κάπως έτσι:

```
try{
        FileOutputStream fileOut = openFileOutput("my_file",  Context.MODE_PRIVATE);
}
catch(IOException ioe){
        Log.e("APP_TAG", "IO Exception", ioe);
}
```

Υλοποιείστε την εγγραφή και ανάκτηση δεδομένων από το αρχείο. Μπορείτε να συνδέσετε τις ενέργειες αυτές με ένα στοιχείο, όπως ας πούμε ένα text box ή ένα κουμπί, μέσα στην κύρια δραστηριότητα.

7.1.3 Χρησιμοποιώντας external files

Με την προϋπόθεση ότι η συσκευή σας διαθέτει εξωτερική μνήμη (external storage) μπορείτε και εκεί να αποθηκεύσετε αρχεία. Σε αυτή την περίπτωση τα αρχεία σας είναι από προεπιλογή δημόσια, αυτό δεν μπορεί να αλλάξει και κατά συνέπεια άλλες εφαρμογές έχουν δυνατότητα να τα προσπελάσουν. Μια εξωτερική μνήμη μπορεί να είναι μια κάρτα SD, κάποιο αφαιρούμενο μέσο αποθήκευσης ή απλά εσωτερική μνήμη χαρακτηρισμένη ως εξωτερική από το σύστημα.

Αντίθετα από την εσωτερική μνήμη, εξωτερική δεν υπάρχει πάντα ενώ σε κάποιες περιπτώσεις μπορεί να υπάρχει εξωτερική μνήμη αλλά να είναι μόνο για ανάγνωση οπότε δεν μπορείτε να δημιουργήσετε αρχεία εκεί. Πριν από τις ενέργειες πρέπει να διερευνήσουμε τι ισχύει. Αυτό γίνεται με την:

```
String extStorageState = Environment.getExternalStorageState();
```

Η συμβολοσειρά extStorageState μπορεί να συγκριθεί τώρα με πεδία της κλάσης Environment που αντιπροσωπεύουν τις καταστάσεις της εξωτερικής μνήμης

Αν διαθέτουμε εγγράψιμη εξωτερική μνήμη τότε θα είναι αληθής η τιμή που προκύπτει από τον έλεγχο Environment.MEDIA_MOUNTED.equals(extStorageState). Αν διαθέτουμε μη-εγγράψιμη (Read-only) εξωτερική μνήμη τότε θα είναι αληθής η τιμή που προκύπτει από τον αντίστοιχο έλεγχο Environment.MEDIA_MOUNTED_READ_ONLY.equals(extStorageState). Οι έλεγχοι αυτοί μπορούν να χρησιμοποιηθούν σε ένα φωλιασμένο If που αντικαθιστά το Try μπλοκ της προηγούμενης παραγράφου.

Φυσικά χρειάζεται πρώτα να έχουμε τη σχετική πρόσβαση (permission) στο δηλωτικό:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>;
```

Με δεδομένο ότι υπάρχει η σχετική δυνατότητα γραφής, χρειαζόμαστε επίσης να ανακτήσουμε και τη θέση του σχετικού καταλόγου πριν αποθηκεύσουμε. Αυτό γίνεται (για API level >=8) με την getExternalFilesDir οπότε η δημιουργία του φακέλου παίρνει τη μορφή:

```
File myFile = new File(getExternalFilesDir(null), "MyFile.txt";);
```

Η χρησιμότητα των εξωτερικών αρχείων αναδεικνύεται όταν οι εφαρμογές γίνουν περισσότερο προχωρημένες οπότε μπορεί να είναι σκόπιμο να επεξεργάζεται κανείς αρχεία

διαθέσιμα και σε άλλες εφαρμογές. Ένα καλό παράδειγμα είναι όταν αναπτύσσουμε μια σουίτα εργαλείων για κάποιον ειδικό σκοπό.

7.2 Ασφάλεια δεδομένων

Είναι φανερό ότι από τους τρεις τρόπους διαχείρισης δεδομένων της προηγούμενης ενότητας , η χρήση εξωτερικών αρχείων είναι ταυτόχρονα ο πλέον δυναμικός αλλά και ο λιγότερο ασφαλής για τα δεδομένα μας. Η επιλογή τρόπου διαχείρισης παίζει σημαντικό ρόλο στο επίπεδο ασφαλείας.

Η χρήση και, πολύ περισσότερο, η δημιουργία διαδικτυακών δεδομένων παίζει επίσης σημαντικό ρόλο και το ίδιο μπορεί να συμβαίνει με τον τρόπο που διαχειριζόμαστε μια βάση δεδομένων. Για τις δύο αυτές περιπτώσεις δεδομένων της εφαρμογής μας θα αναφερθούμε στις δύο επόμενες ενότητες του παρόντος κεφαλαίου.

Άλλοι παράγοντες που παίζουν επίσης ρόλο είναι οι τρόποι επικοινωνίας της εφαρμογής εν γένει με το σύστημα και άλλες εφαρμογές. Η ανταλλαγή μηνυμάτων και η δυνατότητα χρησιμοποίησης τμημάτων της εφαρμογής σας από άλλες είναι δύο ακόμα παράγοντες που επίσης επηρεάζουν αλλά με τους οποίους θα ασχοληθούμε σε επόμενο χρόνο.

7.3 Βάσεις δεδομένων

Οι βάσεις δεδομένων αποκτούν σημασία ως επιλογή αποθήκευσης δεδομένων σε δύο περιπτώσεις: όταν ο όγκος των δεδομένων γίνεται πολύ μεγάλος ή όταν η δομή των δεδομένων μας γίνεται ιδιαίτερα περίπλοκη. Και στις δύο περιπτώσεις οι επιλογές διαχείρισης της προηγούμενης παραγράφου καθίστανται ασύμφορες.

Ευτυχώς το Android παρέχει λειτουργικότητες τόσο για την δημιουργία όσο και για την διαχείριση βάσεων δεδομένων τύπου SQLite μέσα από τις εφαρμογές μας. Όταν η εφαρμογή μας δημιουργήσει μια βάση δεδομένων αυτή είναι ιδιωτική για την εφαρμογή.

Για το σκοπό αυτό μας δίνει τη δυνατότητα να χρησιμοποιήσουμε μια κλάση η οποία προσφέρει αυτές τις λειτουργικότητες την οποία επεκτείνει (extends) η δική μας κλάση για την βάση. Μέσα σε αυτή την κλάση ορίζονται οι ιδιότητες (properties) της βάσης μέσα από τη δημιουργία μεταβλητών κλάσης στις οποίες ορίζονται τα ονόματα των πινάκων της βάσης και οι συμβολοσειρές δημιουργίας τους στην SQL.

Στο παράδειγμα:

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
            "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" + KEY_WORD + " TEXT, " +
             KEY_DEFINITION + " TEXT);";
    DictionaryOpenHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
```

```
            db.execSQL(DICTIONARY_TABLE_CREATE);
        }
    }
```

κατασκευάζουμε ένα λεξικό που αποτελείται από έναν απλό πίνακα με δύο στήλες, κατάλληλα ορισμένες, για τη λέξη και για τη σημασία της.

Μέσα στην κλάση SQLiteOpenHelper μπορούμε να υπεφορτώσουμε τη μέθοδο onCreate για να δημιουργήσουμε τη βάση μας ενώ όταν χρειαστούμε πρόσβαση σε αυτή μέσα από τον κώδικα (π.χ. μιας Δραστηριότητας) μπορούμε να χρησιμοποιήσουμε τις μεθόδους της SQLiteOpenHelper για να εισάγουμε (getWritableDatabase) και να ανακτήσουμε (getReadableDatabase) εγγραφές. Για περισσότερα δες στη θέση:

http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html .

Για να διασχίσουμε τα αποτελέσματα ενός ερωτήματος (query) και να τα εμφανίσουμε στη διεπαφή χρήστη, χρησιμοποιούμε την κλάση Cursor. Με την κλάση αυτή κατασκευάζουμε αντικείμενα που αναφέρονται διαδοχικά σε κάθε γραμμή αποτελέσματος, μέσω της παραμέτρου factory της SQLiteOpenHelper. Για την κλάση Cursor δες στη θέση:

http://developer.android.com/reference/android/database/Cursor.html .

Δραστηριότητα 7.3

Συζητείστε στην τάξη πιθανές περιπτώσεις για χρήση βάσεων και μετά ενσωματώστε κάποια από αυτές στην εφαρμογή σας, χρησιμοποιώντας απλούς πίνακες.

7.4 Διαδικτυακά δεδομένα

Η χρήση διαδικτυακών δεδομένων είναι ιδιαίτερα διαδεδομένη μεταξύ των εφαρμογών Android που συχνά δεν είναι τίποτε περισσότερο από μια διεπαφή προς κάποια διαδικτυακή πηγή δεδομένων. Για να ολοκληρωθεί η επικοινωνία με το διαδίκτυο απαιτείται αφενός να υπάρχει ενεργή σύνδεση στο Internet στη συσκευή του χρήστη για να επιτραπεί η αποθήκευση και ανάκτηση δεδομένων και αφετέρου η πρόσβαση (permission) χρήσης της με τη σχετική εγγραφή στο δηλωτικό που ενσωματώνει την android.permission.INTERNET.

Για να αποφευχθούν κολλήματα της εφαρμογής καλό είναι οι διαδικασίες αποθήκευσης και ανάκτησης μέσω διαδικτύου να εκτελούνται ξεχωριστά από την κεντρική διεπαφή χρήστη της εφαρμογής. Για το σκοπό αυτό υπάρχει η κλάση AsyncTask την οποία προσθέτουμε σε κατάλληλη δραστηριότητα, και μέσω της οποίας μπορούμε να ανακτήσουμε δεδομένα από το διαδίκτυο στο παρασκήνιο και να εμφανίσουμε τα αποτελέσματα στη διεπαφή όταν αφιχθούν χωρίς να αδρανοποιηθεί η διεπαφή χρήστη στο ενδιάμεσο.

Η κλάση AsyncTask προσφέρει τις μεθόδους doINBackground και onPostExecute για την ανάκτηση δεδομένων και την εμφάνισή τους στη διεπαφή αντίστοιχα. Για τη διαχείριση επιστροφής δομημένων δεδομένων υπάρχουν και άλλες λειτουργίες όπως τα JSON feeds που όμως δε θα μας απασχολήσουν σε αυτή τη φάση. Γενικά οι κλάσεις που εξυπηρετούν την επικοινωνία με μια σύνδεση δικτύου βρίσκονται στα πακέτα java.net και android.net.

Κεφάλαιο 8

Εκτέλεση και εκσφαλμάτωση

8. Εκτέλεση και εκσφαλμάτωση

Η εκτέλεση μιας εφαρμογής αποτελεί τον σκοπό της ανάπτυξής της. Η εκσφαλμάτωση διασφαλίζει τη δυνατότητα εκτέλεσης αλλά και την ποιότητα της εμπειρίας του χρήστη από την εφαρμογή. Η τελική όμως ποιότητα της εμπειρίας αυτής απαιτεί την κατανόηση της ποικιλίας των συσκευών στις οποίες απευθύνεται μια Android εφαρμογή.

Στόχοι

Οι μαθητές να μπορούν να:

- Αναγνωρίζουν τη σπουδαιότητα των εικονικών συσκευών για Unix-type λειτουργικά
- Περιγράφουν τον τρόπο που το Android αντιλαμβάνεται τις συσκευές
- Αναγνωρίζουν τα πλεονεκτήματα της προσομοίωσης στην εκσφαλμάτωση
- Αναγνωρίζουν την αναγκαιότητα δομών ελέγχου
- Κατασκευάζουν και χρησιμοποιούν ένα test project
- Περιγράφουν τον τρόπο που οι συσκευές χρησιμοποιούνται στην εκσφαλμάτωση
- Χρησιμοποιούν εργαλεία ελέγχου
- Εκτελούν τα βήματα εκσφαλμάτωσης
- Εκτελούν την εφαρμογή τους

Ενότητες Κεφαλαίου

- Εικονικές και φυσικές συσκευές
- Εκτέλεση
- Έλεγχος (test structure, test project)
- Επιμέρους εργαλεία ελέγχου
- Εκσφαλμάτωση

8.1 Εικονικές και φυσικές συσκευές

Σε προηγούμενο κεφάλαιο είδαμε πως η υποστήριξη διαφορετικών συσκευών επηρεάζει το σχεδιασμό της διεπαφής χρήστη και την οργάνωση του έργου ανάπτυξης γενικότερα. Ο τρόπος που «τρέχει» η εφαρμογή και η αλληλεπίδραση μαζί της αλλάζει ελαφρά από συσκευή σε συσκευή. Για το λόγο αυτό είναι σκόπιμο, αν σκοπεύετε να υποστηρίξετε πολλαπλές συσκευές καλό είναι να δοκιμάζετε την εφαρμογή είτε πάνω στις συγκεκριμένες συσκευές είτε στον προσομοιωτή (emulator) που διαθέτει για το σκοπό αυτό το περιβάλλον.

Ο προσομοιωτής έχει δυνατότητα να ρυθμίσει εικονικές συσκευές προσομοιώνοντας διάφορα χαρακτηριστικά και ρυθμίσεις του υλικού όσο και του λογισμικού. Δεδομένης της μεγάλης διάδοσης του Android σε ένα ευρύ φάσμα πραγματικών, υλικών συσκευών το ποιο πιθανό είναι ότι δεν θα διαθέτετε όλες τις υλικές συσκευές στις οποίες θα θέλατε να τρέχει η εφαρμογή σας οπότε θα πρέπει να ρυθμίσετε κατάλληλες εικονικές συσκευές για να προσομοιώσετε τη λειτουργία τους. Προφανώς ο προσομοιωτής δεν παρέχει όλες τις πιθανές ρυθμίσεις αλλά διαθέτει τις συχνότερα αξιοποιούμενες λειτουργικότητες.

8.1.1 Υλικές Συσκευές

Οι υλικές συσκευές είναι πάντα προτιμότερες για τις δοκιμές σας. Δίνουν καλύτερη εικόνα της εμπειρίας του χρήστη από την εφαρμογή ενώ είναι ο μόνος τρόπος να ελέγξει κανείς κάποιες λειτουργικότητες που δεν υλοποιούνται καλά στον emulator όπως η διαχείριση των τηλεφωνημάτων ή η λειτουργία της πίσω κάμερας. Ακόμα και αν θέλετε να τρέχει και σε παρόμοιες συσκευές με διαφορετικά χαρακτηριστικά η εφαρμογή σας, μπορείτε να ελέγξετε μόνο τις επιπλέον λειτουργικότητες με εικονικές συσκευές διασφαλίζοντας καλύτερη εικόνα για την τελική εμπειρία του χρήστη.

Ο συνηθισμένος τρόπος να χρησιμοποιήσουμε μια υλική συσκευή κατά την ανάπτυξη είναι συνδέοντας τη συσκευή στον υπολογιστή μας μέσω USB. Η σύνδεση συνήθως αναγνωρίζεται αυτόματα από το σύστημα και μετά η συσκευή διατίθεται για χρήση στο Eclipse, με το οποίο αλληλεπιδρά. Μπορούμε να επιβεβαιώσουμε ότι αυτό έχει συμβεί αν στο Eclipse επιλέξουμε διαδοχικά Window →Open Perspective → DDMS, τότε η συσκευή θα περιλαμβάνεται στην Όψη Συσκευών (Device View) στα αριστερά. Επίσης μηνύματα σχετικά με τις διαδικασίες που εκτελούνται στη συσκευή θα εμφανίζονται στην Όψη Καταλόγου Εγγραφών (LogCat View).

Αν η σύνδεση δεν ολοκληρωθεί αυτόματα πιθανότατα δεν είναι ενεργή η εκσφαλμάτωση μέσω USB. Πηγαίνουμε στις ρυθμίσεις (settings) της συσκευής, επιλέγουμε διαδοχικά Developer Options → Advanced Settings ή εναλλακτικά Applications → Development και επιλέγουμε το κουτάκι που ενεργοποιεί το USB debugging.

8.1.2 Εικονικές συσκευές

Για τις εικονικές συσκευές τοEclipse χρησιμοποιεί τον διαχειριστή εικονικών συσκευών Android (Android Virtual Device Manager). Σε αυτόν μπορείτε είτε να κατασκευάσετε μία εικονική συσκευή με τα χαρακτηριστικά που επιθυμείτε είτε να χρησιμοποιήσετε υπάρχοντες τέτοιους οδηγούς.

Μπορείτε να χρησιμοποιήσετε τους υπάρχοντες ορισμούς με δύο τρόπους: είτε ως έχουν είτε δημιουργώντας ένα αντίτυπο (επιλογή Clone) το οποίο τροποποιείτε κατάλληλα για να προσεγγίσει καλύτερα τα επιθυμητά χαρακτηριστικά. Μόνο κλώνοι των αρχικών ορισμών μπορούν να υποστούν επεξεργασία.

Αν γνωρίζετε ακριβώς τα χαρακτηριστικά της συσκευής που σας ενδιαφέρει και αυτή δεν υπάρχει στις προεγκατεστημένες μπορείτε να εκκινήσετε τη δημιουργία νέας συσκευής. Εκεί θα

χρειαστεί να δώσετε όνομα και τιμές για μια σειρά χαρακτηριστικών που ξεκινούν από τη διάσταση της οθόνης και φτάνουν στον προσανατολισμό περνώντας από την ανάλυση οθόνης, τη διαθεσιμότητα αισθητήρων και καμερών, τους τρόπους εισόδου.

Σε κάθε περίπτωση, αφού έχει κατασκευαστεί ο κατάλληλος ορισμός δημιουργείτε μία εικονική συσκευή (AVD) από αυτόν. Το Eclipse θα σας επιστρέψει στη ν ετικέτα AVD όπου θα φαίνεται η νέα συσκευή. Επιλέξτε τη και μετά επιλέξτε εκκίνηση (Start) για να την τρέξετε. Στο αναδυόμενο μενού επιλέξτε Launch και το Eclipse θα τρέξει τον προσομοιωτή με τη νέα AVD μέσα σε αυτόν. Κλείστε τον AVD Manager σε αυτό το σημείο.

Όταν ο προσομοιωτής δείχνει γραφική απεικόνιση της συσκευής μπορείτε να αλληλοεπιδράσετε με αυτή πατώντας τα κουμπιά με το ποντίκι ενώ κάποιες συντομεύσεις λειτουργούν από το πληκτρολόγιο όπως το home για το Home της συσκευής, το ESC για το back της συσκευής, το F2 για το κουμπί menu και άλλα που μπορείτε να βρείτε εδώ: http://developer.android.com/tools/help/emulator.html .

Αν επιστρέψουμε όσο η AVD είναι ανοιχτή στο Eclipse και ανοίξουμε και πάλι την οπτική DDMS θα διαπιστώσουμε ότι η εικονική συσκευή εμφανίζεται στη λίστα της Όψης συσκευών ακριβώς σαν να ήταν υλική. Επίσης θα περιλαμβάνονται μια σειρά διαδικασίες τις οποίες αν επιλέξουμε ενεργοποιούνται διάφορα κουμπιά για την επεξεργασία τους. Η δυνατότητα αυτή είναι ιδιαίτερα χρήσιμη στην εκσφαλμάτωση.

Δραστηριότητα 8.1.2

Κατασκευάστε μια εικονική συσκευή και παίξτε λίγο μαζί της στον προσομοιωτή ώστε να αποκτήσετε αίσθηση της λειτουργικότητας που υλοποιεί.

8.2 Εκτέλεση

Όταν χρησιμοποιούμε ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) και «τρέχουμε» δοκιμαστικά την εφαρμογή μας σε κάποια συσκευή, το περιβάλλον ρυθμίζει τις λεπτομέρειες διευκολύνοντας σημαντικά τη διαδικασία. Παράγει ένα εκτελέσιμο αρχείο εφαρμογής, που στο Android έχει επέκταση APK και το εγκαθιστά στη συσκευή είτε αυτή είναι υλική είτε εικονική. Το ίδιο φυσικά γίνεται και με το Eclipse.

Αρχικά μπορούμε να «τρέξουμε» την εφαρμογή μέσα από το περιβάλλον ανάπτυξης . Σε αυτή την περίπτωση θα χρησιμοποιήσουμε είτε μια εικονική συσκευή είτε μια συσκευή που έχει συνδεθεί με το περιβάλλον με τον τρόπο που περιγράψαμε στην αντίστοιχη παράγραφο της προηγούμενης ενότητας. Φυσικά για να μπορεί να εγκατασταθεί η εφαρμογή στη συσκευή αυτή θα πρέπει να ικανοποιεί τις ελάχιστες απαιτήσεις API που προσδιορίσατε στο δηλωτικό.

Προφανώς αν έχετε συνδέσει περισσότερες από μία συσκευές στο περιβάλλον, για παράδειγμα μια εικονική και μια υλική, θα πρέπει να επιλέξετε σε ποια θα τρέξει η εφαρμογή την πρώτη φορά. Μπορείτε επίσης να επιλέξετε να «τρέχει» πάντα στην ίδια ή να σας ρωτά. Επειδή όταν ετοιμάζει το εκτελέσιμο σας ζητά ένα νέο όνομα με το οποίο όταν ετοιμαστεί εμφανίζεται στην περιοχή εφαρμογών Android (Android Application) στη λίστα ρυθμίσεων εκτέλεσης (Run Configurations).

Όταν η εφαρμογή επιλεγεί για εκτέλεση, το Eclipse αντιγράφει το APK στη συσκευή, το εγκαθιστά και ξεκινά την κύρια Δραστηριότητα (main Activity). Τη διαδικασία αυτή θα την χρησιμοποιήσουμε επανειλημμένα κατά τη διάρκεια της εκσφαλμάτωσης τρέχοντας, διορθώνοντας και ξανατρέχοντας την εφαρμογή.

Καλό είναι να καταγράφονται κάποιες παρατηρήσεις από κάθε δοκιμαστική εκτέλεση. Για το σκοπό αυτό πρέπει να τηρείται ένα αρχείο σχετικών καταγραφών το λεγόμενο Log. Αυτές οι καταγραφές είναι ουσιαστικά μηνύματα που γράφουμε μέσα από τα αρχεία Java στο LogCat για να

παρακολουθήσουμε καλύτερα την εξέλιξη της εκτέλεσης. Οι καταγραφές αυτές (log messages) είναι περισσότερο χρήσιμες όταν είναι φανερό από ποια κλάση προέρχονται και για αυτό το λόγο καλό είναι να οργανωθούν τα μηνύματα ανάλογα.

Δραστηριότητα 8.2.1

Δημιουργείστε την υποδομή για την παρακολούθηση της εκτέλεσης της εφαρμογής σας

8.3 Έλεγχος (test structure, test project)

Ο έλεγχος μιας εφαρμογής στο Android έχει μια ιδιοτυπία: στο Eclipse, απαιτεί τη δημιουργία ενός έργου ελέγχου (test project). Τα εργαλεία που χρησιμοποιούμε στηρίζονται στο JUnit αλλά με επεκτάσεις ειδικά για το Android που προσφέρουν αφιερωμένους πόρους ελέγχου. Με δεδομένη την εμπειρία από το πρώτο μέρος δεν θα πρέπει να αντιμετωπίσετε σημαντικές δυσκολίες.

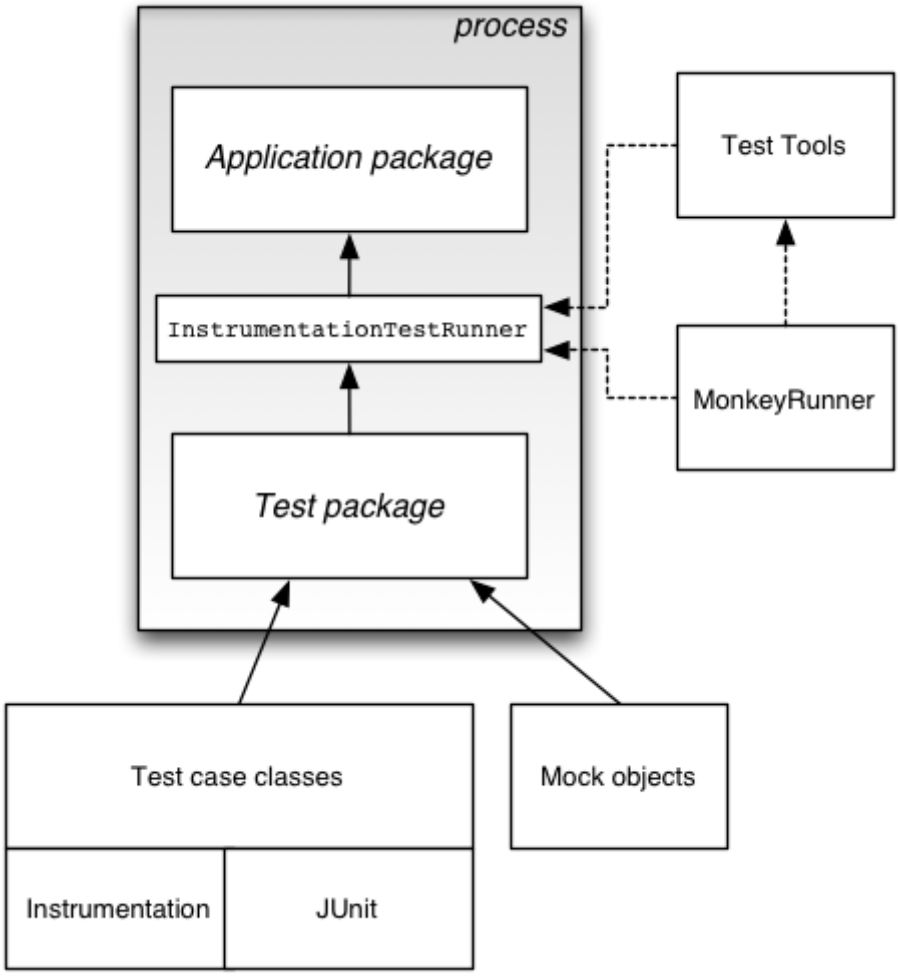
Οι επεκτάσεις για το Android του JUnit παρέχουν κλάσεις ελέγχου εξειδικευμένες στις συνιστώσες της εφαρμογής. Αυτές οι κλάσεις παρέχουν μεθόδους για τη δημιουργία ψευδοαντικειμένων και μεθόδους που βοηθούν στον έλεγχο του κύκλου ζωής μιας συνιστώσας . Οι περιπτώσεις δοκιμών περιέχονται σε πακέτα ελέγχου που είναι παρόμοια με πακέτα στοιχείων εφαρμογής , έτσι ώστε να μην χρειάζεται εξοικείωση με ένα νέο σετ από εργαλεία ή τεχνικές για το σχεδιασμό και τη δόμηση ελέγχων.

Τα εργαλεία SDK για την κατασκευή και τις δοκιμές είναι διαθέσιμα στο Eclipse μέσω των ADT. Αυτά τα εργαλεία αντλούν πληροφορίες από το έργο της εφαρμογής που βρίσκεται υπό δοκιμή και χρησιμοποιούν αυτές τις πληροφορίες για να δημιουργήσουν αυτόματα τα αρχεία κατασκευής , το δηλωτικό , και τη δομή καταλόγου για το πακέτο ελέγχου.

Με αυτή την έννοια η διαδικασία ελέγχου, που στηρίζεται στο ακόλουθο σχήμα, δεν είναι ιδιαίτερα διαφορετική από τη δημιουργία του ίδιου του έργου. Αντίθετα εμφανίζει ένα βαθμό αυτοματισμού και χρησιμοποιεί οικείες λειτουργίες.

Στο JUnit , οικοδομούμε ένα ή περισσότερα αρχεία κώδικα δοκιμών σε ένα αρχείο κλάσης. Ομοίως , στο Android χρησιμοποιούμε εργαλεία του SDK για να οικοδομήσουμε τα περιεχόμενα ενός ή περισσότερων αρχείων κώδικα δοκιμών σε αρχεία κλάσης μέσα σε ένα Android πακέτο δοκιμών . Στο JUnit , μπορείτε να χρησιμοποιήσετε ένα βοηθό ελέγχων για να εκτελέσει τις κλάσεις δοκιμών . Στο Android , μπορείτε να χρησιμοποιήσετε εργαλεία δοκιμής για να φορτώσετε το πακέτο δοκιμής και την υπό εξέταση εφαρμογή, και στη συνέχεια τα εργαλεία να εκτελέσουν ένα εξειδικευμένο για το Android βοηθό ελέγχων.

Ένα έργο ελέγχου είναι ένας κατάλογος, ή στην περίπτωσή μας ένα έργο Eclipse, στο οποίο δημιουργούμε τον πηγαίο κώδικα, το αρχείο δηλωτικού και άλλα απαραίτητα αρχεία για ένα πακέτο ελέγχου. Για περισσότερες πληροφορίες σχετικά δες τις ενότητες The Testing API και Running Tests στη θέση: http://developer.android.com/tools/testing/testing_android.html .



Σχήμα 8.3.1 Διαγραμματική αποτύπωση του πλαισίου ελέγχων

Για την υλοποίηση ελέγχων στην πράξη μπορείτε να δείτε εδώ:

http://developer.android.com/training/testing.html

8.4 Επιμέρους εργαλεία ελέγχου

Το SDK παρέχει δύο εργαλεία για έλεγχο εφαρμογής στο λειτουργικό επίπεδο.

Το μεν UI/Application Exerciser Monkey είναι ένα εργαλείο γραμμής εντολών που στέλνει ψευδο-τυχαία ρεύματα πληκτρολόγησης, αγγιγμάτων και χειρονομιών σε μια συσκευή. Μπορείτε να το εκτελέσετε με το εργαλείο Android Debug Bridge (ADB). Χρησιμοποιείται για την πραγματοποίηση stress-test στην εφαρμογή και την αναφορά λαθών που συνάντησε.

Το δε εργαλείο monkeyrunner είναι ένα API και περιβάλλον εκτέλεσης για προγράμματα ελέγχου γραμμένα σε python. Το API περιλαμβάνει λειτουργίες για τη σύνδεση σε μια συσκευή , την εγκατάσταση και την απεγκατάσταση πακέτων , τη λήψη στιγμιότυπων , τη σύγκριση δύο εικόνων , και για να τρέχει ένα πακέτο ελέγχου πάνω σε μια εφαρμογή. Χρησιμοποιώντας το API , μπορείτε να γράψετε ένα ευρύ φάσμα μεγάλων , ισχυρών και πολύπλοκων ελέγχων . Μπορείτε να εκτελέσετε προγράμματα που χρησιμοποιούν το API με το εργαλείο γραμμής εντολών monkeyrunner.

8.5 Εκσφαλμάτωση

Η εκσφαλμάτωση είναι μια κρίσιμη και πολύπλοκη διαδικασία στην οποία πρέπει να υποβάλλουμε τις εφαρμογές μας για να φτάσουν στο επίπεδο ποιότητας που χαρακτηρίζεται επαγγελματικό. Η διαδικασία περιλαμβάνει και ενσωματώνει εργαλεία και πρακτικές σαν αυτές που αναφέραμε στις προηγούμενες παραγράφους.

Η εκσφαλμάτωση όταν γίνεται σε υψηλό επίπεδο διασφαλίζει τη βελτιστοποίηση της εφαρμογής και απαιτεί ένα σημαντικό σετ δεξιοτήτων που καλλιεργείται με μακροχρόνια εξάσκηση. Όμως υπάρχει ένα χαμηλότερο επίπεδο το οποίο είναι απαραίτητο να αποκτήσει κάθε προγραμματιστής κι αυτό γιατί συχνά χωρίς την εκσφαλμάτωση η εφαρμογή μπορεί απλά να μην «τρέχει». Περισσότερες πληροφορίες για την εκσφαλμάτωση υπάρχουν σε αυτή τη θέση: http://developer.android.com/tools/debugging/index.htm .

Ευτυχώς για εμάς τα ολοκληρωμένα περιβάλλοντα ανάπτυξης συνήθως προσφέρουν μια σειρά εργαλεία που διευκολύνουν τη διαδικασία της εκσφαλμάτωσης. Το Eclipse δεν αποτελεί εξαίρεση: διαθέτει μια αφιερωμένη οπτική debug και το γνωστό μας DDMS (Dalvik Debug Monitor System) του οποίου τα εργαλεία είναι χρήσιμα καθ' όλη τη διαδικασία ανάπτυξης και όχι μόνο για τη φάση της εκσφαλμάτωσης.

Η οπτική debug περιλαμβάνει την ομώνυμη ετικέτα όπου καταλογογραφούνται ιστορικά εκσφαλματωνόμενες εφαρμογές Android και τα σχετικά νήματα μέσα από logs οργανωμένα σε ενότητες. Η ετικέτα μεταβλητές (Variables) περιλαμβάνει τις τιμές μεταβλητών στα σημεία ελέγχου (breakpoints) κατά την εκτέλεση του κώδικα. Τα ίδια τα breakpoints καταλογογραφούνται στην ομώνυμη καρτέλα και την όψη εκσφαλμάτωσης LogCat που έχουμε ήδη δει στην παράγραφο για την εκτέλεση. Η οπτική debug ενεργοποιείται επιλέγοντας διαδοχικά Window → Open Perspective → Debug.

Με μια ενεργή συσκευή συνδεδεμένη στο Eclipse μπορούμε να ανοίξουμε την οπτική DDMS επιλέγοντας διαδοχικά Window → Open Perspective → DDMS. Η οπτική DDMS περιλαμβάνει αρκετές διαφορετικές όψεις. Έχουμε ήδη εξοικειωθεί με την όψη συσκευών όπου φαίνεται η λίστα των συνδεδεμένων συσκευών και σύνδεσμοι σε κάθε διαδικασία που «τρέχει» σε κάποια από αυτές. Μπορούμε να εντοπίσουμε την εφαρμογή μας από το όνομα πακέτου. Αν την επιλέξουμε ενεργοποιούνται τα κουμπιά της όψης. Από την όψη συσκευών (Device View) μπορούμε να σταματήσουμε μια ενεργή (εκτελούμενη) διαδικασία, να την εκσφαλματώσουμε και να επιβάλουμε συλλογή σκουπιδιών (garbage collection).

Δραστηριότητα 8.5.1

Εξοικειωθείτε με τις όψεις (Views)του DDMS. Ανοίξτε και ενημερώστε από τα κουμπιά της όψης συσκευών την όψη νημάτων χρησιμοποιώντας το κουμπί Update Threads, την όψη σωρού πατώντας το κουμπί Update Heap.

Από την όψη προσομοιωτή διαπιστώστε ότι έχετε έλεγχο πάνω στις εικονικές συσκευές και τις διαδικασίες τους.

Από την όψη συσκευών επιλέξτε την εφαρμογή σας και πατήστε το κουμπί Screen Capture (αυτό που μοιάζει με φωτογραφική μηχανή). Αποθηκεύστε την εικόνα που θα εμφανιστεί στο αναδυόμενο παράθυρο για να τη χρησιμοποιήσετε κατά τη δημοσίευση.

Κεφάλαιο 9

Κύκλος ζωής δραστηριότητας (Activity)

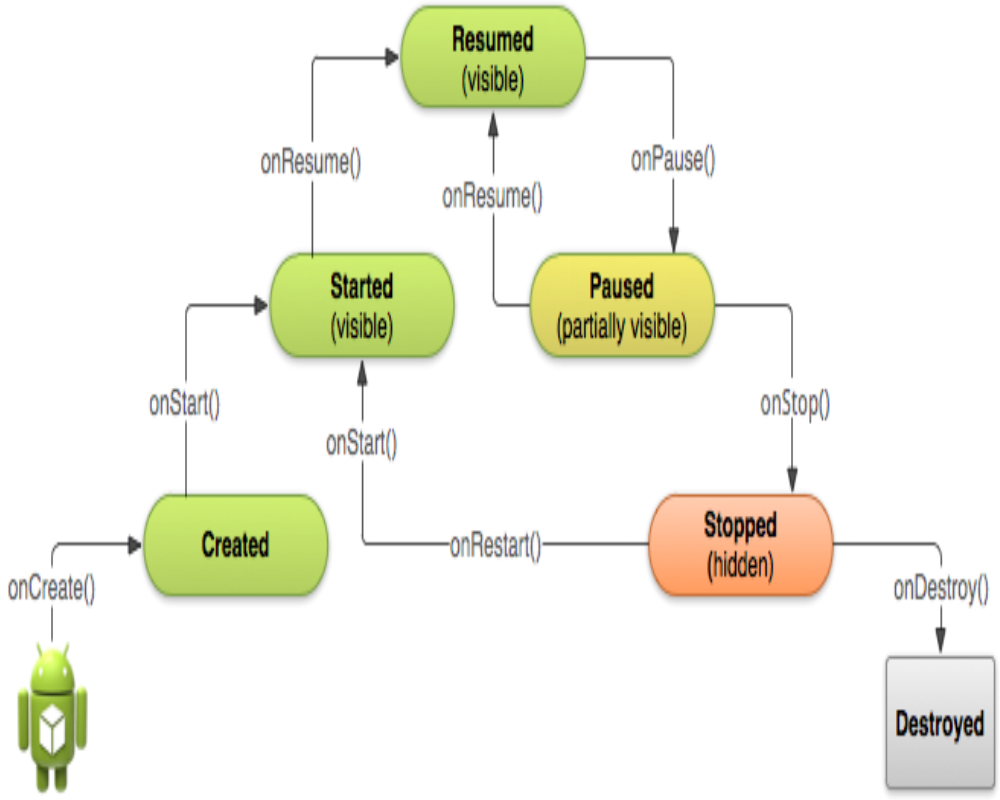
9. Κύκλος ζωής δραστηριότητας (Activity)

Μια κρίσιμη έννοια για να γίνει κατανοητή η λειτουργία κάθε εφαρμογής αλλά και η ταυτόχρονη παρουσία πολλαπλών εφαρμογών σε μια συσκευή Android είναι αυτή του κύκλου ζωής μιας δραστηριότητας. Έχετε προσέξει ποτέ ότι σε μια συσκευή Android συχνά εφαρμογές που έχετε διακόψει τη χρήση τους παραμένουν ενεργές; Βάρυνε η συσκευή σας μετά από κάποιο διάστημα χρήσης χωρίς να φαίνεται να υπάρχει κάποια εφαρμογή ανοιχτή;

Αν χρησιμοποιήσετε τη λάθος μέθοδο ανάκλησης για τη δουλειά που θέλετε να κάνει η εφαρμογή σας, αν ξεχάσετε ότι τα δομικά της στοιχεία έχουν μια σχετική αυτονομία και πρέπει να τα διαχειριστείτε καμιά φορά ξεχωριστά μπορεί να προκαλέσετε τέτοια προβλήματα στους χρήστες σας. Αν ξεχάσετε να αλλάξετε την κατάσταση κάποιας δραστηριότητας, ενώ θα έπρεπε, τα πράγματα επίσης μπορεί να περιπλακούν.

Η μετάβαση από το προσκήνιο στο παρασκήνιο και πίσω επίσης δοκιμάζει τις εφαρμογές. Για να γίνει σωστά η μετάβαση πρέπει να μπορεί η εφαρμογή να διαχειριστεί διάφορες ενέργειες όπως η αποθήκευση και ανάκτηση δεδομένων. Η διαχείριση αυτή γίνεται μέσα από τις διαδικασίες κλήσης/ ανάκλησης που αλλάζουν την κατάσταση της εφαρμογής με κατάλληλο τρόπο.

Το ακόλουθο διάγραμμα (ανακτημένο από τον οδηγό για Android Developers και τη θέση http://developer.android.com/training/basics/activity-lifecycle/starting.html#lifecycle) δίνει μια απλοποιημένη απεικόνιση του κύκλου ζωής μιας δραστηριότητας. Δείχνει επίσης ποιες μέθοδοι κλήσης (callback) χρησιμοποιούνται για να κινηθούμε προς και από την κατάσταση συνέχισης.



Σχήμα 8.5.1 Κύκλος ζωής δραστηριότητας (ανακτημένο από τον οδηγό για Android Developers: http://developer.android.com/training/basics/activity-lifecycle/starting.html#lifecycle.

Στόχοι

Οι μαθητές να μπορούν να:

- Περιγράφουν τις διαφορετικές καταστάσεις στις οποίες μπορεί να βρίσκεται μια δραστηριότητα
- Αναγνωρίζουν την επίδραση αυτής της κατάστασης στη λειτουργία μια εφαρμογής
- Χρησιμοποιούν μεθόδους ανάκλησης
- Αναγνωρίζουν τον τρόπο συμπεριφοράς μιας εφαρμογής σε σχέση με την κατάσταση στην οποία βρίσκεται

Ενότητες Κεφαλαίου

- Μέθοδοι ανάκλησης (Callback)
- Καταστάσεις δραστηριοτήτων (activities) (συνέχισης –resumed, αναστολής- paused, καταστροφής destroyed)
- Συνέπειες του κύκλου ζωής δραστηριοτήτων

9.1 Μέθοδοι ανάκλησης (Callback)

Κάθε μέθοδος ανάκλησης προκαλεί τη μετάβαση της Δραστηριότητας σε κάποια συγκεκριμένη από τις διαθέσιμες καταστάσεις. Αυτό μπορεί να γίνει είτε από το ίδιο το Android είτε από μια ενεργή δραστηριότητα. Οι μέθοδοι αυτές είναι οι ίδιες που χρησιμοποιήσαμε στην περιγραφή του κύκλου ζωής της διαδικασίας για τη μετάβαση μεταξύ των καταστάσεών της και είναι ορισμένες από το Android. Μέσα στην αντίστοιχη κάθε φορά μέθοδο μπορούμε να προγραμματίσουμε τις αντιδράσεις της Δραστηριότητας σε μια σειρά συνδεόμενα με την κατάστασή της γεγονότα.

Γνωρίζοντας ποια μέθοδο καλεί το Android όταν η δραστηριότητά σας εισέρχεται σε κάθε κατάσταση θα σας επιτρέψει να διασφαλίσετε ότι η δραστηριότητα συνεχίζει να λειτουργεί, δε χάνει δεδομένα ή δεν απασχολεί περισσότερους από τους απαραίτητους πόρους. Και είναι καλό να έχετε στο μυαλό σας το διαφορετικό τρόπο με τον οποίο λειτουργεί το συγκεκριμένο περιβάλλον.

Για παράδειγμα, όταν εκκινεί η εφαρμογή σας αντί να ενεργοποιείται μια κύρια μέθοδος, εκτελείται η μέθοδος onCreate της κλάσης κύριας δραστηριότητας. Στο δηλωτικό έχουμε προσδιορίσει την κλάση αυτή ως κύρια δραστηριότητα εκκίνησης. Η μέθοδος onCreate είναι η πρώτη από τις μεθόδους ανάκλησης της Δραστηριότητας και για αυτό εκτελείται μόλις ο χρήστης εκκινήσει την εφαρμογή. Η onCreate βάζει την *δραστηριότητα* στην κατάσταση δημιουργημένη (Created).

Η κατάσταση συνέχισης (Resumed) με την οποία «τρέχει» η εφαρμογή όσο ο χρήστης αλληλεπιδρά μαζί της ακολουθεί τη μέθοδο onResume. Παρεμβάλλεται η κατάσταση εκκίνησης (Started) την οποία ενεργοποιεί η μέθοδος onStart.

Όσο απομακρυνόμαστε από την κατάσταση συνέχισης κινούμαστε προς την κατάσταση αναστολής (Paused) μέσω της onPause και από την οποία μπορούμε να επιστρέψουμε με την onResume. Το επόμενο βήμα είναι η κατάσταση διακοπής (Stopped), η οποία προκαλείται από την onStop. Από αυτήν μπορούμε να επιστρέψουμε στην κατάσταση εκκίνησης μέσω onRestart ή να καταλήξουμε στην κατεστραμμένη (Destroyed) κατάσταση μέσω onDestroy.

Δεν είναι απαραίτητη η υλοποίηση όλων των μεθόδων ούτε η διάσχιση όλων των καταστάσεων. Είναι βέβαια προφανώς απαραίτητη η onCreate ενώ σκόπιμη είναι επίσης η υλοποίηση των onPause και onDestroy αν θέλουμε η εφαρμογή μας να μην έχει αρνητικές συνέπειες στην ισορροπία του συστήματος.

9.2 Καταστάσεις δραστηριοτήτων (activities) (συνέχισης –resumed, αναστολής- paused, κατεστραμμένη- destroyed)

Οι κύριες καταστάσεις μιας εφαρμογής είναι η κατάσταση συνέχισης, η κατάσταση αναστολής και η τερματισμένη κατάσταση. Οι άλλες είναι μεταβατικές καταστάσεις και η εφαρμογή δεν παραμένει σε αυτές.

Όσο η εφαρμογή είναι ενεργή και ο χρήστης αλληλοεπιδρά μαζί της λέμε ότι η εφαρμογή είναι σε κατάσταση συνέχισης. Μεταβαίνει σε κατάσταση αναστολής όταν μια άλλη δραστηριότητα είναι στο προσκήνιο και σε ένα βαθμό την καλύπτει. Ο χρήστης δεν έχει δυνατότητα αλληλεπίδρασης με την εφαρμογή όσο αυτή είναι στην κατάσταση αναστολής, μπορεί όμως να την ανακαλέσει από αυτήν.

Όταν η εφαρμογή βρίσκεται εντελώς στο παρασκήνιο και ο χρήστης δεν μπορεί να την δει και να αλληλοεπιδράσει μαζί της, λέμε ότι η εφαρμογή βρίσκεται σε τερματισμένη κατάσταση ή σε κατάσταση τερματισμού. Στην κατάσταση αυτή η δραστηριότητα μπορεί να διατηρήσει δεδομένα αλλά όχι να εκτελέσει. Συχνά εφαρμογές που δεν υπάρχει λόγος παραμένουν σε αυτή την

κατάσταση λόγω παράλειψης υλοποίησης της μεθόδου που θα οδηγούσε στην καταστροφή τους απελευθερώνοντας πόρους του συστήματος. Όταν μια εφαρμογή είναι σε καταστροφή στην πραγματικότητα δεν βρίσκεται πλέον στην συσκευή με μορφή εκτελέσιμου.

9.3 Συνέπειες του κύκλου ζωής δραστηριοτήτων

Ο κύκλος ζωής των δραστηριοτήτων έχει μια σημαντική συνέπεια: υπάρχει κατάλληλη θέση για κάποιες ενέργειες. Παράλληλα και επειδή υπάρχει η δυνατότητα για πολλαπλές δραστηριότητες σε κάθε εφαρμογή – δυνατότητα που οι περισσότερες εφαρμογές εκμεταλλεύονται – η εκκίνηση, κυρίως, μιας δραστηριότητας μπορεί να γίνει από άλλη, απαιτώντας προσοχή στους χειρισμούς. Ας δούμε όμως αναλυτικά την πορεία προς και από την κατάσταση συνέχισης:

Όταν εκκινεί η κύρια δραστηριότητα μιας εφαρμογής, την στιγμή της εκκίνησης της εφαρμογής, εκτελείται η μέθοδος onCreate επιτρέποντας την προετοιμασία της διεπαφής χρήστη της δραστηριότητας αλλά και προετοιμάζοντας τμήματα δεδομένων χρήσιμα γενικά για την κλάση. Οι τυχόν άλλες δραστηριότητες της ίδιας εφαρμογής ενεργοποιούνται κατά την αλληλεπίδραση με τον χρήστη. Ο τρόπος εκκίνησης βασίζεται στη χρήση της κλάσης των προθέσεων (Intents) κάπως έτσι:

```
Intent NewActivityIntent = new Intent(this, NewActivity.class);

startActivity(NewActivityIntent);
```

Υποθέτοντας ότι την έχουμε ήδη περιλάβει, μαζί με κατάλληλα Intent filtersστο δηλωτικό.

Είτε είμαστε στην κύρια δραστηριότητα είτε σε κάποια από τις άλλες, μόλις η onCreate μέθοδος της δραστηριότητας εκτελεστεί, ακολουθούν άμεσα οι onStart και onResume, οδηγώντας τη δραστηριότητα στην κατάσταση συνέχισης, περνώντας ενδιάμεσα από την κατάσταση δημιουργημένη και την κατάσταση εκκίνησης χωρίς να παραμένει σε αυτές.

Φυσικά στην κατάσταση συνέχισης η εφαρμογή μπορεί επίσης να βρεθεί από την κατάσταση αναστολής ή την κατάσταση διακοπής, επιστρέφοντας στο προσκήνιο στην πρώτη περίπτωση με χρήση της μεθόδου onResume και στην δεύτερη με εκτέλεση της onRestart την οποία ακολουθούν οι onStart και onResume.

Κατά την έξοδο ή απόκρυψη της εφαρμογής αυτή κατευθύνεται από την κατάσταση συνέχισης προς την κατεστραμμένη κατάσταση. Πρώτη στάση είναι η κατάσταση αναστολής στην οποία οδηγείται από τη μέθοδο onPause. Στη μέθοδο αυτή θα πρέπει να υπάρχει πρόβλεψη διακοπής κάθε απαιτητικής σε πόρους δραστηριότητας όπως κινούμενα γραφικά, διαχείριση αισθητήρων και δέκτες εκπομπών.

Όταν εκτελείται η onPause είναι πιθανή και η εκτέλεση της onStop λόγω πιθανής απομάκρυνσης του χρήστη από την εφαρμογή. Επομένως ίσως είναι σκόπιμο να χρησιμοποιηθεί η onPause για αποθήκευση δεδομένων παρότι συνήθως αυτό γίνεται στην onStop. Αν πάντως χρησιμοποιηθεί η onResume για επιστροφή στην κατάσταση συνέχισης είναι σκόπιμο στην τελευταία να ανακτηθούν απελευθερωμένοι πόροι και να επανεκκινηθούν άλλες διεργασίες και υπηρεσίες που πιθανόν διακόπηκαν από την onPause, παρόλο που η onResume εκτελείται επίσης κατά την εκκίνηση.

Η onStop εκτελείται μετά την onPause όταν η εφαρμογή μπαίνει σε κατάσταση διακοπής. Στην onStop είναι σκόπιμη η εκτέλεση ενεργειών αποθήκευσης δεδομένων, όπως ιδιαίτερα η εγγραφή σε βάσεις δεδομένων. Η απελευθέρωση πόρων που χρησιμοποιεί η εφαρμογή την προστατεύει από διαρροές μνήμης αν ακολουθήσει καταστροφή της εφαρμογής.

Στην περίπτωση επιστροφής στην κατάσταση συνέχισης θα εκτελεστούν οι τρεις μέθοδοι που αναφέραμε νωρίτερα. Από αυτές η onRestart εκτελείται μόνο σε αυτήν την περίπτωση οπότε είναι σκόπιμο σε αυτήν να επαναφέρονται όσα στην onStop διακόπηκαν ή αποθηκεύτηκαν. Το

σύστημα από μόνο του αποθηκεύει κάποια στοιχεία δεδομένων όταν η εφαρμογή επανέλθει από διακοπή όπως το τι φαίνεται στις διάφορες όψεις.

Κεφάλαιο 10

Συνήθεις συνιστώσες (components) εφαρμογών Android

10. Συνήθεις συνιστώσες (components) εφαρμογών Android

Σκοπός του παρόντος κεφαλαίου είναι να εξοικειωθούν οι μαθητές με συνιστώσες εφαρμογών Abdroid πέρα από τις δραστηριότητες, να κατανοούν τον τρόπο λειτουργίας και αλληλεπίδρασής τους και να χρησιμοποιούν τις γνώσεις αυτές για την καλύτερη και γρηγορότερη ανάπτυξη των εφαρμογών τους.

Στόχοι

Οι μαθητές να μπορούν να:

- Χρησιμοποιούν εξωτερικές υπηρεσίες και περιεχόμενο
- Διαχειρίζονται τα μηνύματα συστήματος και εφαρμογών
- Χρησιμοποιούν έτοιμα λογικά τμήματα
- Οργανώνουν την εφαρμογή τους σε επαναχρησιμοποιήσιμα λογικά τμήματα

Ενότητες Κεφαλαίου

- Υπηρεσίες
- Πάροχοι περιεχομένου
- Δέκτες εκπομπών συστήματος και εφαρμογών
- Άλλες κλάσεις
- Λογικά τμήματα (fragments)
- Action Bar

10.1 Υπηρεσίες

Οι υπηρεσίες στο Android αντιστοιχούν σε διαδικασίες παρασκηνίου. Συνήθως πρόκειται για διαδικασίες που είναι διαρκείς ή ιδιαίτερα χρονοβόρες. Οι υπηρεσίες δεν διαθέτουν δική τους διεπαφή χρήστη οπότε συνήθως εκκινούνται από μία δραστηριότητα, μετά από κάποια ενέργεια του χρήστη, ο οποίος συνεχίζει να αλληλοεπιδρά με τη δραστηριότητα όσο η υπηρεσία εκτελείται στο παρασκήνιο.

Οι υπηρεσίες στο Eclipse δημιουργούνται με την ίδια διαδικασία που δημιουργούνται και οι δραστηριότητες απλά επιλέγοντας σαν υπερκλάση την Service. Είναι όμως διαφορετικές από τις δραστηριότητες καθώς για παράδειγμα, όταν ο χρήστης απομακρυνθεί από τη μητρική δραστηριότητα, σε μια άλλη εφαρμογή ή δραστηριότητα, η εκτέλεση της υπηρεσίας δεν διακόπτεται απαραίτητα.

Για να προσθέσουμε μια υπηρεσία στο δηλωτικό εισάγουμε, μέσα στην εγγραφή της αντίστοιχης δραστηριότητας, μια εγγραφή για την υπηρεσία του τύπου:

<service android:name=".ServiceClassName" />

Μια υπηρεσία, εφόσον είναι ενεργή, είναι προσβάσιμη από πολλαπλές συνιστώσες της εφαρμογής που την εκκίνησε αλλά ακόμα και από άλλες εφαρμογές. Αυτό μπορεί να γίνει εκκινώντας την με μία πρόθεση (Intent). Βέβαια, μπορούμε να ορίσουμε στο δηλωτικό μια υπηρεσία ως ιδιωτική (private) εμποδίζοντας έτσι την πρόσβασή της από άλλες εφαρμογές.

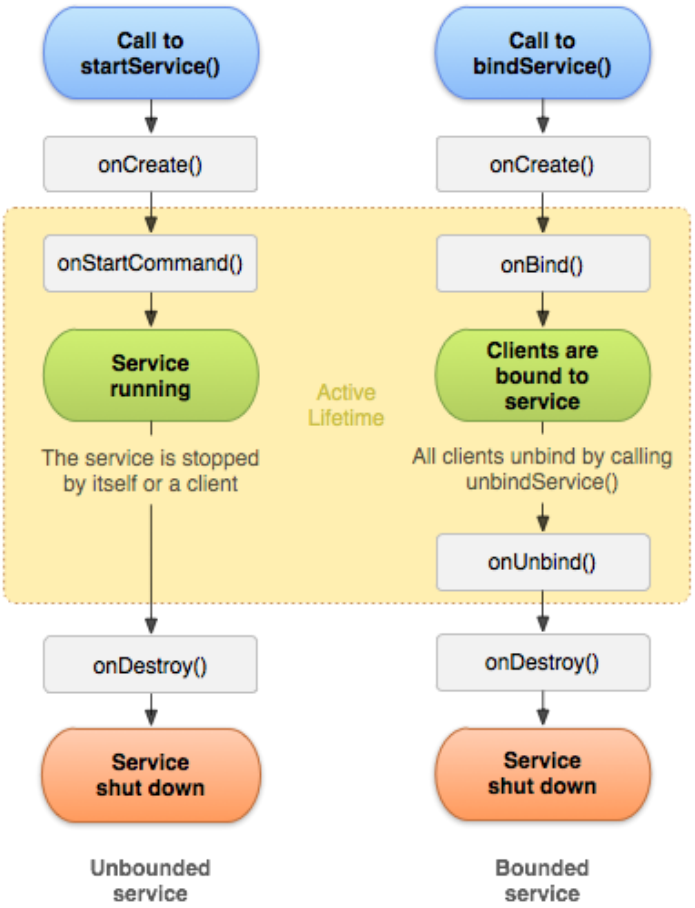
Οι υπηρεσίες έχουν δικές τους μεθόδους αλλαγής κατάστασης. Η εκκίνηση της υπηρεσίας μπορεί να γίνει εναλλακτικά είτε με τη μέθοδο onStartCommand είτε με τη μέθοδο onBind. Συνήθως υλοποιούμε τη μία μόνο από τις δύο μεθόδους.

Στην πρώτη περίπτωση, της υλοποίησης onStartCommand, η υπηρεσία αφού εκκινήσει μπορεί να «τρέχει» στο παρασκήνιο στο διηνεκές ακόμα και αν καταστραφεί η μητρική δραστηριότητα. Συχνά χρησιμοποιείται για απλές λειτουργίες που δεν επιστρέφουν αποτέλεσμα στη συνιστώσα κλήσης π.χ. κατεβάζουν ένα αρχείο από ή ανεβάζουν ένα αρχείο στο δίκτυο. Σε αυτή την περίπτωση η υπηρεσία θα πρέπει να μπορεί να διακόψει μόνη της τη λειτουργία της.

Στην δεύτερη περίπτωση, της υλοποίησης onBind, λέμε ότι έχουμε μια συνδεδεμένη (bind) υπηρεσία. Και σε μια συνδεδεμένη υπηρεσία μπορούμε να έχουμε πολλαπλές διαδικασίεςπου την καλούν. Όσο κάποια από αυτές είναι ενεργή το ίδιο ισχύει και για την υπηρεσία. Όταν όμως δεν υπάρχει πλέον συνδεδεμένη δραστηριότητα η υπηρεσία καταστρέφεται. Οι συνδεδεμένες υπηρεσίες διαθέτουν μια διεπαφή πελάτη-εξυπηρετητή (IBinder) που επιτρέπει σε συνιστώσες εφαρμογής να αλληλοεπιδρούν με την υπηρεσία, να στέλνουν αιτήματα, να παίρνουν αποτελέσματα κλπ.

Όταν έχουμε υλοποιήσει σε μια υπηρεσία τη μέθοδο onStartCommand αυτή εκκινεί από μια άλλη συνιστώσα μέσω της κλήσης της μεθόδου startService. Όπως αναφέραμε νωρίτερα πρέπει η ίδια να διακόψει τον εαυτό της, χρησιμοποιώντας τη μέθοδο stopSelf. Μπορεί ακόμα να τη διακόψει μια άλλη συνιστώσα καλώντας την stopService. Με τη διακοπή της υπηρεσίας, αυτή καταστρέφεται από το σύστημα.

Αντίστοιχα, για τις συνδεδεμένες υπηρεσίες μια συνιστώσα καλεί για την εκκίνησή τους την bindService. Μπορεί να διακόψει τη σύνδεση καλώντας την unbindService. Σε αυτή την περίπτωση οι stopService και stopSelf δεν διακόπτουν την υπηρεσία μέχρι να αποσυνδεθούν όλες οι συνδεδεμένες συνιστώσες. Εφόσον αποσυνδεθούν όλες οι συνδεδεμένες υπηρεσίες το σύστημα ούτως ή άλλως καταστρέφει την υπηρεσία.



Σχήμα 10.1.1 Συγκριτική παράθεση των κύκλων ζωής συνδεδεμένων και μη υπηρεσιών (ανάκτηση από http://developer.android.com/guide/components/services.html#Lifecycle)

10.2 Πάροχοι περιεχομένου

Ένας πάροχος περιεχομένου (Content Provider) είναι μια συνιστώσα εφαρμογής εξειδικευμένη στη διαχείριση δεδομένων. Είναι ουσιαστικά ένα σύνολο δεδομένων το οποίο μπορεί να είναι ιδιωτικό της εφαρμογής ή να είναι διαμοιράσιμο, προσφέροντας δυνατότητα σε άλλες εφαρμογές να ζητούν και να τροποποιούν τα δεδομένα του.

Οι πάροχοι μπορούν να ορίζουν permissions που θα πρέπει να έχουν οι εφαρμογές πελάτες τους για να μπορούν να τους χρησιμοποιήσουν. Για να χρησιμοποιήσει κανείς έναν πάροχο στην εφαρμογή του θα πρέπει να προσθέσει τις συναφείς permissions στο δηλωτικό.

Η επικοινωνία του παρόχου περιεχομένου εσωτερικά με συνιστώσες της ίδιας εφαρμογής εκμεταλλεύεται την κλάση επίλυσης περιεχομένου (content resolver) ώστε οι συνιστώσες να αλληλοεπιδρούν με τα δεδομένα. Το αντικείμενο ContentResolver βρίσκεται στον υποκατάλογο «android.content.Context». Για να εξυπηρετήσει αιτήματα από άλλες εφαρμογές ο πάροχος διαχειρίζεται την πρόσβαση στα δεδομένα μέσα από τυποποιημένες μεθόδους για αλληλεπίδραση με δομημένα σύνολα δεδομένων όπως οι βάσεις δεδομένων.

Οι μέθοδοι αυτές δεν είναι πολύ διαφορετικές από αυτές που χρησιμοποιούν οι διάφορες βάσεις δεδομένων, ειδικά οι σχεσιακές. Ο πάροχος παρουσιάζει τα δεδομένα σε ένα σύνολο πινάκων με γραμμές και στήλες όπου το κάθε κελί περιλαμβάνει μια μόνο τιμή. Έτσι η διαχείριση δεδομένων που επιστρέφει ένας πάροχος γίνεται με παρόμοιο τρόπο με τη διαχείριση των αποτελεσμάτων ενός ερωτήματος σε βάση δεδομένων.

Η χρήση έτοιμων παρόχων είτε προσφερόμενων από το σύστημα είτε σχεδιασμένων από άλλους δημιουργούς (developers) είναι αρκετά διαδεδομένη στην περίπτωση του Android. Τεχνικά χαρακτηριστικά όπως το ημερολόγιο και οι επαφές ανήκουν σε αυτή την κατηγορία.

Για την καλύτερη κατανόηση τον τρόπου λειτουργίας και του τρόπου κατασκευής ενός παρόχου μπορεί κανείς να χρησιμοποιήσει τις διευθύνσεις:

http://developer.android.com/guide/topics/providers/content-provider-basics.html και

http://developer.android.com/guide/topics/providers/content-provider-creating.html αντίστοιχα.

Παράδειγμα 10.2.1

Θα χρησιμοποιήσουμε ένα έτοιμο listView Activity widget, δηλαδή μια διαδικασία που το κύριο layout της περιέχει ένα στοιχείο λίστας με μια στήλη που περιέχει κείμενο. Χρησιμοποιώντας το ContentProvider του Android για τις επαφές κινητού θα γεμίσουμε την λίστα με αλφαριθμητικά που περιέχουν τα πεδία όνομα και τηλέφωνο.

Ας ονομάσουμε το XML για το layout “activity_example_cont_provider.xml”. Η βασική του δομή θα είναι κάπως έτσι:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:padding="10dp"
   android:textSize="16sp" >
</TextView>
```

Σημαντικό είναι να εισάγουμε τα permissions στο manifest.xml

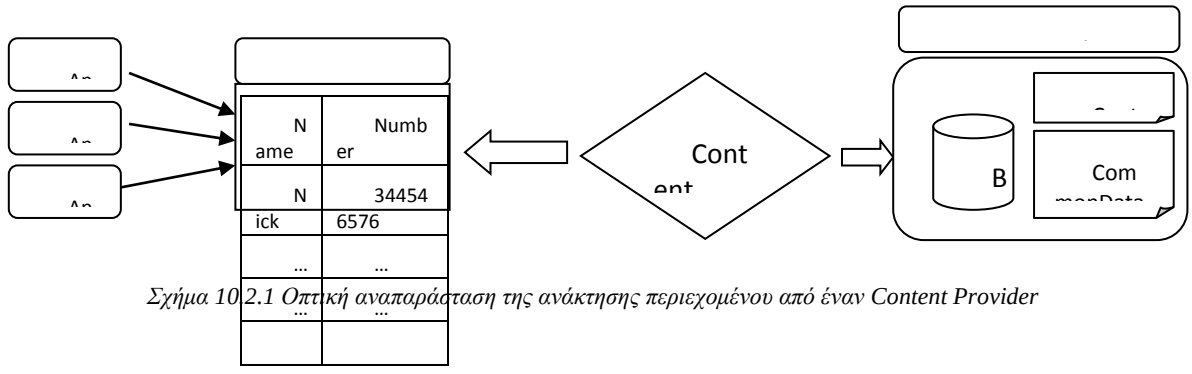
```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Και την επιλογή στην εφαρμογή

```
<application android:allowBackup="false"
```

Αλλιώς η εφαρμογή θα κολλά κατά την εκκίνηση επιστρέφοντας μήνυμα λάθους.

Η εφαρμογή διαβάζει δεδομένα από την βάση δεδομένων που περιέχει πληροφορίες για τις επαφές του κινητού. Στις επαφές γίνεται πρόσβαση μέσω του ContentProvider **ContactsContract** που παρέχεται από την πλατφόρμα. Με τον πίνακα Contacts έχουμε πρόσβαση στα στοιχεία των επαφών όπως το ονοματεπώνυμο και από τον πίνακα CommonDataKinds πρόσβαση στα δεδομένα επικοινωνίας κάθε επαφής όπως ο αριθμός τηλεφώνου.



Σχήμα 10.2.1 Οπτική αναπαράσταση της ανάκτησης περιεχομένου από έναν Content Provider

Η υλοποίηση είναι πολύ απλή και την χτίζουμε στο συμβάν onCreate() της νέας δραστηριότητας.

```
public class ExampleContProvider extends ListActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

// Δημιουργούμε ένα αντικείμενο λίστας που θα κρατά τα δεδομένα που θα ανακτήσουμε από το Provider
      List<String> contacts = new ArrayList<String>();

// Το αντικείμενο contentResolver conRes για να εκτελέσουμε ερωτήματα QUERIES στη βάση δεδομένων
      ContentResolver conRes = getContentResolver();

// Δημιουργούμε ένα string  με τα πεδία που θέλουμε να μας επιστραφούν από τη βάση δεδομένων. Καλό είναι να
//καθορίζουμε τα πεδία όταν διαχειριζόμαστε πίνακες μεγάλο αριθμό εγγράφων και πεδίων
      String[] contactFields= new String[]{ContactsContract.Contacts._ID,
            ContactsContract.Contacts.DISPLAY_NAME,
            ContactsContract.Contacts.HAS_PHONE_NUMBER};

// Cursor είναι το αντικείμενο όπου κρατά τα αποτελέσματα που επιστρέφει το ερώτημα.
      Cursor curs = conRes.query(ContactsContract.Contacts.CONTENT_URI,
// Το …CONTENT_URI είναι απαραίτητο σε κάθε ερώτημα που θέλει να έχει πρόσβαση σε συγκεκριμένα δεδομένα.
//Στη δική μας περίπτωση παρέχεται από την πλατφόρμα με την παραπάνω σταθερά.

            contactFields,          // String [],τα πεδία (στήλες) στις εγγραφές που θα επιστραφούν

            null,                   // φίλτρο επιλογής εγγραφές όπως το Where clause της SQL
            null,                   // επιπλέον περιεχόμενα επιλογής SQL args
            null);                  // ταξινόμηση των δεδομένων σε μορφή SQL order ASC , DES.  Όταν οι
                                    // παράμετροι είναι null δεν λαμβάνονται υπόψη οι παράμετροι και
                                    // επιστρέφονται όλες οι εγγραφές.


// η εντολή του Cursor moveToFirst() μετακινεί τον δείκτη εγγραφών στην αρχή ή επιστρέφει False

            if (curs.moveToFirst()) {
                  do {              // ξεκινά επανάληψη για να διαβαστούν όλες εγγραφές

                     // θέτουμε ένα  δείκτη της στήλης του συγκεκριμένου πεδίου
                     int colIndex = curs.getColumnIndex(ContactsContract.Contacts._ID);
                    // Διαβαζουμε τα δεδομένα για το πεδίο με το συγκεκριμένο δείκτη σε μια μεταβλητή
                   String contactID = curs.getString(colIndex);

                  // Επαναλαμβάνουμε την ίδια διαδικασία και για τα πεδία DISPLAY_NAME
                  // και HAS_PHONE_NUMBER
                    colIndex=curs.getColumnIndex(
                                 ContactsContract.Contacts.DISPLAY_NAME);
                                 String fieldName=curs.getString(colIndex);

                    colIndex=curs.getColumnIndex(
                               ContactsContract.Contacts.HAS_PHONE_NUMBER);
                               String fieldHasPhone=curs.getString(colIndex);
```

```
            // Για να διαβαστεί το πεδίο τηλέφωνο εφόσον έχει πρέπει να γίνει ερώτημα στο δεύτερο
             //πίνακα CommonDataKinds. Άρα πρέπει να δημιουργηθεί νέος Cursor που θα παραλάβει
            // τα αποτελέσματα και νέος ContentResolver αν και θα μπορούσαμε να χρησιμοποιήσουμε
           //και το προηγούμενο, για να κάνουμε το ερώτημα στον Provider που περιέχει τα δεδομένα.

          String fieldPhone="";
          if(fieldHasPhone.equals("1")) {          // ελεγχος αν εχει αριθμο τηλεφωνου

             //δημιουργουμε Cursor και Resolver που κανει το ερωτημα
             ContentResolver conResP = getContentResolver();
             Cursor cursP =conResP.query(
                        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                         null,
                         ContactsContract.CommonDataKinds.Phone.CONTACT_ID
                        + "=" + contactID,
                        null, null);

            // Με τον ιδιο πρωτο όπως παραπανω διαβάζουμε το πεδίο NUMBER
             if(cursP.moveToNext()){
              colIndex= cursP.getColumnIndex(
                        ContactsContract.CommonDataKinds.Phone.NUMBER);
              fieldPhone= cursP.getString(colIndex);
             }
          }
            // Προσθέτουμε σε κάθε επανάληψη  στη λίστα contacts το όνομα και τον αριθμό
           //τηλεφώνου.
            contacts.add(fieldName + " " +fieldPhone);

             } while (curs.moveToNext());  // Συνεχίζεται μέχρι η μέθοδος .moveToNext
                                           // που μετακινεί το δείκτη στην επόμενη
                                           // εγγραφή επιστρέψει false
    }

// Δημιουργούμε ένα που διαχειρίζεται τα δεδομένα που πρόκειται να εμφανιστούν στην ListView
          ArrayAdapter<String> adapter = new ArrayAdapter<String>(
                        this, // Ποιο Activity είναι το περιεχόμενο (το τρέχον)
                        R.layout.activity_example_cont_provider, // Ποιο Layout λίστας θα
                                                                 // χρησιμοποιηθεί
                        contacts);       // ποια λίστα θα συνδεθεί με το listView

          setListAdapter(adapter); // Σύνδεσε το adapter του listView με τον adapter
                                   // που δημιουργήσαμε παραπάνω
    }

}
```

10.3 Δέκτες εκπομπών συστήματος και εφαρμογών

Το Android είναι μια πολύ ενεργητική πλατφόρμα. Συχνά εκπέμπει διαφόρων ειδών πληροφορία που είναι δυνατόν να αξιοποιήσουν οι ενεργές εφαρμογές ή να εκκινήσουν δραστηριότητες. Οι πληροφορίες αυτές μπορεί να αφορούν στην κατάσταση του υλικού ενημερώνοντας για παράδειγμα ότι έσβησε η οθόνη, για το επίπεδο της μπαταρίας, για τη σύνδεση σε φορτιστή και ούτω καθεξής.

Αντίστοιχες εκπομπές μπορούν να γίνουν και από εφαρμογές ή εσωτερικά ανάμεσα σε συνιστώσες μιας εφαρμογής. Η δεύτερη περίπτωση είναι ιδιαίτερα ενδιαφέρουσα από πλευράς ασφάλειας των δεδομένων αφού τα κρατά εντελώς μέσα στο πλαίσιο της συγκεκριμένης εφαρμογής και δεν επιτρέπει διαρροές. Αυτό γίνεται αξιοποιώντας το αντικείμενο LocalBroadcastManager που βρίσκεται στη θέση .content.

Μπορούμε να ενεργοποιήσουμε έναν δέκτη εκπομπών με δύο τρόπους: είτε δυναμικά χρησιμοποιώντας την Context.registerReceiver μέσα στον κώδικα java της εφαρμογής είτε στατικά μέσα από την ετικέτα <receiver> στον κώδικα XML του δηλωτικού, όπως και για κάθε άλλη συνιστώσα. Το σημείο στο οποίο θα εγγράψουμε έναν δέκτη είναι πολύ σημαντικό: αν τον εγγράψουμε μέσα στη μέθοδο Activity.onResume για να τον αποεγγράψουμε θα πρέπει να το κάνουμε από την Activity.onPause μέθοδο χρησιμοποιώντας την Context.unregisterReceiver.

Για τις εξωτερικές της εφαρμογής εκπομπές η διαχείριση είναι παρόμοια και δεν εξαρτάται από το ποιος τις εκπέμπει (το σύστημα ή μια εφαρμογή). Το Android πάντως δομεί τις εκπομπές του ως προθέσεις (Intents) οπότε μπορούν να χρησιμοποιηθούν για να εκκινήσουν μια δραστηριότητα.

Μέσα στον ορισμό ενός δέκτη εκπομπών, μπορούμε να έχουμε ένα φίλτρο προθέσεων (intent filter) που προσδιορίζει ποιες ενέργειες μπορεί να δεχτεί. Ένα παράδειγμα:

```
<receiver android:name=".YourReceiver">
<intent-filter>
<action android:name="android.intent.action.BATTERY_LOW" />
</intent-filter>
</receiver>
```

Ο δέκτης του παραδείγματος μπορεί να δεχτεί μόνο την πρόθεση battery _low. Σε κάποιες περιπτώσεις ανακοινώσεων του συστήματος η εγγραφή πρέπει να γίνει υποχρεωτικά δυναμικά στον κώδικα java. Για παράδειγμα αυτό ισχύει με την ενέργεια battery_changed.

Περισσότερα για τους δέκτες εκπομπών στη θέση:

http://developer.android.com/reference/android/content/BroadcastReceiver.html .

10.4 Άλλες κλάσεις

Οι τέσσερις βασικοί τύποι συνιστωσών εφαρμογών Android έχουν σχεδιαστεί ώστε να επιτρέπουν την αλληλεπίδραση μεταξύ εφαρμογών. Όπως οι ανακοινώσεις που εκπέμπονται μπορούν να είναι διαθέσιμες σε κάθε εφαρμογή στο σύστημα, το Android προβλέπει τη δυνατότητα υλοποίησης μέσω τυποποιημένων ενεργειών (actions) κάποιων συνήθων εργασιών (tasks) όπως η κλήση ενός τηλεφωνικού αριθμού.

Έχουμε ήδη αναφέρει τη δυνατότητα χρήσης λειτουργικότητας ανεπτυγμένης από άλλους δημιουργούς στα πλαίσια των εφαρμογών τους μειώνοντας σημαντικά τον απαραίτητο κώδικα που απαιτείται να αναπτυχθεί για μια νέα εφαρμογή, πέρα από τα πρωτότυπα στοιχεία της. Μπορεί να εκκινηθεί μια δραστηριότητα από άλλη εφαρμογή μέσω μίας πρόθεσης και εφόσον η καλούσα

εφαρμογή αναμένει κάποια απάντηση η δραστηριότητα που κλήθηκε δεν διακόπτεται ακόμα και αν η μητρική της εφαρμογή έχει διακοπεί.

Υπάρχουν και άλλοι τύποι συνιστωσών διαθέσιμοι οι οποίοι επίσης έχουν σκοπό την επαναχρησιμοποίηση κώδικα. Τα ποιο χαρακτηριστικά παραδείγματα είναι τα λογικά τμήματα και η μπάρα ενεργειών.

10.5 Λογικά τμήματα (fragments)

Μέχρι στιγμής ορίσαμε τη διεπαφή χρήστη για κάθε οθόνη μας μέσα από μια δραστηριότητα και μία διάταξη (Layout). Είναι πιο πρακτικό να διαιρέσουμε τη διεπαφή χρήστη σε λογικά τμήματα και έτσι να μπορούμε να τα χρησιμοποιήσουμε σε περισσότερες από μία οθόνες της ίδιας εφαρμογής. Αυτό αφενός επιτρέπει οικονομία χρόνου κατά τη σχεδίαση αλλά και ευκολότερη μεταφορά αλλαγών και τροποποιήσεων στο σύνολο της εφαρμογής.

Ένα λογικό τμήμα (fragment) αντιπροσωπεύει μια συμπεριφορά ή ένα τμήμα της διεπαφής χρήστη στα πλαίσια μιας δραστηριότητας. Η δυνατότητα σύνθεσης της διεπαφής χρήστη μιας δραστηριότητας από πολλαπλά λογικά τμήματα, πέρα από τη δυνατότητα επαναχρησιμοποίησης των τμημάτων, επιτρέπει ακόμα την αντιμετώπισή τους ως σχετικά αυτόνομα δομικά στοιχεία με δικά τους συμβάντα εισόδου και τα οποία μπορούν να ενεργοποιούνται και απενεργοποιούνται ενόσω η δραστηριότητα είναι ενεργή.

Το λογικό τμήμα πρέπει πάντα να ενσωματώνεται σε μια δραστηριότητα και ο κύκλος ζωής του επηρεάζεται άμεσα από τον κύκλο ζωής της δραστηριότητας που το φιλοξενεί. Έτσι αν η δραστηριότητα είναι αδρανής ή έχει διακοπεί, το ίδιο συμβαίνει για κάθε λογικό της τμήμα. Όμως όσο ο ξενιστής είναι ενεργός το κάθε λογικό τμήμα είναι αυτόνομα διαχειρίσιμο. Τις αυτόνομες συναλλαγές (transactions) κάθε λογικού τμήματος συνηθίζουμε να αποθηκεύουμε σε μια στοίβα επιστροφής (back stack) την οποία διαχειρίζεται ο ξενιστής, επιτρέποντας έτσι στο χρήστη την επιστροφή σε προηγούμενη κατάσταση, με χρήση για παράδειγμα ενός πλήκτρου επιστροφής (back) στην πλοήγηση.

Για περισσότερες λεπτομέρειες:
http://developer.android.com/guide/components/fragments.html

10.6 Action Bar

Η μπάρα ενεργειών αποτελεί επίσης σημαντική εξυπηρέτηση για τη διεπαφή χρήστη. Αυτή η συνιστώσα της διεπαφής χρήστη είναι συνεπής και σταθερή για όλα τα τμήματα του λειτουργικού Android λειτουργώντας έτσι με οικείο τρόπο για τους χρήστες της πλατφόρμας. Περιλαμβάνει συνήθως συντομεύσεις για συνήθεις ενέργειες, περιλαμβανομένης της πλοήγησης μεταξύ τμημάτων της εφαρμογής. Μπορεί ακόμα να περιλαμβάνει μια ένδειξη θέσης μέσα στην εφαρμογή.

Για την εισαγωγή της action bar σε μια δραστηριότητα, η αντίστοιχη κλάση πρέπει να περιλαμβάνει μια δήλωση επέκτασης (extends) της κλάσης ActionBarActivity και να εφαρμόσει ένα θέμα AppCompat ως χαρακτηριστικό στη δραστηριότητα μέσα στο δηλωτικό. Περισσότερα σχετικά στη θέση: http://developer.android.com/guide/topics/ui/actionbar.html .

Κεφάλαιο 11

Χρησιμοποιώντας δείγματα (samples)

11. Χρησιμοποιώντας δείγματα (samples)

Τα δείγματα εφαρμογών του Android αποτελούν χρήσιμα παραδείγματα των διαθέσιμων λειτουργιών στα πλαίσια ενός συγκεκριμένου API. Αποτελούν όμως και πολύτιμες πηγές επαναχρησιμοποιήσιμου κώδικα για τα πιο τυποποιημένα στοιχεία μιας εφαρμογής.

Στόχοι

Οι μαθητές να μπορούν να:

- Γνωρίσουν την ύπαρξη των samples
- Αναγνωρίζουν τη χρησιμότητά τους για την κατανόηση επιμέρους θεμάτων προγραμματισμού για Android
- Χρησιμοποιούν τα samples για την επέκταση της κατανόησής τους

Ενότητες Κεφαλαίου

- Τι είναι τα δείγματα
- Εκπαιδευτικές χρήσεις
- Εγκατάσταση samples
- Δημιουργία έργων Sample
- Τρόποι χρήσης των samples

11.1 Τι είναι τα δείγματα

Τα δείγματα (samples) είναι πλήρως λειτουργικές Android εφαρμογές οι οποίες επιτρέπουν την εξοικείωση με την κατασκευή διάφορων συνιστωσών εφαρμογών Android. Για το σκοπό αυτό επιτρέπουν, τουλάχιστον μέχρι τη στιγμή που γράφονται αυτές οι γραμμές, την πρόσβαση στους πόρους, στα αρχεία πηγαίου κώδικα ενώ μπορεί κανείς να έχει εικόνα και για τη γενική δομή της εφαρμογής. Κάποια από τα δείγματα επιδεικνύουν λειτουργικότητες συγκεκριμένων APIs.

Η λογική τους είναι λογική παραδείγματος αλλά διαθέτουν τον υποδειγματικό κώδικα στο πλαίσιο μιας πλήρως λειτουργικής εφαρμογής. Επιτρέπουν την αντιγραφή τμημάτων κώδικα αλλά και την εξοικείωση με την οργάνωση ενός έργου. Η παρατήρηση του τρόπου με τον οποίο διαφορετικά APIs αξιοποιούνται κατά τη διάρκεια ανάπτυξης μιας εφαρμογής βοηθά στην κατανόηση τόσο της λειτουργίας των εφαρμογών στο Android όσο και στην αντίληψη της εξέλιξης του λειτουργικού αυτού.

Το Android SDK περιλαμβάνει αρκετά δείγματα για τη διευκόλυνση της εκμάθησης του Android και των εκδόσεών του. Τα δείγματα αυτά είναι οργανωμένα με βάση το API για το οποίο έχουν κατασκευαστεί ενώ η πρόσβαση σε αυτά γίνεται μέσω του Android SDK manager. Προφανώς έτοιμα έργα μπορούν να βρεθούν και σε άλλες πηγές όπως στις θέσεις:

http://www.java2s.com/Code/Android/CatalogAndroid.htm και

https://code.google.com/p/apps-for-android/source/browse/#git .

11.2 Εκπαιδευτικές χρήσεις

Αν δεν γνωρίζετε με ποιο τρόπο μπορείτε να υλοποιήσετε συγκεκριμένες λειτουργικότητες σε κάποια έκδοση του Android ένας απλός τρόπος για να ξεπεράσετε το πρόβλημα είναι να βρείτε ένα δείγμα που τις χρησιμοποιεί και να ανοίξετε το αντίστοιχο έργο στο Eclipse. Μετά από αυτό μια επίσκεψη στα αντίστοιχα αρχεία θα σας δείξει με ποιο τρόπο έχουν αυτές υλοποιηθεί και σε τι διαφέρουν από την έκδοση που έχετε γνωρίσει στα προηγούμενα κεφάλαια.

Σημαντικό εδώ είναι να θυμόμαστε ότι όχι μόνο το Android αλλά και το Android SDK έχει διαφορετικές εκδόσεις. Αν σας ενδιαφέρει μια πολύ παλιά έκδοση του Android (π.χ. η 1.5) αυτή μπορεί να αντιστοιχεί σε ένα API level πολύ παλιό για την έκδοση του SDK σας (στην περίπτωση του Android 1.5 αυτό είναι το επίπεδο 3). Αυτό σημαίνει ότι κάποιες λειτουργικότητες της έκδοσης αυτής δεν θα «τρέχουν» στο περιβάλλον μας. Μπορείτε να δείτε τον κώδικα ενός κατάλληλου δείγματος, αν και με πολλά μηνύματα για ξεπερασμένο κώδικα (deprecated code). Όμως αν θέλετε να «τρέξετε» την εφαρμογή, ή πολύ περισσότερο να υλοποιήσετε μια εφαρμογή για ένα «ξεπερασμένο» API θα πρέπει να ξανασετάρετε το Eclipse χρησιμοποιώντας κατάλληλη παλιότερη έκδοση του SDK.

Μια περισσότερο ενδιαφέρουσα περίπτωση έχει να κάνει με την εξοικείωση με τις διαδικασίες ελέγχου που συναντήσαμε στο κεφάλαιο 8. Κάποια από τα δειγματικά έργα που διαθέτει το SDK τελειώνουν σε «>testes». Αυτά τα έργα είναι έργα ελέγχου του JUnit τα οποία αντιστοιχίζονται σε ένα άλλο δειγματικό έργο του οποίου αποτελούν ελέγχους. Η σύγκριση των δύο έργων μπορεί να βοηθήσει πολύ στην κατανόηση του τρόπου που κατασκευάζονται έργα ελέγχου.

Δραστηριότητα 11.2

Αφού εγκαταστήσετε τα διαθέσιμα samples χρησιμοποιώντας τις οδηγίες της επόμενης ενότητας επιλέξτε ένα έργο και το αντίστοιχο έργο ελέγχου του από τη λίστα για το API στόχο της εφαρμογής σας. Αφού μελετήσετε το κύριο έργο και το έργο ελέγχου του και διαπιστώσετε τις διαφορές τους επιστρέψτε στην εφαρμογή σας. Αν έχετε ήδη χτίσει ένα έργο ελέγχου με βάση όσα

μάθατε στο κεφάλαιο 8 ελέγξτε την πληρότητα των ελέγχων σας και βελτιώστε κατάλληλα το έργο ελέγχου σας. Αν δεν έχετε δομήσει ένα έργο ελέγχου για την εφαρμογή σας, τώρα είναι η στιγμή να το κάνετε.

11.3 Εγκατάσταση samples

Όπως αναφέραμε και νωρίτερα τα δείγματα του Android SDK μπορούν να προσπελαστούν από τον Android SDK manager. Μέσα στον manager επιλέγουμε Samples for SDK και εγκαθιστούμε. Η επιλογή αυτή θα κατεβάσει (download) τα δείγματα σε έναν υποκατάλογο στην εγκατάσταση του SDK.

Στο Eclipse επιλέξτε διαδοχικά File → New → Other, αναπτύξτε το φάκελο Android και επιλέξτε Android Sample Project . Αυτές οι ενέργειες θα εκκινήσουν τον βοηθό Sample Project.

Στο περιβάλλον του βοηθού προβάλλεται μια λίστα από πιθανά API στόχους από την οποία μπορείτε να επιλέξετε ένα. Θυμηθείτε τα όσα αναφέραμε σχετικά με την προς τα πίσω συμβατότητα.

Έχοντας επιλέξει API, η επόμενη επιλογή είναι αυτή του δειγματικού έργου. Μπορείτε τώρα να δώσετε ένα διαφορετικό όνομα για να μην υπάρξει σύγχυση αν μετά ανοίξετε ένα με ίδιο όνομα από το φάκελο ενός άλλου API.

Το έργο που δημιουργήσατε περιλαμβάνει το δείγμα που επιλέξατε και είναι ανοιχτό στο Eclipse. Επομένως μπορείτε να δείτε τα αρχεία, τα πακέτα και τη δομή της εφαρμογής.

11.4 Δημιουργία έργων Sample

Ένα έργο που φτιάχτηκε από ένα δείγμα έχει ένα σημαντικό πλεονέκτημα: κατά κανόνα θα μπορεί να μεταφραστεί και να «τρέξει» άμεσα γιατί πρόκειται για μια πλήρως λειτουργική εφαρμογή. Αυτό σημαίνει ότι μπορούμε, μέσα από το Eclipse να το εκτελέσουμε είτε σε μια εικονική είτε σε μια συνδεδεμένη και ενεργή φυσική συσκευή.

Δραστηριότητα 11.4.1

Αφού εκκινήσετε τον προσομοιωτή από τις ρυθμίσεις εκτέλεσης (Run →Run Configurations) επιλέξτε Android Application και αναζητείστε (browse) στην καρτέλα επιλογής έργου (Project Selection) το δειγματικό έργο σας. Επιλέξτε το και προχωρήστε. Αφού δώσετε ένα όνομα και επιλέξετε να ρωτάει πάντα για ποια συσκευή (Always prompt to pick device). Από τον επιλογέα συσκευής επιλέξτε μια κατάλληλα για το API Του δείγματος εικονική συσκευή και δοκιμάστε την εφαρμογή του δείγματος.

Δραστηριότητα 11.4.2

Αφού συνδέσετε μια φυσική συσκευή σας (κινητό ή τάμπλετ) στο σύστημα που «τρέχει» το Eclipse και ελέγξετε τη συμβατότητα του λειτουργικού της με το API της εφαρμογής επιλέξετε την στον επιλογέα και δοκιμάστε και σε αυτήν την εφαρμογή του δείγματος.

11.5 Τρόποι χρήσης των samples

Ένας τρόπος χρήσης των δειγμάτων είναι μέσα από ένα περιβάλλον ανάπτυξης. Τον τρόπο αυτό αναπτύξαμε για το Eclipse. Είναι ενδιαφέρον να τον εφαρμόσουμε για ένα δείγμα API Demos. ΤΑ δείγματα αυτά περιέχουν πλούτο κώδικα που επιδεικνύει λειτουργίες που αφορούν στα γραφικά, animation, άλλα μέσα αλλά και λειτουργίες ασφάλειας και προσβάσεων για το

συγκεκριμένο API στο οποίο απευθύνεται η εφαρμογή. Η εκτέλεση της εφαρμογής προσφέρει περαιτέρω κατανόηση.

Όμως όπως έχουμε ήδη αναφέρει τα δείγματα υπάρχουν πλέον σε υποκατάλογο του καταλόγου SDK (που με τη σειρά του βρίσκεται μέσα στον κατάλογο ADT Bundle). Ο υποκατάλογος ονομάζεται samples και περιλαμβάνει με τη σειρά του υποκαταλόγους για κάθε κατηγορία (API) δειγμάτων. Μπορείτε επομένως να έχετε απευθείας πρόσβαση στον αρχεία πηγαίου κώδικα της κάθε εφαρμογής/ δείγματος για να τα μελετήσετε, να εκκινήσετε από εδώ μια εφαρμογή ή να χρησιμοποιήσετε τμήμα του κώδικα. Η χρησιμοποίηση λογικών τμημάτων της διεπαφής χρήστη από ένα δείγμα κατά την ανάπτυξη εφαρμογής είναι συνηθισμένη πρακτική που επιτρέπει την επικέντρωση στον πηγαίο κώδικα της ίδιας της εφαρμογής. Όμως κατάχρηση αυτής της δυνατότητας οδηγεί συχνά σε παρόμοιες εμφανισιακά και σε κάποιες περιπτώσεις παρόμοιας λειτουργικότητας εφαρμογές πράγμα αθέμιτο.

Κεφάλαιο 12

Δημοσίευση και κυκλοφορία έργου

12. Δημοσίευση και κυκλοφορία έργου

Αν υποθέσουμε ότι μια εφαρμογή έχει περάσει με επιτυχία τις φάσεις του ελέγχου και της εκσφαλμάτωσης είναι η ώρα η εφαρμογή αυτή να δημοσιευτεί. Η δημοσίευση αυτή μπορεί να γίνει με πολλούς τρόπους αλλά ο πλέον διαδεδομένος είναι η ανάρτηση της εφαρμογής μέσα από το Google Play. Άλλοι τρόποι μπορεί να είναι απλούστεροι και να έχουν λιγότερες απαιτήσεις, όμως γενικά είναι καλό να εφαρμόζονται καλές πρακτικές κατά τη διάρκεια οποιασδήποτε διαδικασίας δημοσίευσης.

Στόχοι

Οι μαθητές να μπορούν να:

- Διαχωρίζουν μεταξύ μιας έκδοσης ανάπτυξης και μιας έκδοσης διανομής
- Περιγράφουν την προετοιμασία της δημοσίευσης
- Αναγνωρίζουν την ανάγκη πολλαπλών εκδόσεων
- Περιγράφουν και υλοποιούν τα βήματα δημοσίευσης μιας εφαρμογής

Ενότητες Κεφαλαίου

- Προετοιμασία
- Εκδόσεις (Versioning)
- Υπογραφή
- Δημοσίευση

12.1 Προετοιμασία

Πέρα από την ολοκλήρωση της εκσφαλμάτωσης, πριν τη δημοσίευση μιας εφαρμογής καλό είναι να απομακρυνθούν τα ίχνη ελέγχου που μπορεί να υπάρχουν. Για παράδειγμα, αν στον κώδικα υπάρχουν ακόμα δηλώσεις καταχώρησης (log statements) ή άλλες κλήσεις για εξόδους μηνυμάτων εκσφαλμάτωσης αυτές πρέπει να απομακρυνθούν πριν την κυκλοφορία.

Αλλαγές πρέπει να γίνουν και στο δηλωτικό. Πέρα από τα χαρακτηριστικά της έκδοσης (version) που πρέπει να επικαιροποιηθούν απαιτείται και η απομάκρυνση πριν την κυκλοφορία της ρύθμισης χαρακτηριστικών android:debuggable .

Μεγάλη σημασία για μια δημοσιευμένη εφαρμογή έχει να έχουν περιληφθεί όλοι οι απαραίτητοι πόροι στους υποκαταλόγους του πακέτου. Ειδική φροντίδα θα πρέπει να δοθεί για τα πολυμεσικά στοιχεία, όπως τα σχεδιάσιμα για κάθε υποστηριζόμενη ρύθμιση/ τύπο συσκευής. Επιπλέον αν χρησιμοποιούνται εξωτερικοί πόροι όπως βάσεις δεδομένων θα πρέπει να επιβεβαιωθεί ότι και αυτοί είναι κατάλληλα προετοιμασμένοι για χρήση.

Αν η εφαρμογή χρησιμοποιεί απομακρυσμένους εξυπηρετητές ή υπηρεσίες πρέπει να επιβεβαιωθεί ότι είναι έτοιμοι για «εμπορική» λειτουργία. Επίσης είναι σημαντικό να είναι ασφαλείς ώστε να μην κινδυνεύει εξ αιτίας τους η ασφάλεια της εφαρμογής. Αν χρησιμοποιείτε υπηρεσίες όπως τα Google Maps θα χρειαστείτε ένα ειδικό κλειδί δημοσίευσης για Maps API ώστε να μπορείτε να χρησιμοποιήσετε τη σχετική εξωτερική βιβλιοθήκη.

Το δηλωτικό χρειάζεται επίσης προετοιμασία. Εάν υπάρχουν προσβάσεις (permissions) που απαιτεί η εφαρμογή σας, αυτές πρέπει να έχουν καταλογογραφηθεί στο δηλωτικό με κατάλληλα στοιχεία uses-permission. Οι ιδιότητες icon και label του στοιχείου εφαρμογής (application) πρέπει επίσης να έχουν οριστεί κατάλληλα. Γενικά στοιχεία που εμφανίζονται στο δηλωτικό ρυθμίζουν λεπτομέρειες της εγγραφής της εφαρμογής στο Google Play Store.

Τέλος ίσως είναι σκόπιμο να προετοιμάσετε μια συμφωνία χρήσης για τον τελικό χρήστη (End User License Agreement –EULA) που θα καθορίζει τους όρους χρήσης, θα διασφαλίζει τα πνευματικά δικαιώματα και θα καθορίζει το βαθμό ανάληψης ευθύνης για την καλή λειτουργία της εφαρμογής σας. Φυσικά οι σχετικοί περιορισμοί δεν μπορούν να ξεπεράσουν κάποια όρια που βάζει ο νόμος σε κάθε χώρα.

Για περισσότερα σχετικά με την προετοιμασία για δημοσίευση δες στη θέση: https://developer.android.com/tools/publishing/preparing.html.

12.2 Εκδόσεις (Versioning)

Έχουμε, σε προηγούμενο κεφάλαιο, ήδη αναφέρει πώς δίνουμε στην εφαρμογή μας έναν αριθμό (Code) και όνομα (Name) έκδοσης. Αυτά περιλαμβάνονται σαν χαρακτηριστικά του ίδιου του δηλωτικού στο ριζικό στοιχείο του δηλωτικού (manifest) με τη μορφή android:versionCode και android:versionName αντίστοιχα.

Το χαρακτηριστικό versionCode επιβάλλεται να είναι ένας ακέραιος, μεγαλύτερος από όλους τους προηγούμενους για κάθε επόμενη έκδοση της εφαρμογής. Παρότι λογικά πρέπει να ξεκινήσετε με τον αριθμό 1, πράγμα που κάνει αυτόματα το Eclipse όταν δημιουργεί ένα έργο, και να αυξάνετε με κάποιο βήμα κάθε φορά που εκδίδετε μια νεότερη έκδοση ή μια ανανέωση (update), στην πράξη ο μόνος έλεγχος που γίνεται έχει σχέση με το αν ο αριθμός μιας νεότερης έκδοσης είναι μεγαλύτερος από μιας παλιότερης. Η τιμή αυτού του χαρακτηριστικού δεν είναι

ορατή στους τελικούς χρήστες και χρησιμοποιείται αποκλειστικά κατά τη διάρκεια της διαδικασίας δημοσίευσης για να προσδιορίσει αν η έκδοση είναι νεότερη από την ως τώρα εγκατεστημένη.

Το χαρακτηριστικό versionName αντίθετα είναι μια συμβολοσειρά ορατή στους τελικούς χρήστες. Δεν υπάρχει υποχρέωση αντιστοίχισης στον κωδικό της έκδοσης ή τήρησης ενός συγκεκριμένου τύπου. Όμως υπάρχει ένα προηγούμενο, και οι χρήστες το περιμένουν λίγο ως πολύ, το string να αποτελείται από δύο ή περισσότερους ακέραιους χωριζόμενους από τελείες, με τον πρώτο να ξεκινά από το ένα και τον δεύτερο από μηδέν και η αύξηση να γίνεται για ανανεώσεις και μικρομεταβολές στον τελευταίο αριθμό ανά ένα, ενώ μια μεγάλη αλλαγή ή ανασχεδίαση υποδηλώνεται με αύξηση το πρώτου αριθμού κατά ένα και μηδενισμό των επόμενων. Προφανώς μπορείτε να υιοθετήσετε κάποια άλλη σύμβαση με την οποία οι χρήστες σας θα εξοικειωθούν στην πορεία, καλό όμως είναι να έχετε καταλήξει σε αυτήν προκαταβολικά.

12.3 Υπογραφή

Για την εγκατάσταση μιας εφαρμογής σε ένα σύστημα Android, μια εφαρμογή πρέπει να υπογραφεί με ένα πιστοποιητικό χρησιμοποιώντας ένα δημόσιο κλειδί. Παρότι κατά την εκσφαλμάτωση το Android μας απαλλάσσει από τη διαδικασία, υπογράφοντας αυτόματα την εφαρμογή με ένα κλειδί κατάλληλο για εκσφαλμάτωση, οι εκδόσεις για εκσφαλμάτωση και δημοσίευση είναι διαφορετικές. Έτσι οι τελευταίες απαιτούν την υπογραφή μέσω του ιδιωτικού κλειδιού του δημιουργού της εφαρμογής.

Το Java Development Kit (JDK) προσφέρει μια δυνατότητα διαχείρισης και δημιουργίας κλειδιών μέσω του keytool. Δημιουργείται σε αυτό μια θέση φύλαξης (keystore) για το ιδιωτικό κλειδί για το οποίο διαλέγουμε ένα αναγνωριστικό (alias name) και ένα κωδικό (password), με τους οποίους μπορεί να προσπελαστεί το κλειδί για να υπογραφεί η εφαρμογή.

Δραστηριότητα 12.3.1

Χρησιμοποιείστε το keytool του JDK για να αποχτήσετε ένα ιδιωτικό κλειδί, το αναγνωριστικό και τον κωδικό του. Αν χρειάζεστε περισσότερες πληροφορίες επισκεφτείτε τη θέση http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html.

Δραστηριότητα 12.3.2

Σκοπός αυτής της δραστηριότητας είναι να δημιουργήσετε μια έκδοση για δημοσίευση της εφαρμογής σας . Πρέπει πρώτα να έχετε ολοκληρώσει την προηγούμενη δραστηριότητα.

Στο εξής, για κάθε νέα έκδοση του πακέτου θα χρειάζεστε το ίδιο κλειδί δημοσίευσης οπότε μην το χάσετε.

12.4 Δημοσίευση

Η διαδικασία δημοσίευσης ξεκινά μόλις έχουμε στη διάθεσή μας το APK αρχείο μιας έκδοσης για δημοσίευση. Δεν αρκεί όμως αυτό: στην σχετική ανάρτηση περιλαμβάνονται στοιχεία όπως διαφημιστικές εικόνες, περιγραφή της εφαρμογής κλπ. τα οποία πρέπει να είναι διαθέσιμα πριν ξεκινήσει η διαδικασία.

Η δημοσίευση μπορεί να γίνει σε δική σας ιστοσελίδα, στην αγορά της Google ή σε κάποια εναλλακτική αγορά. Σε όλες τις περιπτώσεις καλό είναι να προσφέρονται οι πληροφορίες για την εφαρμογή με τρόπο σαφή και εύκολα αναγνωρίσιμο. Επίσης σκόπιμο είναι η εφαρμογή να έχει υπογραφεί πριν από τη διάθεση. Μπορείτε επίσης να διαθέσετε την εφαρμογή σας μέσω email. Στη συνέχεια θα περιγράψουμε τη διαδικασία για δημοσίευση στην αγορά της Google.

Κατά την διαδικασία θα χρειαστεί να ρυθμιστούν και κάποιες ακόμα επιλογές για την εγγραφή στο κατάστημα. Αυτές αφορούν στην τιμολόγηση, ρυθμίσεις γλώσσας κ.ά. Μπορεί κανείς να δει αναλυτικά τι χρειάζεται για την διαδικασία στη θέση:

http://developer.android.com/distribute/tools/launch-checklist.html.

Η διαδικασία ξεκινά από έναν λογαριασμό Google και με επίσκεψη στην Developer Console. Με την επιλογή Add New Application ανοίγει ένα αναδυόμενο μενού όπου μπορείτε να επιλέξετε την προεπιλεγμένη γλώσσα της εφαρμογής σας και να εισάγετε το όνομά της.

Μπορείτε πλέον να αρχίσετε να επεξεργάζεστε την εγγραφή για την εφαρμογή σας και να ανεβάσετε το αρχείο APK. Η διαδικασία είναι απαιτητική, λεπτομερής και χρονοβόρα καθώς μπορείτε να περιλάβετε εικόνες, screenshots και video, ενώ είναι αναγκαίο να περιλάβετε κάποια περιγραφή της εφαρμογής, να την κατατάξετε σε κατηγορίες, να αναρτήσετε αξιολογήσεις περιεχομένου και σκόπιμο να αναρτήσετε στοιχεία επικοινωνίας μαζί σας.

Σε κάθε περίπτωση προχωρήστε πρώτα σε μια ανάρτηση της εφαρμογής σας σε draft (πρόχειρη και όχι δημοσιευμένη) εγγραφή. Αυτό θα σας επιτρέψει να ελέγξετε πως βλέπει ο χρήστης την εγγραφή, να βεβαιωθείτε ότι δεν παραλείψατε κάτι και πως οι ρυθμίσεις που κάνατε ήταν αυτές που θέλατε. Αν όλα έχουν πάει καλά αρκεί να επιλέξετε δημοσίευση (Publish) για να ολοκληρώσετε.

Αργότερα χρειάζεται να οριστεί τιμή για το προϊόν ή να διατεθεί δωρεάν. Μια δωρεάν εφαρμογή δεν μπορεί να μεταβληθεί σε πληρωνόμενη. Μπορεί όμως να τροποποιηθεί μια πληρωνόμενη ως προς την τιμή ή ακόμα και να μετατραπεί σε δωρεάν. Μια άλλη επιλογή είναι οι χρεώσεις μέσα στην εφαρμογή (in-app billing) ενώ υπάρχουν και άλλοι τρόποι (όπως π.χ. application licensing) για να δημιουργήσετε έσοδα, τους οποίους θα πρέπει να ψάξετε λίγο μόνοι σας.

Πριν προσπαθήσετε όμως να χρεώσετε κάτι για την εφαρμογή σας ή και απλά να την δημοσιεύσετε στο Google Store θα πρέπει να ενημερωθείτε λίγο πάνω σε θέματα πνευματικών δικαιωμάτων. Η επαναχρησιμοποίηση κώδικα έχει τα όριά της ενώ για το πολυμεσικό υλικό και τα κείμενα υπάρχουν συγκεκριμένοι κανόνες ως προς το τι μπορεί να χρησιμοποιηθεί για ποιο σκοπό.

Δραστηριότητα 12.4

Ενημερωθείτε για το ζήτημα των δικαιωμάτων χρήσης και του ανοιχτού λογισμικού στις παρακάτω θέσεις:

https://foss.ntua.gr/wiki/index.php/Άδειες_Ελεύθερου_Λογισμικού_και_Πατέντες

http://www.eexi.gr/?q=node/16 και http://www.eexi.gr/?q=node/25 στα ελληνικά.

Επίσης στα αγγλικά στη θέση

http://www.gnu.org/licenses .

ΕΝΟΤΗΤΑ 2β

ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ (APPS) ΜΕ APPINVENTORRR

Κεφάλαιο 13

Το προγραμματιστικό περιβάλλον AppInventor2

13. Το προγραμματιστικό περιβάλλον AppInventor2

Εισαγωγή

Η δεύτερη αυτή ενότητα του δεύτερου μέρους είναι προσανατολισμένη στην υλοποίηση. Με όλες τις έννοιες να έχουν αναπτυχθεί στις δύο προηγούμενες ενότητες σκοπός της παρούσας είναι να δοθεί η δυνατότητα με ένα σχετικά εύκολο τρόπο να υλοποιηθεί μία ή περισσότερες άρτιες εφαρμογές. Αυτό μπορεί να συμβεί χρησιμοποιώντας για την υλοποίηση την γλώσσα ψηφίδων του περιβάλλοντος AppInventor.

Οι στόχοι της ενότητας 2β δεν είναι άλλοι από τους στόχους της ενότητας 2α. Επιπλέον αυτών η αξιοποίηση ενός περιβάλλοντος οικείου στο μαθητή όπως το Appinventor και η εφαρμογή όσων διδάχτηκε με εύκολο και ευχάριστο τρόπο, διασφαλίζουν τη μεγαλύτερη αποτελεσματικότητα της ενότητας 2β για την βαθύτερη κατανόηση των απαιτήσεων της ανάπτυξης μιας Android εφαρμογής.

Ενότητες Κεφαλαίου

- Το έργο (project) στο προγραμματιστικό περιβάλλον AppInventor2
- Δομή περιβάλλοντος
- Σχεδίαση και υλοποίηση εφαρμογής (Design/ Block)
- Η χρήση του προσομοιωτή για κινητά και υπολογιστή (companion/emulator)
- Ανάρτηση εφαρμογής

13.1 Το έργο (project) στο προγραμματιστικό περιβάλλον AppInventor2

Ο προγραμματισμός στο AppInventor μπορεί να ξεκινήσει επιλέγοντας το σύνδεσμο ai2.appinventor.mit.edu. Ο σύνδεσμος αυτός ανοίγει τη νεότερη έκδοση του AppInventor η οποία δόθηκε το Δεκέμβριο του 2013 και η οποία αποκαλείται από πολλούς και ως AppInventor 2. Σε αυτές τις σημειώσεις θα χρησιμοποιήσουμε την έκδοση AppInventor 2.

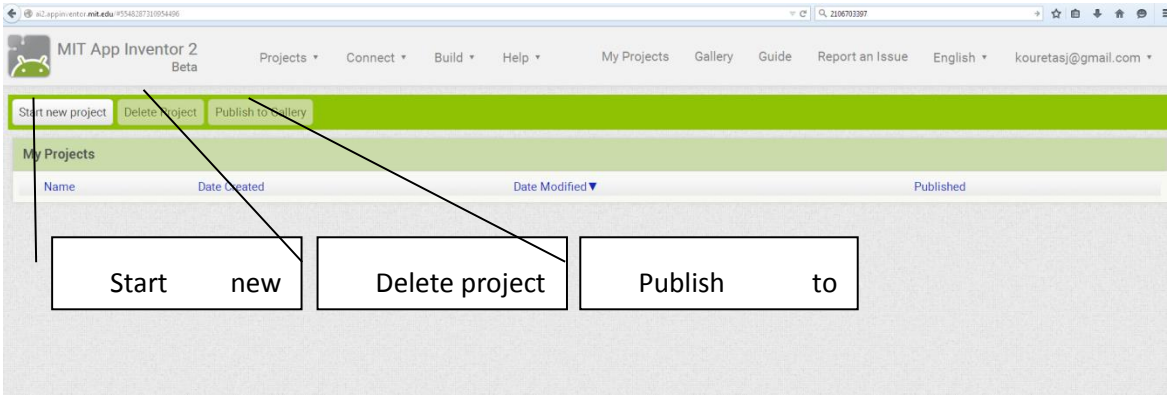
Το περιβάλλον του AppInventor αποτελείται από δύο τμήματα.

- Το περιβάλλον σχεδίασης (ComponentDesigner). Εδώ βρίσκονται τα στοιχεία που απαρτίζουν το γραφικό μέρος της εφαρμογής, κουμπιά, εικόνες κτλ.
- Το περιβάλλον με τα προγραμματιστικά πλακίδια (BlockEditor). Εδώ βρίσκεται το πρόγραμμα της εφαρμογής σε μορφή προγραμματιστικών πλακιδίων, τα οποία δίνουν ενέργεια στα στοιχεία της εφαρμογής.

Επίσης είναι απαραίτητη μια συσκευή με Android όπου θα τρέχει η εφαρμογή. Σε περίπτωση που δεν είναι διαθέσιμη, παρέχεται από το περιβάλλον του Appinventor ένας εξομοιωτής της συσκευής.

Όσα στην προηγούμενη ενότητα 2Α ρύθμιζε το Eclipse χρησιμοποιώντας XML, εδώ θα τα ρυθμίζετε από το παράθυρο properties του Component Designer, ενώ ό,τι θα προγραμματίζατε σε Java εδώ γράφεται με πλακίδια στον Block Editor. Μερικές ρυθμίσεις, όπως τα permissions, ρυθμίζοντας αυτόματα ανάλογα με τα αντικείμενα/ blocks που θα χρησιμοποιήσετε στον κώδικά σας. Περισσότερα θα δούμε στην πράξη.

Ακολουθώντας το σύνδεσμο *ai2.appinventor.mit.edu* οδηγούμαστε στο περιβάλλον του AppInventor όπως φαίνεται στην Εικόνα 13.1.1.

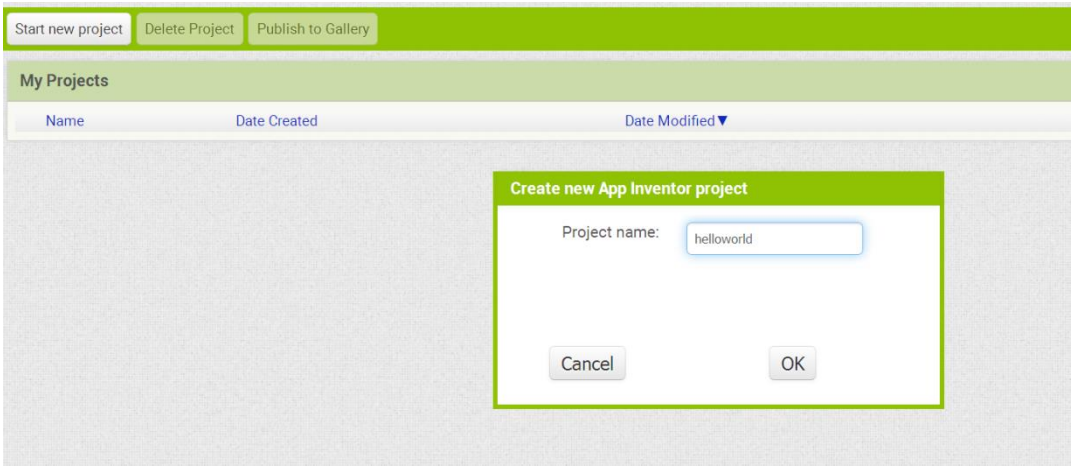


Εικόνα 13.1.1 Το περιβάλλον του AppInventor

Οι τρεις βασικές επιλογές που έχουμε εδώ είναι:

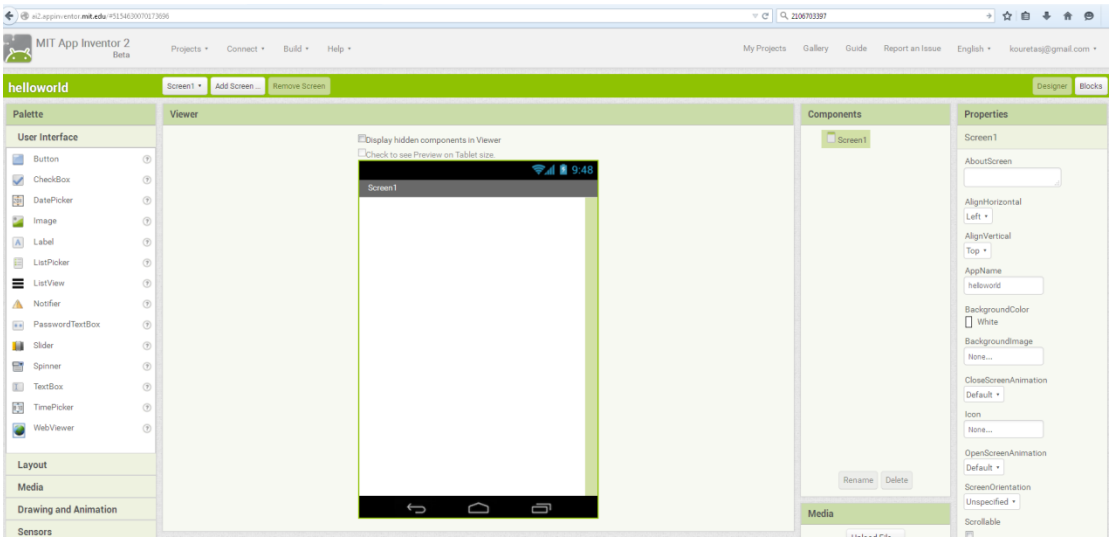
1. Δημιουργία καινούριου έργου (Start new project)
2. Διαγραφή έργου (Delete Project)
3. Δημοσίευση του έργου (Publish to Gallery)

Επιλέγοντας Δημιουργία καινούριου έργου, μας ζητάει να δώσουμε ένα όνομα στο έργο μας όπως φαίνεται στην Εικόνα 13.1.2.



Εικόνα 13.1.2 Δημιουργία νέου έργου.

Στη συνέχεια αφού πληκτρολογήσουμε το όνομα του νέου έργου (π.χ. helloworld) αυτόματα οδηγούμαστε στο περιβάλλον σχεδίασης της εφαρμογής, όπως φαίνεται στην Εικόνα 13.1.3.



Εικόνα 13.1.3 Το περιβάλλον σχεδίασης.

13.2 Δομή περιβάλλοντος

Στο αριστερό μέρος της οθόνης υπάρχει η παλέτα στην οποία βρίσκονται τα στοιχεία που μπορούμε να τοποθετήσουμε πάνω στην οθόνη του κινητού, όπως είναι κουμπιά, κείμενο, σύνδεση με βάση δεδομένων, εικόνα, βίντεο κ.α. Η τοποθέτηση γίνεται με τη μέθοδο σύρε και άφησε (drag and drop).

Τα στοιχεία αυτά βρίσκονται σε κατηγορίες – Basic, Media, Animation, Social, Sensors, κ.λ.π. Πατώντας πάνω σε μια κατηγορία εμφανίζονται τα στοιχεία της κατηγορίας. Υπάρχουν πολλά στοιχεία με τα οποία κάποιος μπορεί να πειραματιστεί.

Στο κέντρο φαίνεται η οθόνη από το κινητό τηλέφωνο και το όνομά της που είναι screen1. Εδώ τοποθετούνται τα στοιχεία με τη μέθοδο σύρε και άφησε.

Στο δεξιό μέρος φαίνονται οι παράμετροι που καθορίζουν κάθε ένα από τα στοιχεία που τοποθετούνται στην οθόνη. Αυτοί οι παράμετροι μπορεί να είναι το χρώμα, το μέγεθος, το κείμενο σε ένα στοιχείο κειμένου κ.λ.π.

13.3 Σχεδίαση και υλοποίηση εφαρμογής (Design/ Block)

13.3.1 Παράδειγμα εφαρμογής 1

Σε αυτό το σημείο θα δείξουμε ένα πολύ απλό παράδειγμα εφαρμογής. Θέλουμε να δημιουργήσουμε μια εφαρμογή στην οποία θα φαίνεται η εικόνα μιας γάτας την οποία μόλις πατήσουμε η γάτα θα νιαουρίζει.

Για τη συγκεκριμένη εφαρμογή θα χρειαστούμε δύο στοιχεία, ένα label το οποίο θα γράφει τον τίτλο της εφαρμογής, π.χ. «Χάιδεψε τη γάτα» και ένα κουμπί (button).

Βήμα 1ο. Δημιουργούμε ένα καινούριο έργο με όνομα Catmiaou.

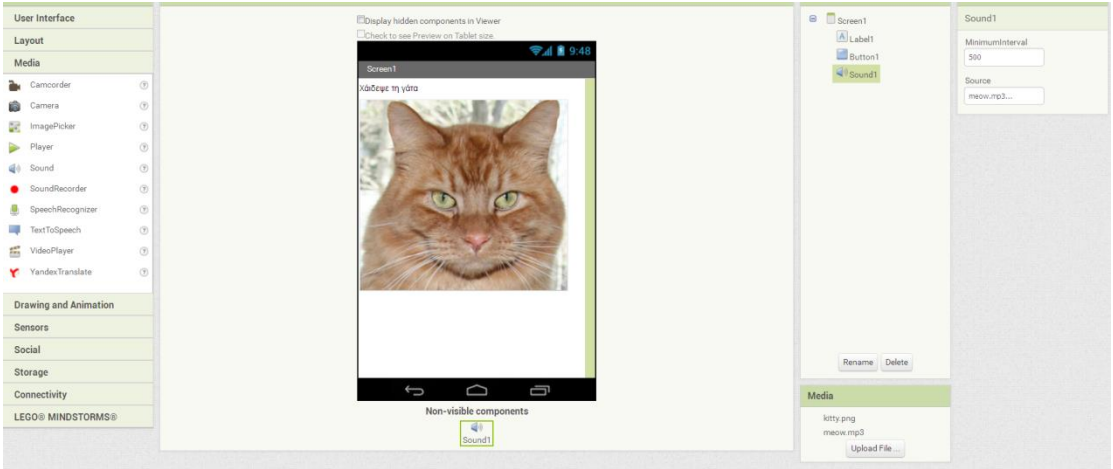
Βήμα 2ο. Πηγαίνουμε στην αριστερή πλευρά του περιβάλλοντος σχεδίασης και επιλέγουμε την κατηγορία user interface και από εκεί ένα label. Το σύρουμε και το τοποθετούμε στο κέντρο στο screen1.Το label περιέχει εξορισμού το κείμενο «Text for label». Πάμε στη δεξιά μεριά, στις ιδιότητες του label και στο σημείο text εισάγουμε την έκφραση «Χάιδεψε τη γάτα», η οποία τώρα φαίνεται στο label στην οθόνη του κινητού.

Βήμα 3ο. Κατεβάζουμε από το διαδίκτυο μία εικόνα γάτας, π.χ. http://appinventor.org/bookFiles/HelloPurr/kitty.png και ένα ήχο γάτας που νιαουρίζει, π.χ.http://appinventor.org/bookFiles/HelloPurr/meow.mp3. Αν έχετε κατάλληλη διάθεση αναζητείστε άλλες.

Βήμα 4ο. Ανεβάζουμε από την επιλογή upload file στην περιοχή Media τα αρχεία ήχου και εικόνας για την γάτα.

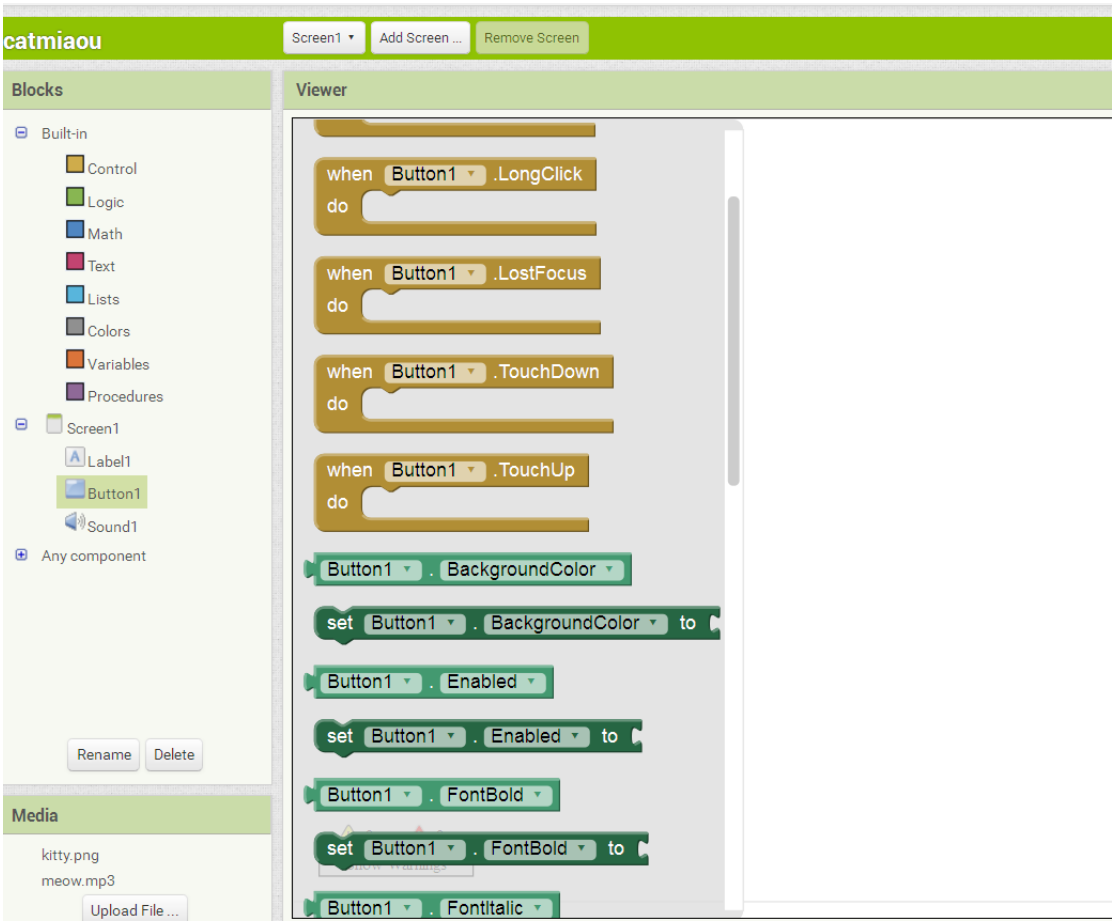
Βήμα 5ο.Επιλέγουμε από την αριστερή παλέτα ένα κουμπί και το σύρουμε και το αφήνουμε στην οθόνη (screen1). Πηγαίνουμε στη δεξιά πλευρά στις ιδιότητες. Στην επιλογή image επιλέγουμε το αρχείο με την εικόνα της γάτας. Στην επιλογή Text διαγράφουμε το κείμενο.

Βήμα 6ο. Τοποθετούμε από την αριστερή παλέτα και την κατηγορία media τον ήχο (sound) και το σύρουμε στην οθόνη (screen1). Επιλέγουμε από τις ιδιότητες το αρχείο ήχου. Οπότε έχουμε το αποτέλεσμα που φαίνεται στην Εικόνα 13.3.1. Παρατηρείστε ότι ο ήχος βρίσκεται στα μη ορατά στοιχεία της οθόνης (non-visible components). Τα στοιχεία τα οποία είναι μη ορατά εκτελούν λειτουργίες για την εφαρμογή αλλά δε φαίνονται στην οθόνη.



Εικόνα 13.3.1. Παράδειγμα 1 – γάτα που νιαουρίζει.

Θα πρέπει τώρα να δώσουμε ενέργεια στη γάτα, οπότε όταν την ακουμπάμε να εκτελείται ο ήχος του νιαουρίσματος για ένα συγκεκριμένο χρονικό διάστημα. Για να γίνει αυτό επιλέγουμε δεξιά πάνω το κουμπί blocks και οδηγούμαστε στο περιβάλλον προγραμματισμού όπως φαίνεται στην Εικόνα 13.3.2.

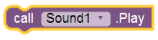


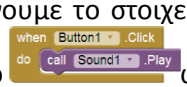
Εικόνα 13.3.2. Το περιβάλλον προγραμματισμού.

Στο αριστερό σημείο κάτω από τη λέξη **Blocks** υπάρχει μία στήλη με τις επιλογές build-in, στην οποία περιέχονται οι κατηγορίες με τα προγραμματιστικά πλακίδια (ξεχωρίζουν από το χρώμα τους) και τα στοιχεία τα οποία έχουμε τοποθετήσει στην οθόνη. Όταν κάνουμε κλικ σε ένα από τα στοιχεία, π.χ. στο Button1 εμφανίζονται τα πλακίδια που αντιστοιχούν σε αυτό.

Επιλέγουμε το button1 και στη συνέχεια το πλακίδιο 

. Τα πλακίδια που περιέχουν τη λέξη when όπως το συγκεκριμένο διαχειρίζονται γεγονότα. Στη συγκεκριμένη περίπτωση το γεγονός είναι το πάτημα τουbutton1. Δηλαδή, όταν (when) πατηθεί (click) το button1 (γεγονός) τί θα συμβεί (do).

Στη συνέχεια επιλέγουμε το στοιχείο sound1 και σύρουμε το πλακίδιο 

και το τοποθετούμε μέσα στο άλλο 

ώστε να ταιριάξει σύμφωνα με τις εγκοπές.

13.4 Η χρήση του προσομοιωτή για κινητά και υπολογιστή (companion/emulator)

Το επόμενο βήμα είναι να δούμε στην οθόνη του κινητού είτε του δικού μας είτε του εικονικού να τρέχει η εφαρμογή που μόλις φτιάξαμε.

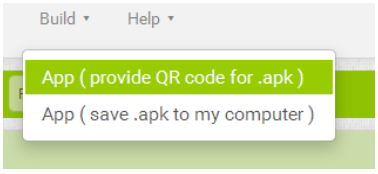
Μπορούμε με δύο τρόπους να εκτελέσουμε την εφαρμογή μας.

1ος.Να συνδέσουμε με το ίδιο ασύρματο δίκτυο και τον υπολογιστή στον οποίο δημιουργήσαμε την εφαρμογή και τη συσκευή μας με android. Θα χρειαστεί να εγκαταστήσουμε στη συσκευή android το MIT AIcompanion. Αυτό μπορεί να γίνει εύκολα διαβάζοντας τον παρακάτω QR κωδικό.

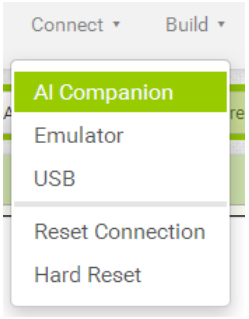
Εικόνα 13.4.1. Για αυτόματη ανανέωση λογισμικούMITAITcompanion.

Εικόνα 13.4.2. Χειροκίνητη ανανέωση λογισμικούMITAITcompanion.

Στη συνέχεια επιλέγουμε από το μενού Build→App (provideQRcodefor .apk) (βλέπε Εικόνα 13.4.3) .Κατόπιν επιλέγουμεConnect→AICompanion (βλέπε Εικόνα 13.4.4). και σκανάρουμε με την εφαρμογή MIT AIcompanion τον QR κωδικό και τρέχουμε την εφαρμογή μας σε πραγματικό χρόνο στη συσκευή μας.

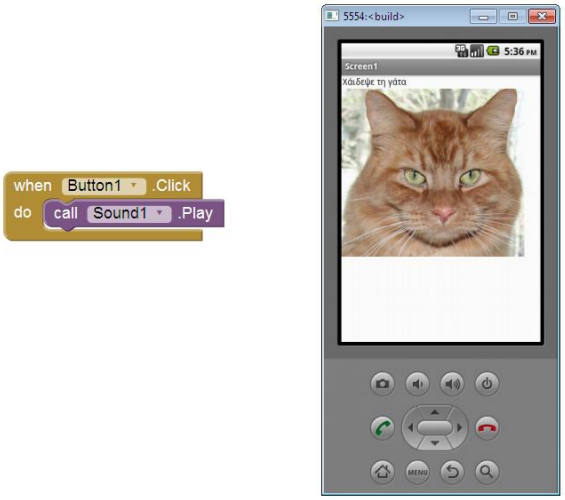


Εικόνα 13.4.3. Επιλογή για τη δημιουργία του QRκωδικού για την εφαρμογή μας.



Εικόνα 13.4.4. Σύνδεση μέσω AICompanion.

2ος.Εγκαθιστούμε στον υπολογιστή μας το aiStarter (από τη θέσηhttp://appinventor.mit.edu/explore/ai2/setup-emulator.html). Τρέχουμε το aiStarter και στη συνέχεια επιλέγουμε να συνδεθούμε (connect→emulator) με τον προσομοιωτή. Σε αυτή την περίπτωση εμφανίζεται μια εικονική συσκευή androidστην οποία εκτελείται η εφαρμογή μας, βλέπε Εικόνα 13.4.5.



Εικόνα 13.4.5. Εκτέλεση της εφαρμογής σε εικονική συσκευή Android.

13.5 Ανάρτηση εφαρμογής

Στην Εικόνα 13.1.1 είδαμε το περιβάλλον διαχείρισης των έργων που δημιουργούμε στο AppInventor. Στο περιβάλλον αυτό έχουμε τη δυνατότητα να δημοσιεύσουμε την εφαρμογή μας με την επιλογή publish to gallery. Με τη δημοσίευση μοιραζόμαστε την εφαρμογή μας με την κοινότητα του AppInventor παρέχοντας παράλληλα στοιχεία, όπως είναι για παράδειγμα τί κάνει η εφαρμογή, αν έχει κάποιο βίντεο σχετικό, αν έχει πάρει κάποια κομμάτια προγραμματιστικά από άλλη εφαρμογή και ποια είναι αυτή.

Κεφάλαιο 14

Σχεδίαση Διεπαφής χρήστη και γεγονότα

14. Σχεδίαση Διεπαφής χρήστη και γεγονότα

Εισαγωγή

Έχουμε ήδη σχεδιάσει, στο προηγούμενο κεφάλαιο, μια απλή διεπαφή χρήστη. Όπως γνωρίζουμε όμως η διεπαφή είναι το σημείο από το οποίο ξεκινά η λειτουργικότητα της εφαρμογής. Εδώ τοποθετούνται τα αντικείμενα πάνω στα οποία ο χρήστης θα έχει τη δυνατότητα να αλληλοεπιδράσει με την εφαρμογή.

Όσο πιο σύνθετες είναι οι εφαρμογές μας τόσο περισσότερα στοιχεία πρέπει να εισάγουμε στη διεπαφή. Συχνά μάλιστα θα έχουμε και διαφορετικές οθόνες (screen ή layout) διεπαφής που θα γίνονται ενεργές όταν επιλέγουμε να εκτελέσουμε τις αντίστοιχες λειτουργίες που στην ενότητα 2Α τις ονομάσαμε Δραστηριότητες (Activities). Οι εφαρμογές που θα αναπτύξουμε στη συνέχεια ακολουθούν αυτή τη λογική.

Ενότητες Κεφαλαίου

- Εισαγωγή και σχεδίαση εικόνων, data assets
- Διαχείριση ενεργειών χρήστη (είσοδοι από πληκτρολόγιο, ποντίκι, οθόνη αφής)
- Η χρήση του καμβά σε σχέση με την οθόνη αφής
- Υλοποίηση άλλων γεγονότων (όπως συγκρούσεις)

14.1 Εισαγωγή και σχεδίαση εικόνων (data assets)

Στο AppInventor έχουμε τη δυνατότητα να εισάγουμε εικόνες από το σκληρό δίσκο οι οποίες αποθηκεύονται ως data assets. Data asset μπορεί να είναι και οποιοδήποτε άλλο αρχείο που χρησιμοποιείται από την εφαρμογή, όπως παράδειγμα ένα αρχείο ήχου. Αυτό σημαίνει ότι όταν θα εκτελεστεί η εφαρμογή σε συσκευή android τα data assets ακολουθούν την εφαρμογή και αποθηκεύονται αυτόματα στη μνήμη της συσκευής.

14.2 Διαχείριση ενεργειών χρήστη.

Θα δημιουργήσουμε μια εφαρμογή η οποία θα αποτελείται από δύο οθόνες. Μια εισαγωγική, στην οποία θα εμφανίζεται ένα μήνυμα καλωσορίσματος για πέντε δευτερόλεπτα, και μια με ένα μενού με μουσική υπόκρουση.

Χρειαζόμαστε δύο πλαίσια κειμένου (label), ένα για το μήνυμα καλωσορίσματος κι ένα που δεν θα είναι ορατό αλλά θα χρησιμεύσει για τον υπολογισμό των δευτερολέπτων που θέλουμε να χρησιμοποιήσουμε ως όριο προβολής της οθόνης. Επίσης, θα χρησιμοποιήσουμε ένα αντικείμενο *Sound* για το ηχητικό σήμα που θα βάλουμε ως έναρξη της εφαρμογής, κι ένα *Clock* για τον υπολογισμό των δευτερολέπτων.

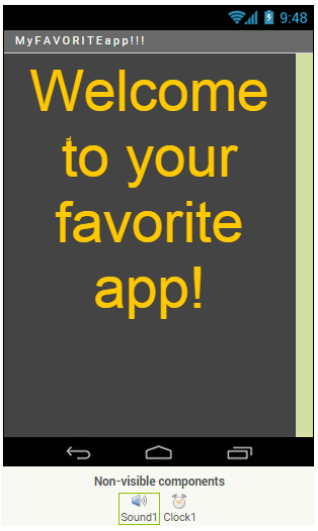
Αντικείμενα: για να σχεδιάσουμε την οθόνη μας λοιπόν, θα χρειαστούμε τα εξής αντικείμενα,

- Label και τα μη ορατά
- Sound και
- Clock

Βήμα 1:Στο AppInventor επιλέγουμε το περιβάλλον σχεδίασης (*Designer)*.

Βήμα 2: Επιλέγουμε καινούριο έργο (*New Project)*

Βήμα 3: Επιλέγουμε από το User Interface το label και στην ιδιότητα text γράφουμε το μήνυμα «welcome to your favorite application». (βλ. Εικόνα 14.2.1)



Εικόνα 14.2.1. Αρχική εικόνα.

Βήμα 4: Επιλέγουμε ένα δεύτερο αντικείμενο τύπου label και ένα τύπου sound από την κατηγορία media. Στις ιδιότητες του sound επιλέγουμε τον ήχο (Source) που θέλουμε και τον οποίο

θα πρέπει να έχουμε ήδη ανεβάσει στο Media της εφαρμογής μας. Επίσης επιλέγουμε τις ιδιότητες MinimumInterval=500 (5 seconds). (Εικόνα 14.2.2)

Εικόνα 14.2.2. Ιδιότητες ήχου.

Βήμα 5: Επιλέγουμε ένα αντικείμενο τύπουclockαπό την κατηγορία Sensors.

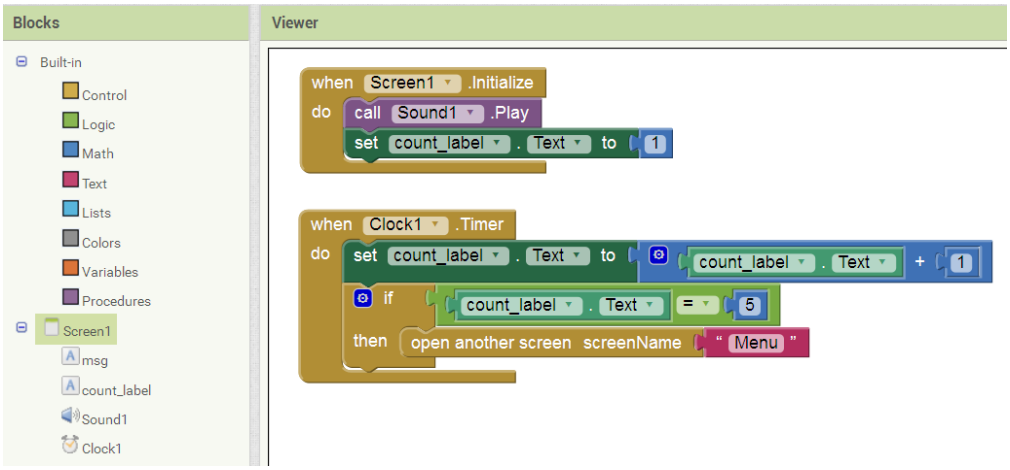
Βήμα 6: Στο περιβάλλον σχεδίασης επιλέγουμε add screen και δίνουμε το όνομα menu. Είναι η βασική οθόνη με το μενού μας. Τοποθετούμε ένα label με κείμενο MENU.

Πάμε στο περιβάλλον προγραμματισμού (blocks) για να δώσουμε ενέργειες στα αντικείμενα που τοποθετήσαμε στις οθόνες.

Όταν ξεκινάει η εφαρμογή και φορτώνεται η οθόνη Screen1, χρησιμοποιήσουμε μία ετικέτα (label) (με ιδιότητα Visible=false) ως μεταβλητή τύπου μετρητή, με αρχική τιμή 1.

Επίσης επιλέγουμε να ακουστεί ο ήχος που διαλέξαμε. Όταν (when) αρχικοποιείται η οθόνη (Screen1.initialize) θέσε τιμή στην ετικέτα μετρητή το 1 (dosetcounter_label.Textto 1).

Όταν ξεκινάει η εφαρμογή, ενεργοποιείται το χρονόμετρο του clock(Timer). Σε κάθε αλλαγή του χρονομέτρου προσθέτουμε 1 στην ετικέτα-μετρητή. Όταν αυτή η τιμή γίνει 5 τότε εμφανίζεται η επόμενη οθόνη η οποία είναι το Menu. (βλέπε Εικόνα 14.2.3).

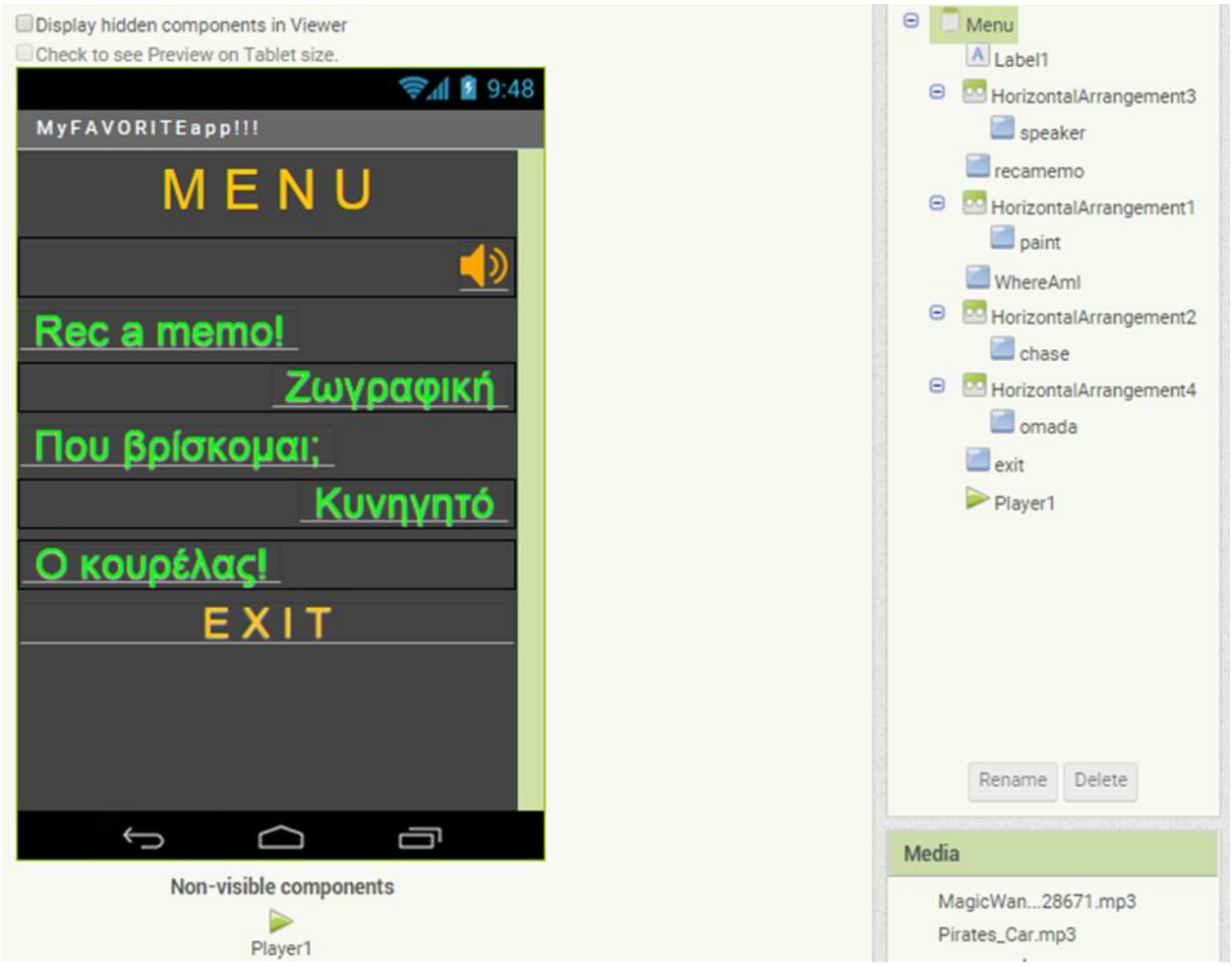


Εικόνα 14.2.3. Πρόγραμμα για την αλλαγή της οθόνης

14.2.1 Κατασκευή μενού και ήχος background

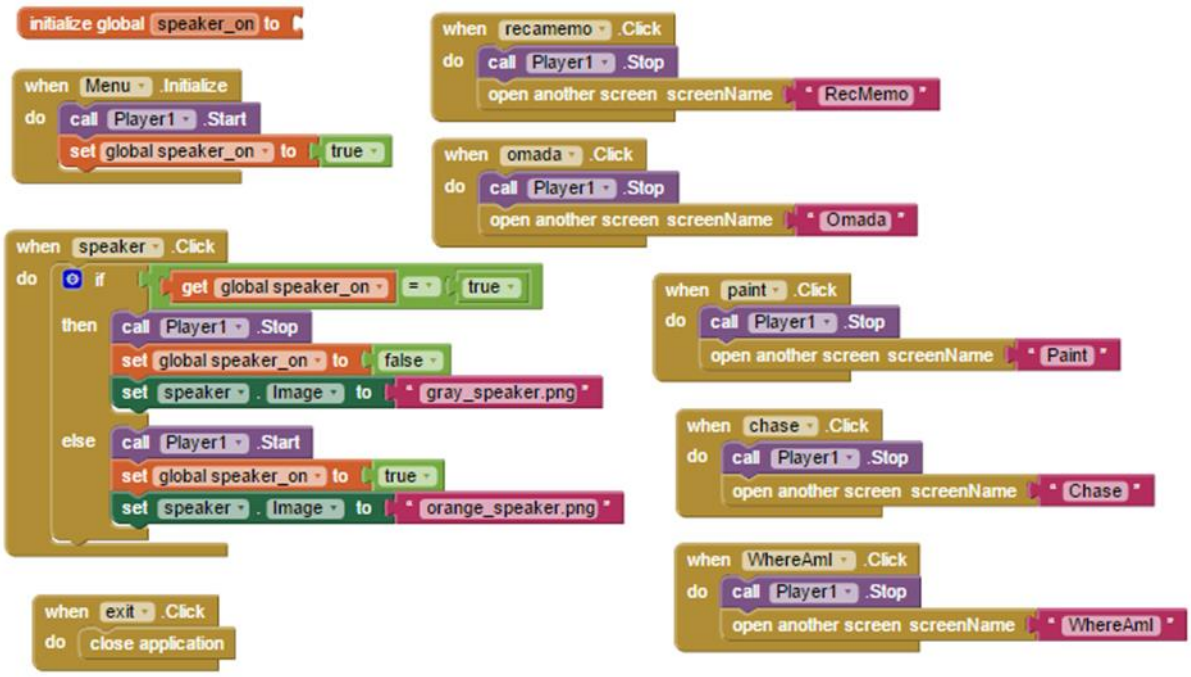
Η καινούρια οθόνη (menu) περιλαμβάνει τις αγαπημένες εφαρμογές τις οποίες μπορεί να επιλέξει, πατώντας το αντίστοιχο κουμπί, ο χρήστης. Η οθόνη θα συνοδεύεται από μουσική.

Για τη σχεδίαση της οθόνης, θα χρησιμοποιήσουμε τα παρακάτω αντικείμενα: buttons, label, player και περιέργως horizontal arrangement ώστε να τοποθετηθούν τα υπόλοιπα αντικείμενα στην επιθυμητή θέση. Ορίζουμε έτσι αυτά που θα λέγαμε στο 2Α κατάλληλα λογικά τμήματα στην οθόνη.



Το αντικείμενο player μας επιτρέπει να ενσωματώσουμε ήχο στο background, ο οποίος θα παίζει ενόσω η οθόνη μας περιμένει κάποια ενέργεια του χρήστη. Με το κουμπί speaker, στο οποίο έχουμε τοποθετήσει κατάλληλο Image ηχείου, ο χρήστης μπορεί να απενεργοποιήσει τον player σταματώντας τη μουσική. Από την ίδια θέση μπορεί να επανενεργοποιήσει. Προφανώς θα πρέπει να συνδεθεί ένα αρχείο ήχου με τον player, συμπληρώνοντας το όνομά του στην ιδιότητα source του player, αφού πρώτα έχει ανέβει στο φάκελο Media.

Ακολουθεί ο σχετικός κώδικας.



14.3 Η χρήση του καμβά σε σχέση με την οθόνη αφής

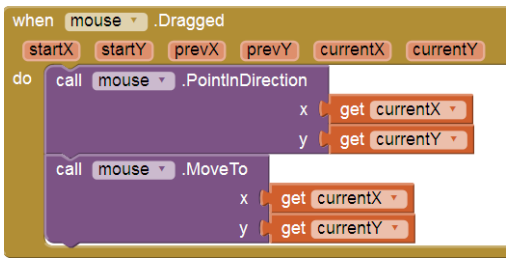
Στη συνέχεια δημιουργούμε μια καινούρια οθόνη με όνομα chase στην οποία θα σχεδιάσουμε ένα παιχνίδι με ποντίκι, τυρί και γάτα. Το ποντίκι τρώει τα τυράκια που εμφανίζονται τυχαία πάνω στην οθόνη, ενώ η γάτα προσπαθεί να φάει το ποντίκι.

Τοποθετούμε στην οθόνη καμβά. Ο καμβάς είναι ένα πάνελ δύο διαστάσεων το οποίο είναι ενεργό στην αφή και στο οποίο είναι δυνατό να σχεδιαστούν εικόνες. Ο καμβάς βρίσκεται στην κατηγορία Drawing and Animation. Τον καμβά τον ονομάζουμε GrassCanvas και τοποθετούμε από τις ιδιότητες ως εικόνα φόντου μία εικόνα γρασίδι (Grass.jpg). Επίσης στις ιδιότητες πλάτος και ύψος επιλέγουμε fill parent ώστε να καταλαμβάνει όσο χώρο έχει η οθόνη της συσκευής.

Ο λόγος που χρησιμοποιούμε τον καμβά είναι να οριοθετήσουμε την ενεργή περιοχή της εφαρμογής μας στην οθόνη, στην οποία θα τοποθετούνται ή κινούνται γραφικά. Επίσης έχει να κάνει με τη διαχείριση γεγονότων αφής σε μια κατάλληλη οθόνη (touch screen). Οι εικόνες (image sprites) που τοποθετούνται πάνω του μπορούν να ταυτοποιηθούν από τη θέση τους. Ταυτόχρονα ο καμβάς αναγνωρίζει γεγονότα αφής (dragged, flang, touchdown, touchup, touched) και τη θέση που αφορούν. Έχοντας τη θέση τόσο για κάθε κινούμενο ή σταθερό sprite όσο και για το γεγονός εύκολα αναγνωρίζει σε ποιο γραφικό αντικείμενο αντιστοιχεί το γεγονός. Φυσικά εμείς βλέπουμε και διαχειριζόμαστε, μέσα από μεθόδους του αντικειμένου, το τελικό στάδιο όταν το αντικείμενο ήδη έχει ενημερωθεί ότι έχει ακουμπηθεί, τραβηχτεί, συγκρουστεί με άλλο κλπ.

Τοποθετούμε στον καμβά μία εικόνα (image sprite) ποντικιού και το ονομάζουμε mouse. Στις ιδιότητες του image sprite επιλέγουμε ως πηγή το αρχείο του ποντικιού mouse.png.

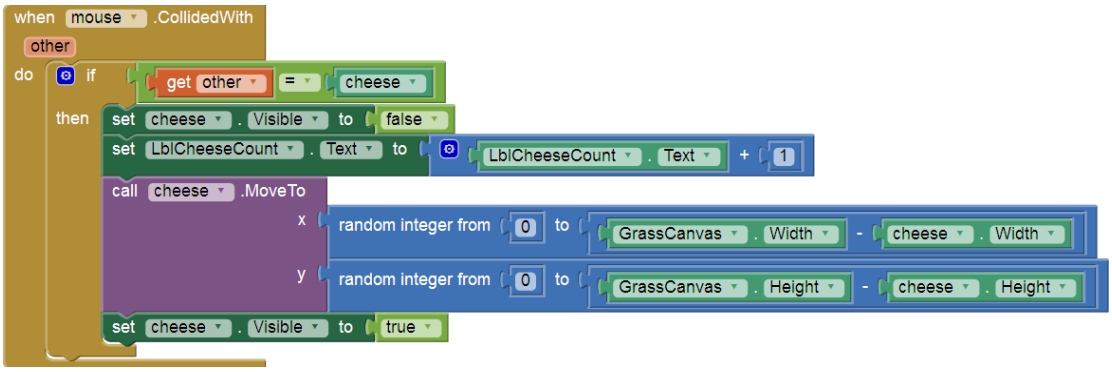
Στο περιβάλλον προγραμματισμού δίνουμε ενέργεια στο ποντίκι όπως φαίνεται στην Εικόνα 14.3.1. Με τον τρόπο αυτό το ποντίκι θα κινείται προς την κατεύθυνση που κινείται το δάκτυλό μας πάνω στην οθόνη της συσκευής. Το πλακίδιο callmouse.PointDirectionείναι μια μέθοδος η οποία στρέφει το ποντίκι να δείχνει προς τις συντεταγμένες του σημείου αφής. Στη συνέχεια με την επόμενη μέθοδο mouse.MoveTo, το ποντίκι κινείται προς τις συγκεκριμένες συντεταγμένες.



Εικόνα 14.3.1. Η κίνηση του ποντικιού

14.4 Υλοποίηση άλλων γεγονότων (όπως συγκρούσεις)

Θέλουμε τώρα το ποντίκι να τρώει το τυρί. Τοποθετούμε στον καμβά ένα image sprite με εικόνα τυριού, ονόματι cheese. Όταν το ποντίκι συγκρουστεί με το τυρί τότε το τυρί εξαφανίζεται δημιουργώντας την αίσθηση ότι έχει φαγωθεί και εμφανίζεται σε μια τυχαία θέση μέσα στον καμβά, όπως φαίνεται στην Εικόνα 14.4.1.



Εικόνα 14.4.1. Διαχείριση της σύγκρουσης των αντικειμένων.

Κεφάλαιο 15

Αντικείμενα στο περιβάλλον AppInventor2

15. Αντικείμενα στο περιβάλλον AppInventor2

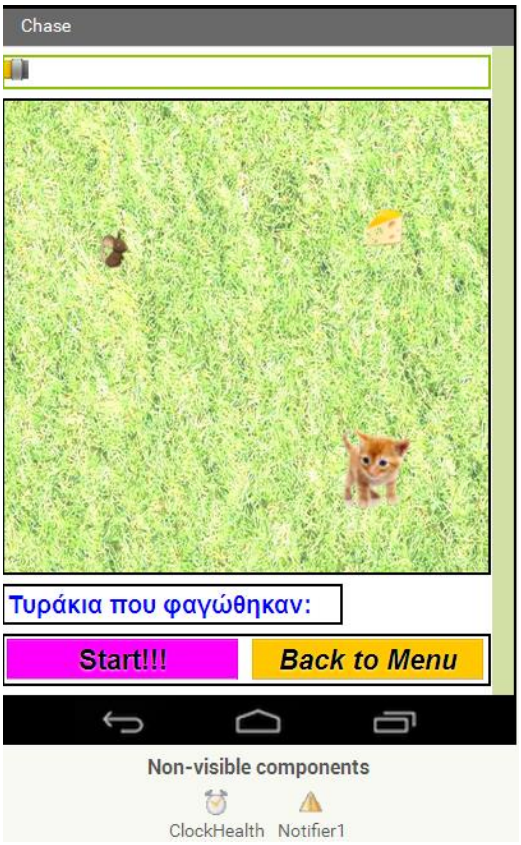
Ενότητες Κεφαλαίου

- Αντικείμενα και κώδικας στο AppInventor2
- Ιδιότητες αντικειμένου (Εμφάνιση, Κίνηση κλπ.)
- Μέθοδοι αντικειμένου
- Χειρισμός Πολυμεσικών αντικειμένων
- Αλληλεπίδραση αντικειμένων μέσα από μηνύματα
- Το AppInventor ως γλώσσα προγραμματισμού με περιβάλλον ανάπτυξης

15.1 Αντικείμενα και κώδικας στο AppInventor2

Είδαμε στα προηγούμενα κεφάλαια πώς μπορούμε να τοποθετήσουμε διάφορα στοιχεία πάνω σε μία οθόνη. Τα στοιχεία που τοποθετούμε στην οθόνη είτε ορατά (visible) είναι αόρατα (nonvisible) αποτελούν τα αντικείμενα του προγράμματος και έχουν την ίδια έννοια με τα αντικείμενα της java. Στο AppInventor στο σημείο Components φαίνονται τα αντικείμενα που έχουμε τοποθετήσει στην οθόνη. Σημειώνεται ότι υπάρχει η δυνατότητα να δημιουργούνται αντικείμενα και από το προγραμματιστικό περιβάλλον.

Για παράδειγμα στην εφαρμογή μας στην οθόνη chase έχουμε τα αντικείμενα cat, mouse, cheese. Στην Εικόνα 15.1.1 φαίνεται η οθόνη chase. Περιέχει επίσης ένα label, δύο κουμπιά (buttons) και ένα αντικείμενο τύπου slider. Παρατηρούμε ότι υπάρχουνε και δύο αόρατα αντικείμενα , το ClockHealth το οποίο μετράει τον χρόνο για να λήξει το παιχνίδι και το Notifier1, το οποίο εμφανίζει ένα κατάλληλο μήνυμα όταν λήξει το παιχνίδι.



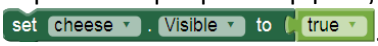
Εικόνα 15.1.1. Η οθόνη chase

Στο προγραμματιστικό περιβάλλον του AppInventor δίνουμε ενέργεια σε κάθε αντικείμενο μέσα από τη χρήση των προγραμματιστικών πλακιδίων τα οποία αντιστοιχούν στον κώδικα της εφαρμογής μας.

15.2 Ιδιότητες αντικειμένου (Εμφάνιση, Κίνηση κλπ.)

Κάθε αντικείμενο που δημιουργούμε έχει ιδιότητες. Τις ιδιότητες ενός αντικειμένου μπορούμε να τις αλλάξουμε με δύο τρόπους. Είτε από το περιβάλλον σχεδίασης στο σημείο properties είτε μέσα από το πρόγραμμά μας.

Από το περιβάλλον σχεδίασης: Για παράδειγμα έχουμε ένα label στο οποίο θέλουμε να τοποθετήσουμε το κείμενο «Τυράκια που φαγώθηκαν». Πάμε στις ιδιότητες και στο σημείο Text προσθέτουμε το κείμενο που θέλουμε.

Από το περιβάλλον προγραμματισμού: Μπορούμε να εμφανίζουμε και να εξαφανίζουμε το τυράκι από την ιδιότηταVisible ως εξής: 

.

Δίνοντας τιμή true γίνεται ορατό ενώ με την τιμή false γίνεται αόρατο.

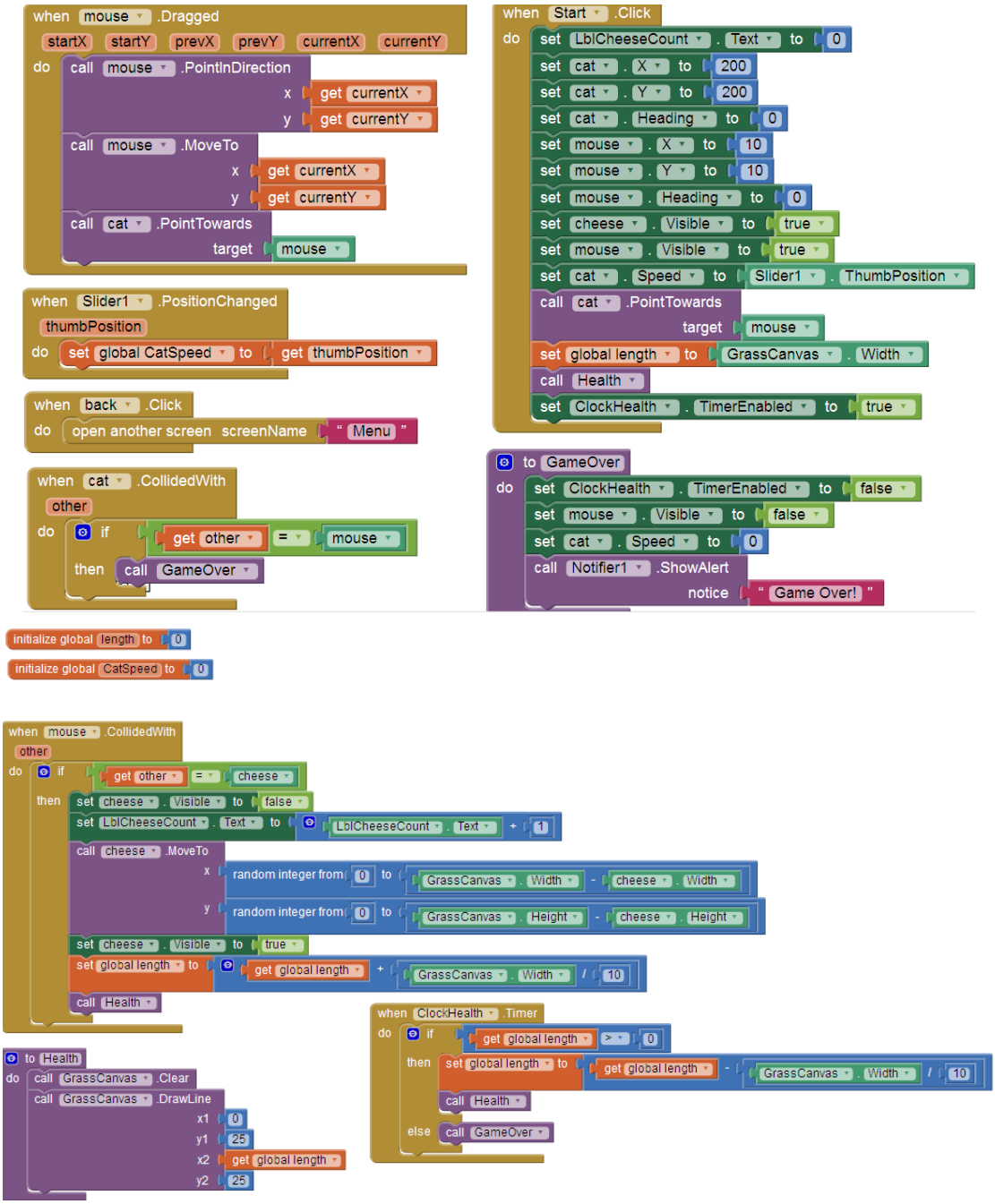
Το αντικείμενο slider θα χρησιμοποιηθεί για τη ρύθμιση της ταχύτητας ενός άλλου αντικειμένου, συγκεκριμένα του imagespite cat, ώστε ο χρήστης να έχει την δυνατότητα να επηρεάζει κατ' επιλογή του την ταχύτητα της γάτας, προσαρμόζοντας τη δυσκολία του παιχνιδιού.

Για να μπορέσουμε αποτυπώσουμε την αυξομείωση της ζωής του πρωταγωνιστή ποντικού, χρησιμοποιούμε ένα διανυσματικό γραφικό, στην πραγματικότητα μια απλή γραμμή που σχεδιάζεται σε συγκεκριμένη θέση στον καμβά (drawline). Αυτό γίνεται χρησιμοποιώντας μια μεταβλητή (globalLength), που μεταβάλλεται με βάση τη μεταβολή της ζωής, δηλαδή μειώνεται με το χρόνο (ClockHealth) και αυξάνεται με την επιτυχημένη συλλογή αντικειμένων (των τυριών).

15.3 Μέθοδοι αντικειμένου

Σε κάθε αντικείμενο, ανάλογα με το είδος του, αντιστοιχούν διάφορες ενέργειες οι οποίες καλούνται μέθοδοι. Στην Εικόνα 15.3.1 φαίνεται το πρόγραμμα για την οθόνη chase. Για το αντικείμενο mouse όταν εκτελεστεί το γεγονός Dragged, που σημαίνει ότι προσπαθήσουμε να σύρουμε το αντικείμενο mouse στην οθόνη της συσκευής περιγράφεται στο do τί θα συμβεί. Μία μέθοδος για το αντικείμενο mouse είναι το PointTowardstarget. Παρατηρούμε ότι και το αντικείμενο cat έχει την ίδια μέθοδο PointTowardstarget αφού πρόκειται για το ίδιο είδος αντικειμένου. Μπορούμε να δημιουργήσουμε και τη δική μας μέθοδο όπως είναι η GameOver (από τις επιλογές εντολών Procedure), την οποία και καλούμε χωρίς τη χρήση αντικειμένου. Για παράδειγμα χρησιμοποιήσαμε την εντολή cat.PointTowardstarget, ενώ για το GameOver το καλέσαμε χωρίς τη χρήση αντικειμένου.

Παρατηρείστε τα κόκκινα πλακίδια τα οποία αναφέρονται σε μεταβλητές για τις οποίες θα μιλήσουμε και στη συνέχεια.



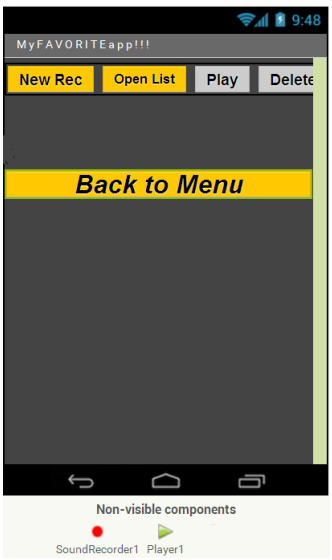
Εικόνα 15.3.1. Το πρόγραμμα για την οθόνη Chase.

15.4 Χειρισμός Πολυμεσικών αντικειμένων

Στην κατηγορία media υπάρχουν διάφορα πολυμεσικά αντικείμενα που μπορούν να τοποθετηθούν στο πρόγραμμά μας. Εμείς εδώ χρησιμοποιήσαμε τον ήχο ο οποίος ξεκίνησε να παίζει κατά την αρχικοποίηση της αρχικής οθόνης screen1, και σταμάτησε μετά από κάποια δευτερόλεπτα.

Για κάθε αντικείμενο τύπου media υπάρχουν διάφορες μέθοδοι και γεγονότα τα οποία μας βοηθούν στο να τα διαχειριστούμε.

Δημιουργούμε μια καινούρια οθόνη ονόματι RecMemo όπως φαίνεται στην Εικόνα 15.4.1.



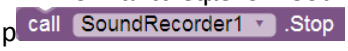
Εικόνα 15.4.1. Οθόνη RecMemo.

Η νέα εφαρμογή καταγράφει και αποθηκεύει ηχητικές σημειώσεις, οι οποίες μπορούμε να τις βλέπουμε σε μορφή λίστας, να επιλέγουμε και να αναπαράγουμε όποιο ηχητικό αρχείο θέλουμε ή να το διαγράφουμε.

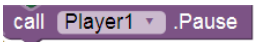
Παρατηρούμε ότι έχουμε

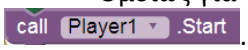
- SoundRecorder
- Player
- 5 Buttons

Το αντικείμενο SoundRecorder έχει τη μέθοδο start 

και stop 

. Οπότε θα πρέπει να φτιάξουμε το πρόγραμμά μας έτσι ώστε, όταν θα πατηθεί το κουμπί REC να ξεκινήσει η εγγραφή του ήχου, ενώ όταν το ξαναπατήσουμε να σταματά. Θα δούμε σε επόμενο κεφάλαιο πώς το αρχείο που δημιουργείται αποθηκεύεται.

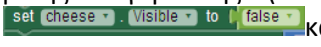
Ομοίως για το αντικείμενο Player θα χρησιμοποιηθούν οι μέθοδοι 

και 

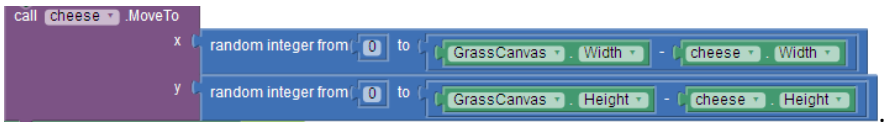
.

Αργότερα, σε παρακάτω κεφάλαιο, θα χρησιμοποιήσουμε μια απλή βάση δεδομένων (tinyDB), ενός πίνακα, για την αποθήκευση των εγγραφών μας ώστε να μπορούμε να επιλέξουμε ποια θα ανακαλέσουμε.

15.5 Αλληλεπίδραση αντικειμένων μέσα από μηνύματα

Στην οθόνη chase είδαμε ότι όταν το ποντίκι (mouse) συγκρουστεί με το τυρί (cheese) τότε το τυρί εξαφανίζεται και εμφανίζεται σε μια τυχαία θέση μέσα στον καμβά που έχουμε ορίσει. Στην πραγματικότητα, η μέθοδος διαχείρισης σύγκρουσης (CollidedWith) στέλνει μήνυμα στο αντικείμενο τυρί να εξαφανιστεί 

και επίσης στέλνει μήνυμα να τοποθετηθεί σε τυχαία θέση με τη μέθοδο:

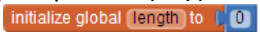


.

Παρατηρούμε ότι μέσα από την κλήση των μεθόδων με τη χρήση μηνυμάτων μπορούν τα αντικείμενα να επικοινωνούν μεταξύ τους και να ενεργοποιούν το ένα το άλλο.

Στην πραγματικότητα, χρησιμοποιείται στο background ένα αντικείμενο τύπου Content Provider για τη διαχείριση της επικοινωνίας μεταξύ των αντικειμένων της ενεργής οθόνης.

15.6 Το AppInventor ως γλώσσα προγραμματισμού με περιβάλλον ανάπτυξης

Το AppInventor διαθέτει τη δομή της μεταβλητής. Επίσης θα πρέπει να αρχικοποιούνται οι μεταβλητές όπως και σε κάθε γλώσσα προγραμματισμού. Παράδειγμα στην περίπτωση της οθόνης chase αρχικοποιούνται οι μεταβλητές length και speed ως 

και 

αντίστοιχα.

Επίσης, υπάρχουν οι δομές επιλογής και επανάληψης στην επιλογή control του περιβάλλοντος προγραμματισμού, όπως και κάποιες δομές δεδομένων, όπως είναι η λίστα που θα δούμε σε επόμενο κεφάλαιο αλλά και αλληλεπίδραση με βάση δεδομένων. Επιπλέον υπάρχει η έννοια του αντικειμένου, της μεθόδου και της επικοινωνίας μεταξύ των αντικειμένων.

Βγάζουμε το συμπέρασμα ότι το AppInventor αποτελεί ένα ολοκληρωμένο περιβάλλον προγραμματισμού το οποίο είναι πολύ φιλικό στο χρήστη.

Κεφάλαιο 16

Αξιοποίηση υπάρχοντος κώδικα

16. Αξιοποίηση υπάρχοντος κώδικα

Ενότητες Κεφαλαίου

- Η λογική του ανοιχτού λογισμικού
- Η κοινότητα του AppInventor
- AppInventor tutorials

16.1 Η λογική του ανοιχτού λογισμικού

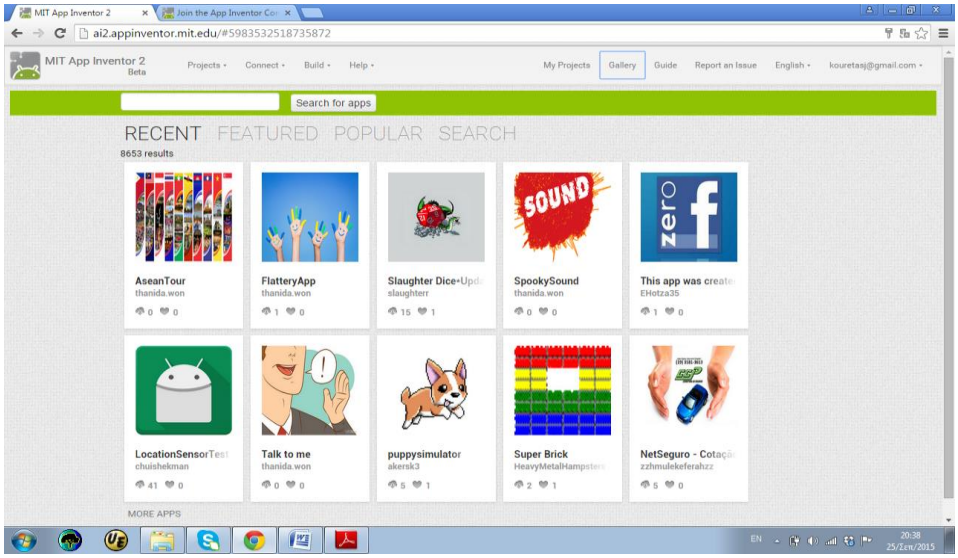
Το Ελεύθερο Λογισμικό / Λογισμικό Ανοικτού Κώδικα (ΕΛ/ΛΑΚ) είναι το λογισμικό που ο καθένας μπορεί ελεύθερα να χρησιμοποιεί, να αντιγράφει, να διανέμει και να τροποποιεί ανάλογα με τις ανάγκες του. Είναι ένα εναλλακτικό μοντέλο ανάπτυξης και χρήσης λογισμικού που βασίζεται στην ελεύθερη διάθεση του πηγαίου κώδικα, το οποίο παρέχει τη δυνατότητα αλλαγών ή βελτιώσεων ώστε να καλύπτονται οι ανάγκες αυτού που το χρησιμοποιεί.

Το κόστος άδειας χρήσης των εφαρμογών ελεύθερου λογισμικού είναι τις περισσότερες φορές μηδενικό. Δεν αγοράζονται άδειες χρήσεις και μπορούμε να έχουμε απεριόριστο αριθμό εγκαταστάσεων. Η χρήση ανοιχτού κώδικα δεν περιορίζει τον οργανισμό ή τον απλό χρήστη σε μια σχέση εξάρτησης από εταιρίες και επειδή η διανομή, η διόρθωση σφαλμάτων και η ανάπτυξη του λογισμικού ΕΛ/ΛΑΚ μπορεί να γίνει από κάθε τεχνικά καταρτισμένη ομάδα, δημιουργείται ένα περιβάλλον έντονου ανταγωνισμού ο οποίος οδηγεί σε χαμηλές τιμές και υψηλές υπηρεσίες υποστήριξης.

Το AppInventor διαθέτει όλα εκείνα τα χαρακτηριστικά του ελεύθερου λογισμικού με αποτέλεσμα να καθίσταται ιδανικό για την εκπαίδευση. Όλοι μπορούν να έχουν πρόσβαση σε αυτό χωρίς περιορισμούς και χωρίς κόστος.

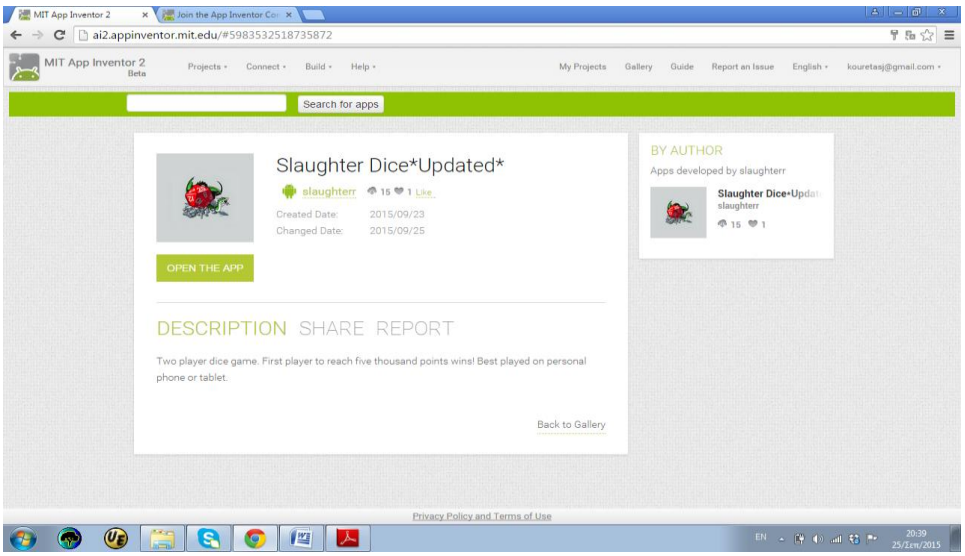
16.2 Η κοινότητα του AppInventor

Στο περιβάλλον έργου μπορούμε να επιλέξουμε το κουμπί gallery το οποίο μας οδηγεί στην Εικόνα 16.2.1. Εδώ μπορούμε να αναζητήσουμε έργα άλλων μελών της κοινότητας.



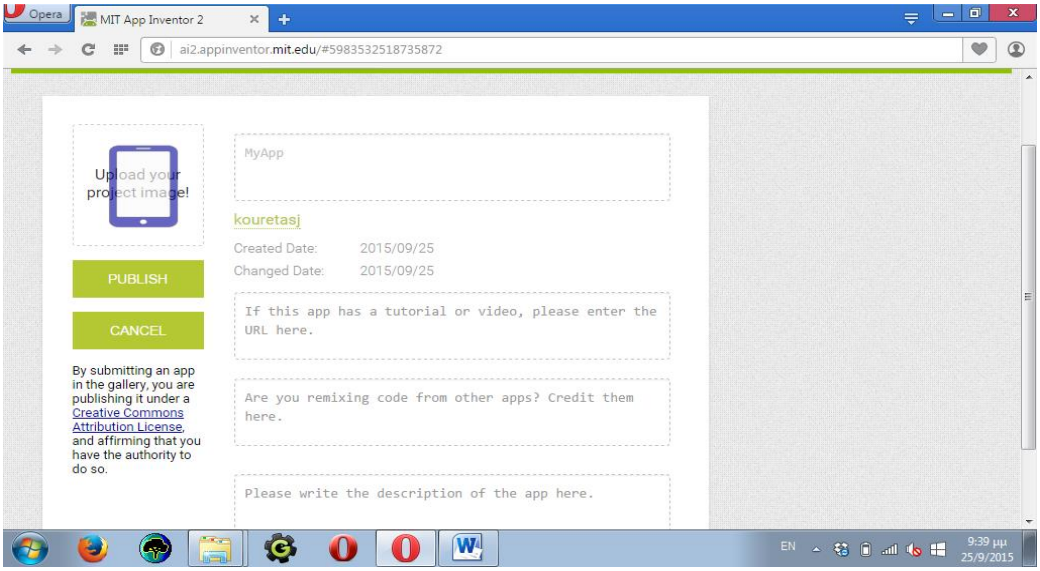
Εικόνα 16.2.1. Αναζήτηση έργων από χρήστες της κοινότητας.

Μπορούμε να επιλέξουμε ένα έργο και να το ανοίξουμε στο φυλλομετρητή μας και να δούμε πως είναι το γραφικά του στοιχεία αλλά και τα προγραμματιστικά πλακίδια που χρησιμοποιεί. Επίσης μπορούμε να αλλάξουμε το πρόγραμμα και να δημιουργήσουμε ένα νέο έργο με το νέο πρόγραμμα.



Εικόνα 16.2.2. Άνοιγμα ενός υπάρχοντος έργου και επεξεργασία του.

Τέλος το πιο σημαντικό σημείο είναι ότι μπορούμε με την επιλογή publishtogallery να δημοσιεύσουμε κι εμείς το δικό μας έργο δίνοντας παράλληλα περιγραφή, τίτλο και εικόνα. (βλέπε *Εικόνα 16.2.3*). Σημειώνεται ότι όταν χρησιμοποιούμε έτοιμα στοιχεία από ένα άλλο έργο αυτό θα πρέπει να αναφέρεται και στο νέο δημιουργημένο έργο ως αναφορά στο αρχικό έργο.



Εικόνα 16.2.3. Δημοσίευση έργου στην κοινότητα.

16.3 AppInventor tutorials

Στη σελίδα του AppInventor http://appinventor.mit.edu/explore/ai2/tutorials.html (01/10/2015), υπάρχουν οργανωμένα tutorial, στα οποία μπορεί κάποιος να ανατρέξει για να ξεκινήσει να μαθαίνει αλλά και για να προχωρήσει τον προγραμματισμό. Κάθε tutorial χαρακτηρίζεται από το βαθμό δυσκολίας (basic, intermediate, advanced) (βασικό, ενδιάμεσο, προχωρημένο).

Έτσι ο χρήστης ανάλογα το επίπεδο που κατέχει στον προγραμματισμό AppInventor μπορεί να ανοίξει κάποιο από αυτά.

Τα tutorial παρέχουν βήμα βήμα εξοικείωση με διάφορες ενέργειες με χρήση εικόνων έτσι ώστε να είναι πλήρως κατανοητά και εφαρμόσιμα.

Tutorial μπορεί επίσης να βρει κάποιος μέσω της αναζήτησης. Tutorial υπάρχουν και σε έντυπη μορφή βιβλίου στο εμπόριο.

Κεφάλαιο 17

Διαχείριση αισθητήρων και άλλων στοιχείων κινητού

17. Διαχείριση αισθητήρων και άλλων στοιχείων κινητού

Ενότητες Κεφαλαίου

- Διαχείριση οθόνης αφής
- Διαχείριση αισθητήρα κίνησης (επιταχυνσιόμετρου)
- Διαχείριση αισθητήρα θέσης
- Διαχείριση συσκευών κινητού (ηχείο, δόνηση, μικρόφωνο, κάμερα, ρολόι, επαφές)

17.1 Διαχείριση οθόνης αφής

Δημιουργούμε μια καινούρια οθόνη ονόματι paint. Στην Εικόνα 17.1.1 φαίνεται η μορφή της οθόνης . Περιέχει τα παρακάτω στοιχεία:

Έναν καμβά με ένα σχήμα μπάλας μέσα. Ο καμβάς καταλαμβάνει όλο το διαθέσιμο χώρο σε πλάτος και ύψος που του δίνεται δίνοντας στις ιδιότητες την τιμή FillParent στο μήκος και στο πλάτος.

Τρία κουμπιά ονόματι blue, red, green και ball με αντίστοιχα τα χρώματα. Το χρώμα σε ένα κουμπί μπορούμε να το ρυθμίσουμε από τις ιδιότητες. Για να μπορέσουμε να τοποθετήσουμε ομοιόμορφα τα κουμπιά χρησιμοποιούμε ένα στοιχείο το οποίο ονομάζεται HorizontalArrangement και βρίσκεται στην κατηγορία Layout.

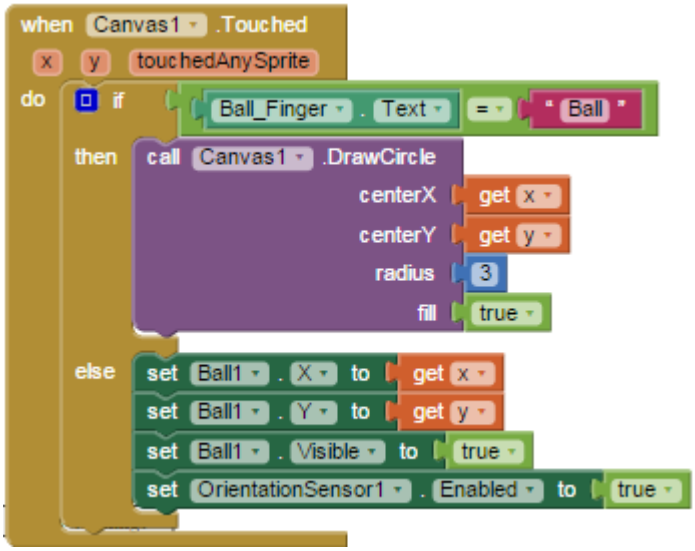
Επίσης υπάρχουν δύο κουμπιά, return to menu και take a picture τα οποία είναι και αυτά τοποθετημένα σε οριζόντια διάταξη.

Τέλος έχουμε τα μη ορατά αντικείμενα, επιταχυνσιόμετρο (Accelarator), κάμερα (camera1), αισθητήρας θέσης, και ένα αντικείμενο τύπου notifier.

Εικόνα 17.1.1 Η οθόνη paint.

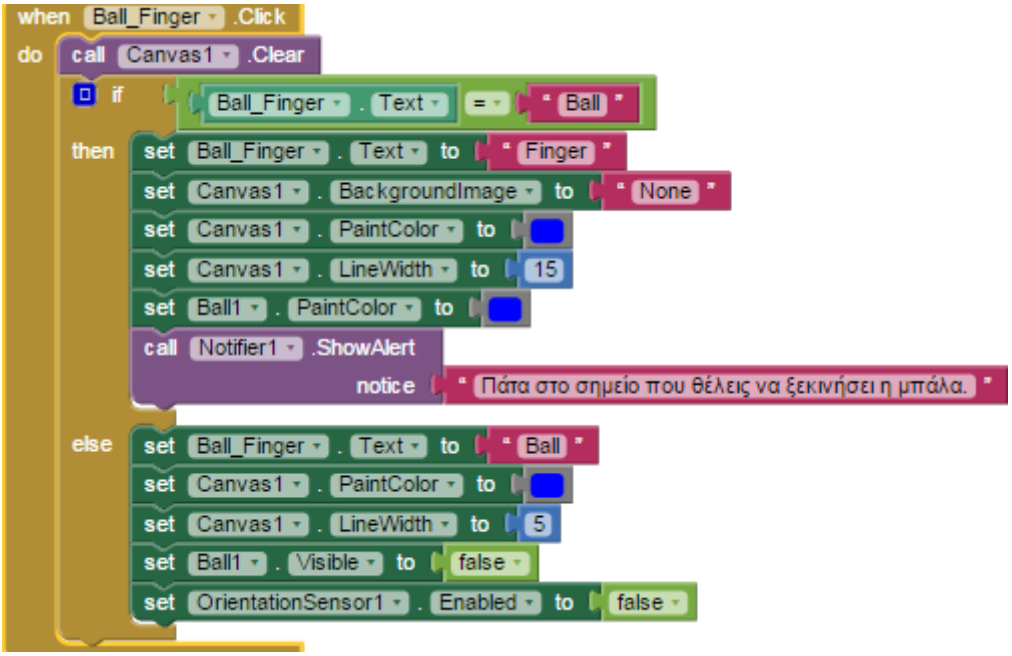
Εικόνα 17.1.2. Όταν πατηθεί κάποιο από τα κουμπιά.

Στην Εικόνα 17.1.2 φαίνονται τα προγραμματιστικά πλακίδια για το γεγονός του πατήματος ενός κουμπιού. Το ίδιο γίνεται και για το πράσινο κουμπί. Φαίνεται ότι με το πάτημα του κουμπιού ο καμβάς ζωγραφίζεται με το ίδιο χρώμα ενώ η μπάλα χρωματίζεται ομοίως.



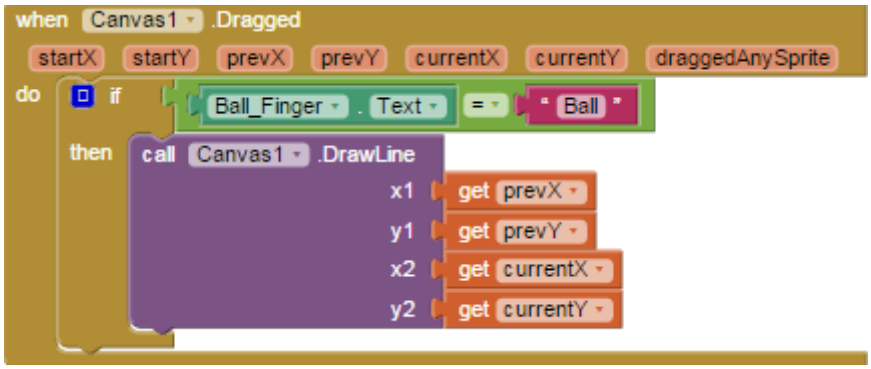
Εικόνα 17.1.3. Όταν πραγματοποιήσουμε αφή στον καμβά.

Όταν ακουμπήσουμε τον καμβά χωρίς να σύρουμε το δάκτυλό μας και στο μαύρο κουμπί γράφει Ball τότε δημιουργείται ένας κύκλος με διάμετρο 3 στο σημείο που θα κάνουμε αφή (βλέπε Εικόνα 17.1.3).



Εικόνα 17.1.4. Όταν πατήσουμε το μαύρο κουμπί.

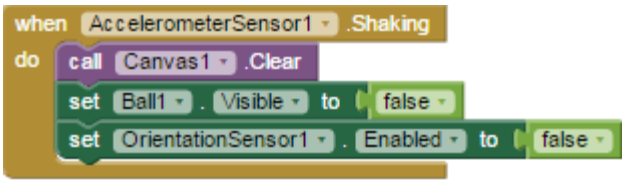
Η Εικόνα 17.1.4 δείχνει τι θα συμβεί αν πατηθεί το μαύρο κουμπί. Καταρχήν καθαρίζεται ο καμβάς από τυχόν προηγούμενη ζωγραφική και αν γράφει ball μετατρέπεται αυτόματα σε Finger μας προτρέπει με ένα μήνυμα του Notifier1 να πατήσουμε σε κάποιο σημείο της οθόνης για να ξεκινήσει η μπάλα. Στην περίπτωση που γράφει Finger τότε αυτόματα αλλάζει σε ball και θέτει το χρώμα που θα εμφανιστεί η γραμμή αν αρχίσουμε να σέρνουμε το δάκτυλό μας στη οθόνη ή ο κύκλος αν απλώς ακουμπήσουμε την οθόνη. Στην Εικόνα 17.1.5 φαίνεται το πρόγραμμα για το σχεδιασμό γραμμής πάνω στον καμβά.



Εικόνα 17.1.5. Όταν σύρουμε πάνω στον καμβά.

17.2 Διαχείριση αισθητήρα κίνησης (επιταχυνσιόμετρου)

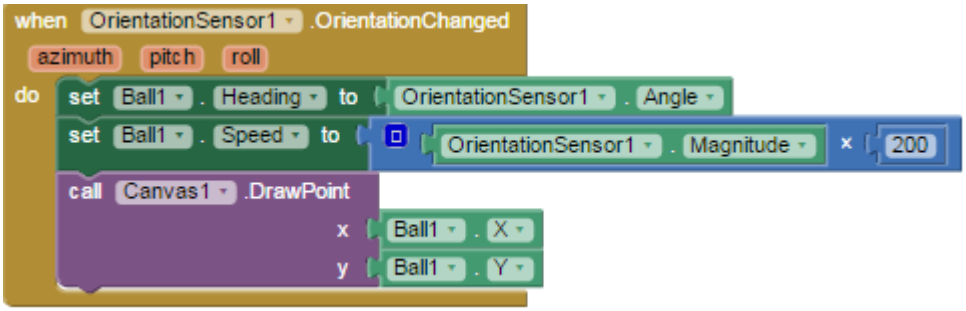
Η Εικόνα 17.2.1 δείχνει πώς μπορούμε να διαχειριστούμε το γεγονός της ενεργοποίησης του αισθητήρα (επιταχυνσιόμετρου). Εδώ έχουμε επιλέξει όταν κουνάμε γρήγορα το κινητό να καθαρίζει η οθόνη.



Εικόνα 17.2.1. Διαχείριση αισθητήρα κίνησης (επιταχυνσιόμετρου)

17.3 Διαχείριση αισθητήρα θέσης

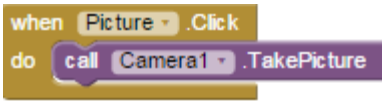
Στην Εικόνα 17.3.1 φαίνεται πώς όταν αλλάζει γωνία το κινητό, αναλόγως κινείται και η μπάλα αφήνοντας ίχνος σε σχήμα τετράγωνου. Η μπάλα θέτει ως προορισμό κίνησης (heading) τη γωνία που έχουμε στρέψει τη συσκευή. Ενώ πολλαπλασιάζει επί 200 την ταχύτητα ανάλογα με την κλίση, νομίζοντας ότι η μπάλα έχει βάρος και πέφτει.



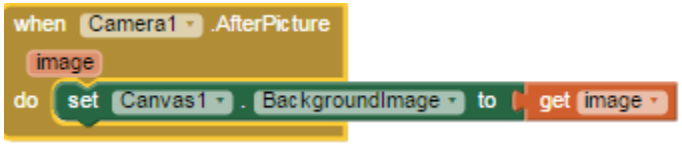
Εικόνα 17.3.1. Αισθητήρας θέσης.

17.4 Διαχείριση συσκευών κινητού (ηχείο, δόνηση, μικρόφωνο, κάμερα, ρολόι, επαφές)

Οι Εικόνες 17.4.1 και 17.4.2 δείχνουν την περίπτωση της χρήσης της κάμερας της συσκευής. Σε προηγούμενα κεφάλαια είδαμε για τη χρήση του ήχου, του μικροφώνου και του ρολογιού (στην οθόνη recmemo). Με την εντολή sound.vibrate για ένα συγκεκριμένο αριθμό δευτερολέπτων μπορούμε να κάνουμε τη συσκευή μας να δονηθεί. Προϋπόθεση είναι να έχουμε το αντικείμενο sound.



Εικόνα 17.4.1. Φωτογραφία από κάμερα.



Εικόνα 17.4.2. Τοποθέτηση στο φόντο της εφαρμογής μας.

Κεφάλαιο 18

Εξωτερική Επικοινωνία

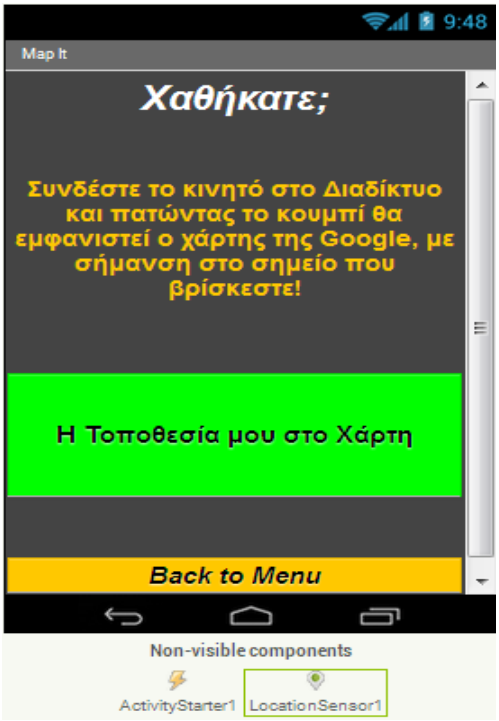
18. Εξωτερική Επικοινωνία

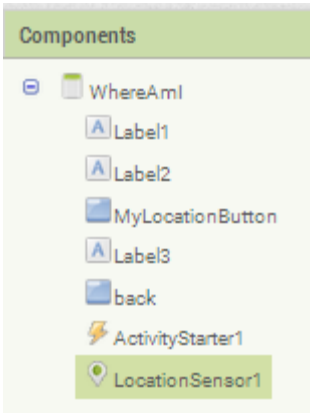
Ενότητες Κεφαλαίου

- Επικοινωνία με υπηρεσίες διαδικτύου
- Επικοινωνία μέσω διαδικτύου
- Υλοποιήσεις με μοντέλο πελάτη - εξυπηρετητή

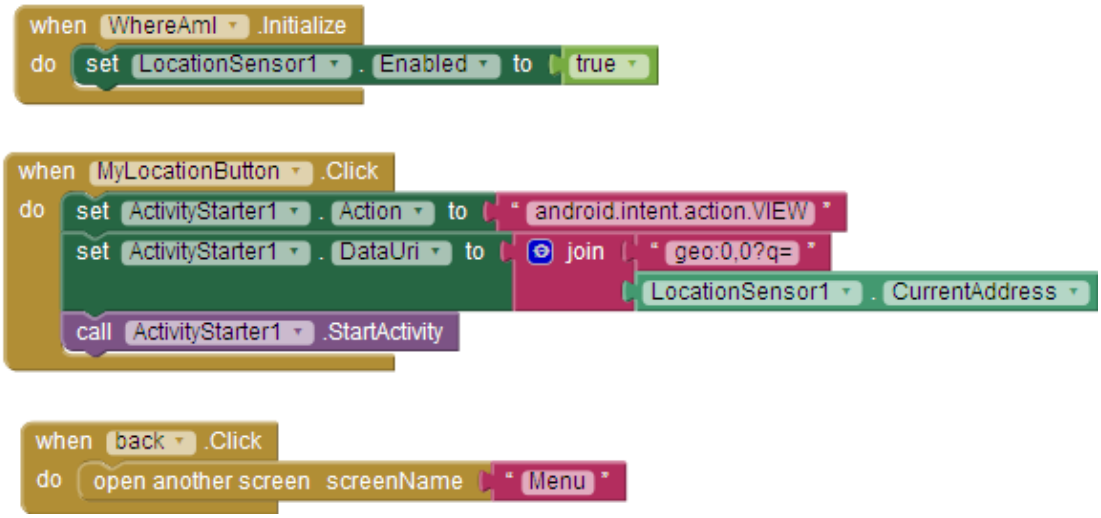
18.1 Επικοινωνία με υπηρεσίες διαδικτύου

Στην αρχική οθόνη μενού που έχουμε δημιουργήσει προσθέτουμε μια επιπλέον επιλογή κουμπί με κείμενο «που βρίσκομαι». Δημιουργούμε στο έργο μια καινούρια οθόνη ονόματι whereAmI, όπως φαίνεται παρακάτω μαζί με τα αντικείμενα.





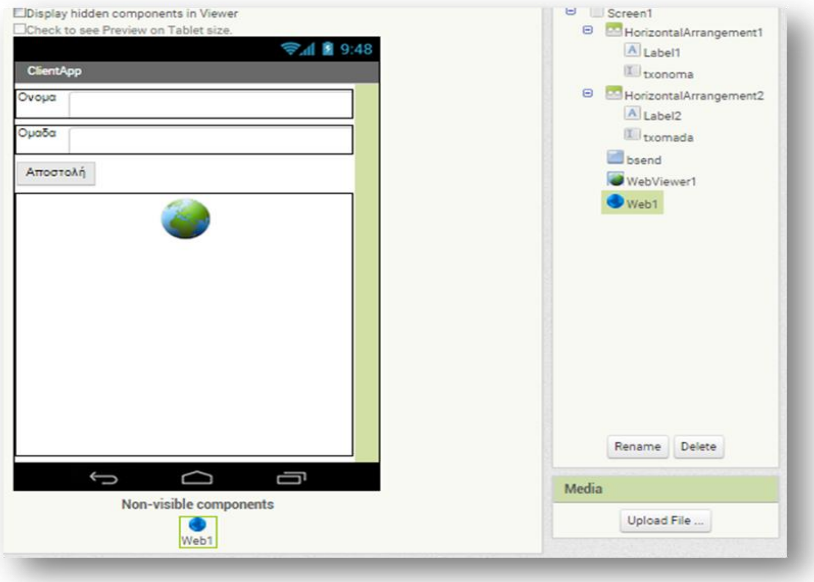
Παρατηρήστε το αντικείμενο ActivityStarter και το LocationSensor. Το ActivityStarter ξεκινά μια διεργασία του Android (android.intent.action.VIEW) ενώ το LocationSensor περιέχει στοιχεία για την τοποθεσία που βρίσκεται η συσκευή μας. Στη συνέχεια παρατίθενται τα σχετικά προγραμματιστικά πλακίδια. Όταν πατηθεί το «Η Τοποθεσία μου στο χάρτη» (MyLocationButton) τότε ενεργοποιείται το activity του Android παίρνοντας στοιχεία για τη γεωγραφική θέση από τον αισθητήρα θέσης (GPS), και μας εμφανίζει τη θέση μας στο χάρτη.



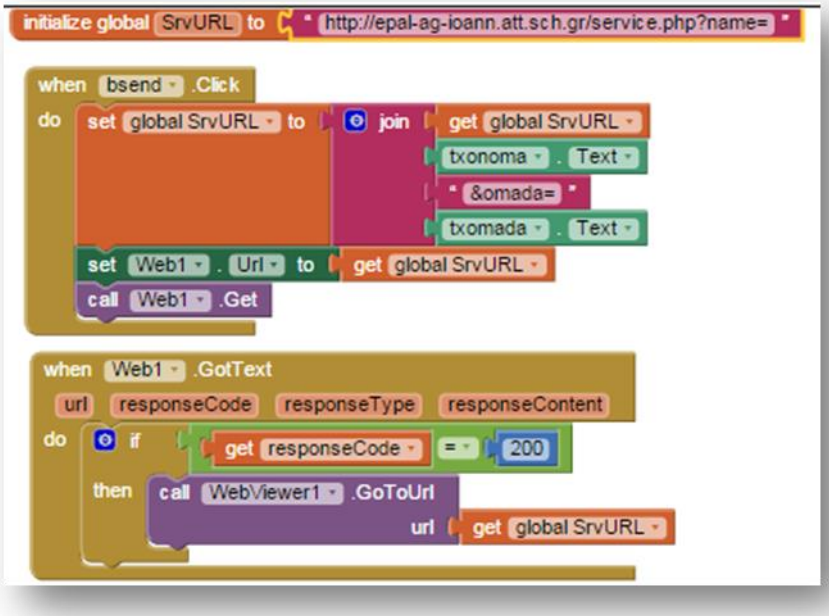
18.2 Επικοινωνία μέσω διαδικτύου

Το κλασσικό μοντέλο πελάτη – εξυπηρετητή χρησιμοποιείται κατά κόρο στις υπηρεσίες που παρέχονται στο διαδίκτυο. Σ' αυτό το μοντέλο ανάπτυξης εφαρμογών ανταλλάσσονται δεδομένα μεταξύ μιας εφαρμογή πελάτη – χρήστη και της εφαρμογής εξυπηρέτησης (υπηρεσίες εταιρειών ,οργανισμών κ.λπ.), όπως για παράδειγμα η online αγορά από κάποιο ηλεκτρονικό κατάστημα ή η παρακολούθηση του λογαριασμού της τράπεζας μέσω web banking κ.λπ. Το AppInventor έχει τη δυνατότητα να ανταλλάξει δεδομένα με ένα εξυπηρετητή όπου αυτός αναλαμβάνει να τα επεξεργαστεί και να μας επιστρέψει τα αποτελέσματα αυτής της ενέργειας αν χρειάζεται. Στη συνέχεια παρουσιάζεται ένα τέτοιο απλό παράδειγμα με μια εφαρμογή πελάτη στο κινητό μας μέσω του AppInventor στην οποία στέλνουμε το όνομά μας και την ομάδα που υποστηρίζουμε σε μια εφαρμογή κάπου στο διαδίκτυο, όπου αφού γίνει μια στοιχειώδης επεξεργασία των δεδομένων μας εμφανίζεται ένα μήνυμα.

Σε πρώτη φάση σχεδιάζουμε την οθόνη. Χρειαζόμαστε δύο label και δύο αντικείμενα τύπου textbox, τα οποία δέχονται κείμενο από το πληκτρολόγιο. Επίσης, από την κατηγορία userInterface επιλέγουμε το WebViewer, ένα στοιχείο με το οποίο μπορούμε να ενσωματώσουμε ένα στοιχειώδη περιηγητή στην εφαρμογή το οποίο υποκρύπτει στην πραγματικότητα μια ελαφριά έκδοση του Google chrome για φορητές συσκευές. Ο περιηγητής μας επιτρέπει να δούμε τα αποτελέσματα της επεξεργασίας των πληροφοριών που εκτελείται στον εξυπηρετητή αφού η υπηρεσία εκτελείται σε μια γλωσσά που υποστηρίζεται από τους φυλλομέτρησες όπως η PHP. Τέλος από την κατηγορία connectivity το Web ένα στοιχείο που μας επιτρέπει να επικοινωνήσουμε με έναν http server εκτελώντας εντολές Get σε ένα συγκεκριμένο URL. (βλέπε Εικόνα 18.2.1)



Εικόνα 18.2.1. Η οθόνη για την εφαρμογή



Εικόνα 18.2.2. Τα προγραμματιστικά πλακίδια για την εφαρμογή πελάτη - εξυπηρετητή.

Με το στοιχείο Web1 στέλνουμε σε συγκεκριμένο URL τις πληροφορίες που εισάγει ο χρήστης στα πλαίσια κειμένου με την εντολή Web1.Get. Και όταν παραλάβει κάποια απάντηση αφού ελέγξουμε ότι δεν έχει συμβεί κάποιο σφάλμα ενημερώνουμε το στοιχείο του περιηγητή.

Στο προγραμματιστικό περιβάλλον, Εικόνα 18.2.2, φαίνεται η αρχικοποίηση της μεταβλητής SrvURL με το url στο οποίο έχουμε ανεβάσει το αρχείο service.php. Συγκεκριμένα, για url γράφουμε http://epal-ag-ioann.att.sch.gr?service.php?name=.

Το αρχείο service.php φαίνεται παρακάτω.

```
<?php
header("Content-Type: text/html; charset=utf-8");
echo '<body style="background-color:yellow">';
echo '<b><font face="ARIAL" size="18">';
if (rand()&1==1)
 {
echo "Μπράβοείσαιπρωταθλητής".$_GET["name"]." ".$_GET["omada"]." Ομαδάρα!!! ";
 }
else
{
echo $_GET["name"]." είσαι ".$_GET["omada"].".Υποβιβάστηκες!!!";
}
?>
```

Στο παράδειγμά μας έχουμε ανεβάσει στο url http://epal-ag-ioann.att.sch.gr το σχετικό αρχείο.

Οπότε όταν πατήσουμε το κουμπί αποστολή (whenbsend.click) τότε ουσιαστικά δημιουργείται το url http://epal-ag-ioann.att.sch.gr?service.php?name=<το όνομα>&omada=<το όνομα ομάδας>. <Το όνομα> είναι αυτό που γράψαμε για όνομα και <το όνομα ομάδας> είναι αυτό που γράψαμε για όνομα ομάδας. Για παράδειγμα να δώσαμε για όνομα Yiannis και για όνομα ομάδας Olympiakos, τότε το url που δημιουργείται είναι:

http://epal-ag-ioann.att.sch.gr?service.php?name=Yiannis&omada=Olympiakos

Στην πλευρά του εξυπηρετητή με την χρήση της μεταβλητής $_GET["name"] και GET["omada"] χειριζόμαστε τις πληροφορίες που έχουν σταλεί με την εντολή get. Με τυχαίο τρόπο χρησιμοποιώντας την εντολή (rand()&1) που παράγει τυχαία μια τιμή μεταξύ των τιμών 1 ή 0 και επιλέγεται μεταξύ δύο διαθέσιμων μηνυμάτων το ένα και εμφανίζεται ως αποτέλεσμα της ιστοσελίδας της υπηρεσίας που παρέχει ο εξυπηρετητής το οποίο και εμφανίζεται στην οθόνη στο αντικείμενο webviewer της εφαρμογής oid.

18.3 Υλοποιήσεις με μοντέλο πελάτη – εξυπηρετητή και server-side προγραμματισμό σε php

Στη συνέχεια θα δούμε πώς μπορούμε να δημιουργήσουμε μια απλή εφαρμογή για συνομιλία (chat) στο κινητό μας. Η εφαρμογή μας έχει την οθόνη όπως φαίνεται στην Εικόνα 18.3.1. Περιέχει τα εξής στοιχεία.

Ένα label μαζί με το αντίστοιχο textbox για την εισαγωγή ονόματος.

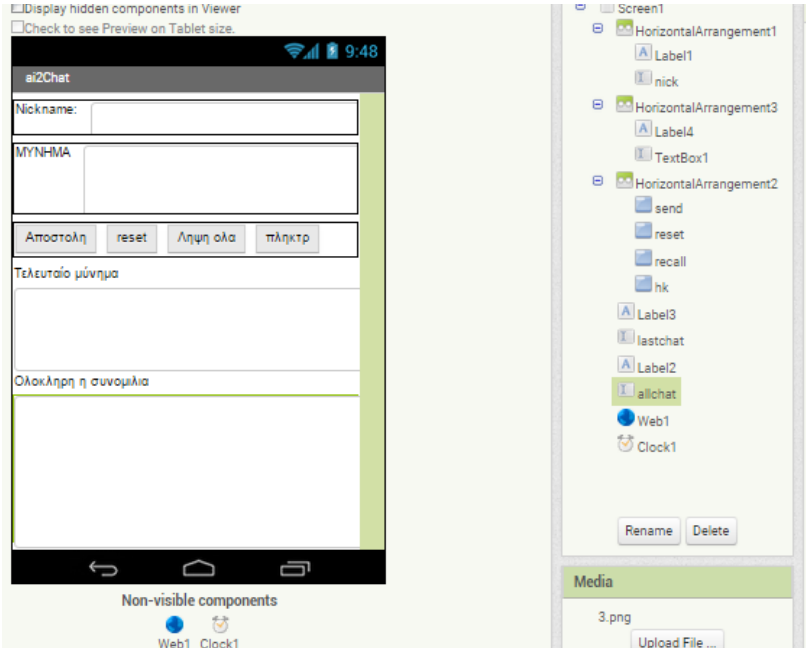
Ένα label μαζί με το αντίστοιχο textbox για την εισαγωγή του μηνύματος.

Τέσσερα κουμπιά.

Επιπλέον δύο textbox μαζί με label.

Ένα αντικείμενο μη ορατό τύπου web, για τη σύνδεσή μας στη σελίδα.

Ένα αντικείμενο μη ορατό τύπου clock.



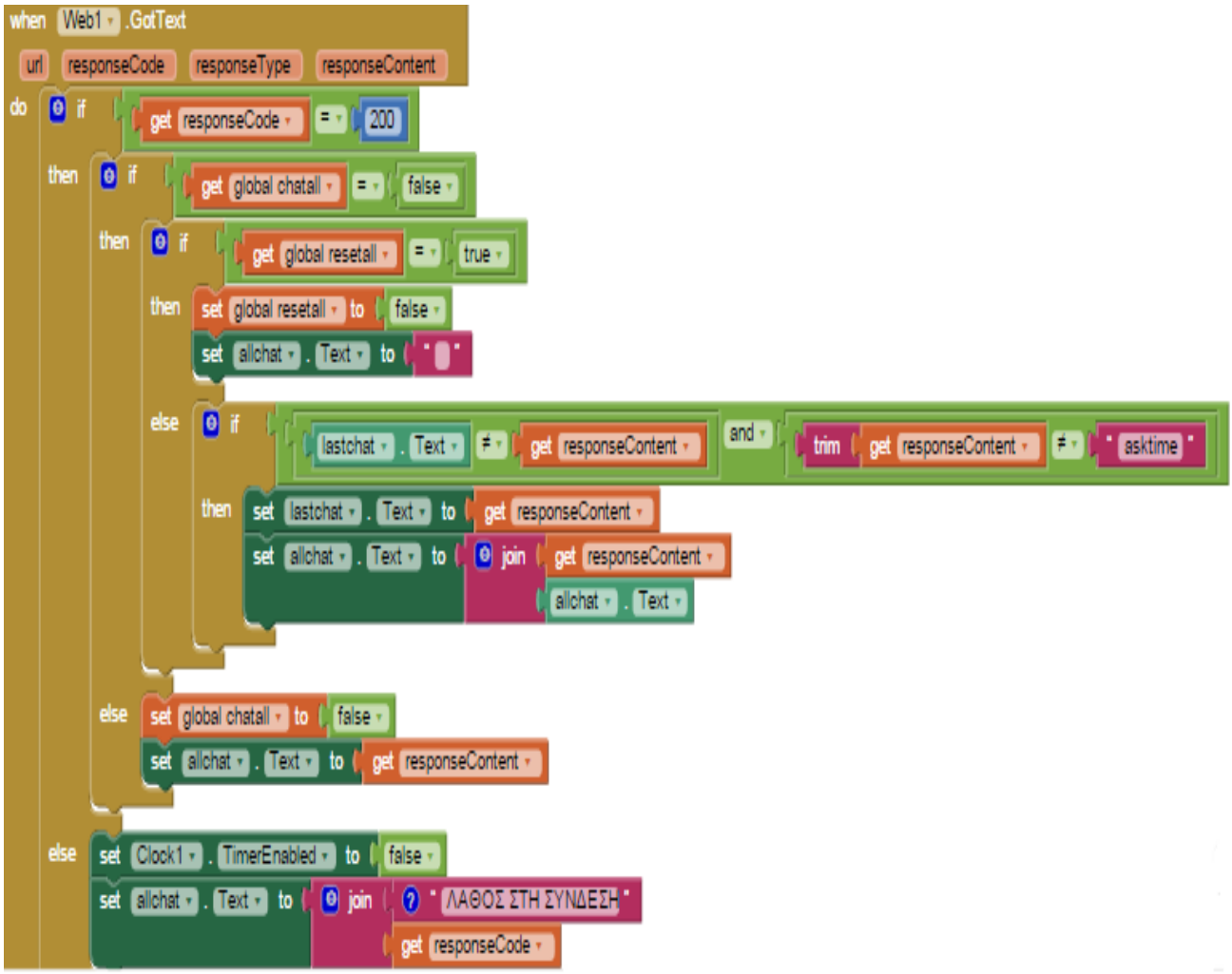
Εικόνα 18.3.1. Οθόνη εφαρμογής chat.

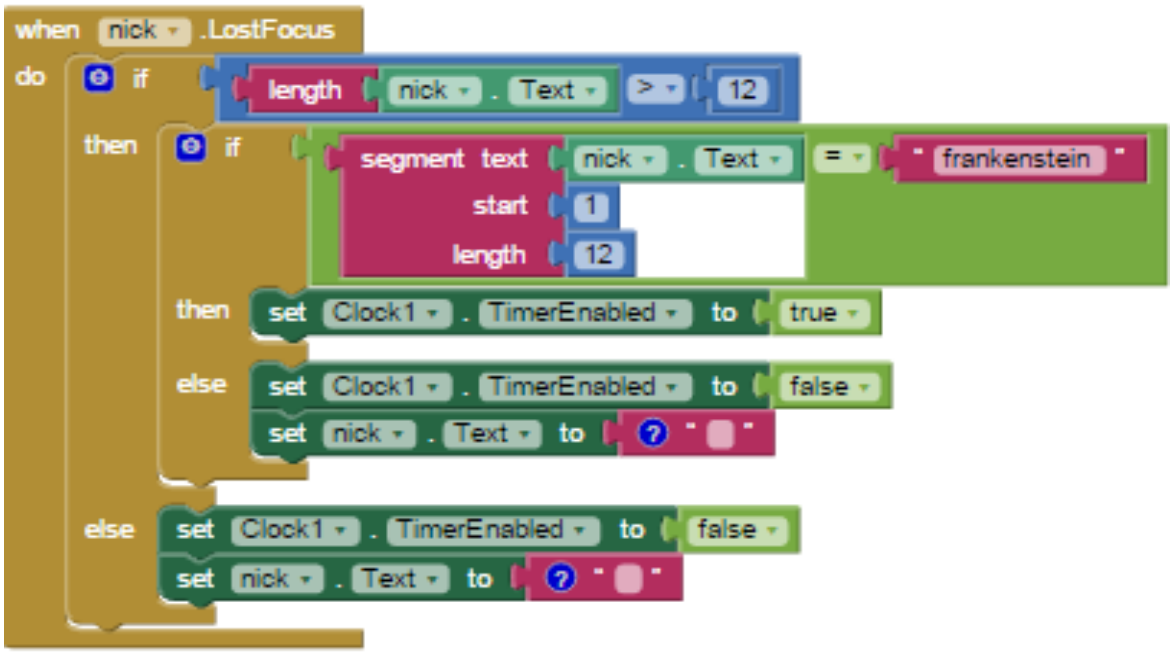
Στη συνέχεια αρχικοποιούμε τις μεταβλητές μας. Η επεξεργασία των δεδομένων που στέλνουμε πραγματοποιούνται με τον ίδιο τρόπο στο ίδιο url με τη χρήση του αρχείου chat.php το οποίο δίνεται παρακάτω.

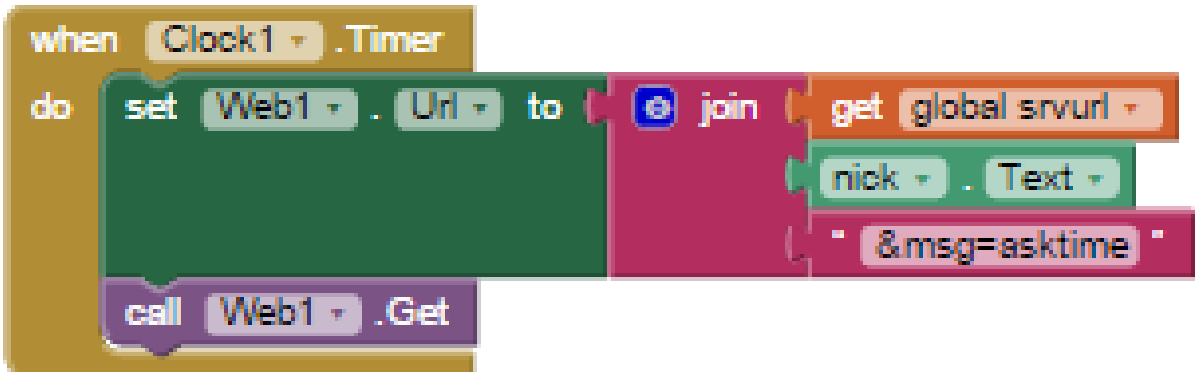
```
<?php
 $chfile = "chat.txt";
 $chnd = fopen($chfile, "r");
 $dat = fread($chnd , filesize($chfile));
fclose($chnd);
   $chfilea = "chata.txt";
 $chnda = fopen($chfilea, "r");
 $data = fread($chnda , filesize($chfilea));
fclose($chnda);
if (substr($_GET["nick"],0,12)=="frankenstein")  {
if ($_GET["msg"]=="rec!all")
        echo $data;
        elseif ($_GET["msg"]=="asktime")
        echo $dat."\n";
else {
        if ($dat==".") $dat="";
        if ($data==".") $data="";
    $dat=$_GET["nick"]."\n".$_GET["msg"];
         $chnd = fopen($chfile, "w") or die("Cannot open file:  ".$chfile);
fwrite($chnd, $dat);
fclose($chnd);
        echo $dat."\n";
         $data=$_GET["nick"]."\n".$_GET["msg"].$data."\n";
         $chnda = fopen($chfilea, "w") or die("Cannot open file:  ".$chfilea);
fwrite($chnda, $data);
fclose($chnda);
         }
        }
elseif ($_GET["nick"]=="frankadmin")  {
if ($_GET["msg"]=="rst") {
         $chnd = fopen($chfile, "w") or die("Cannot open file:  ".$chfile);
        fwrite($chnd, ".");
        fclose($chnd);
    $chnda = fopen($chfilea, "w") or die("Cannot open file:  ".$chfilea);
        fwrite($chnda, ".");
fclose($chnda);
        echo "reset done";
         }
}  else
echo "Δεν εισαι γνωστος ".$_GET["nick"]."\n" ;
?>
```

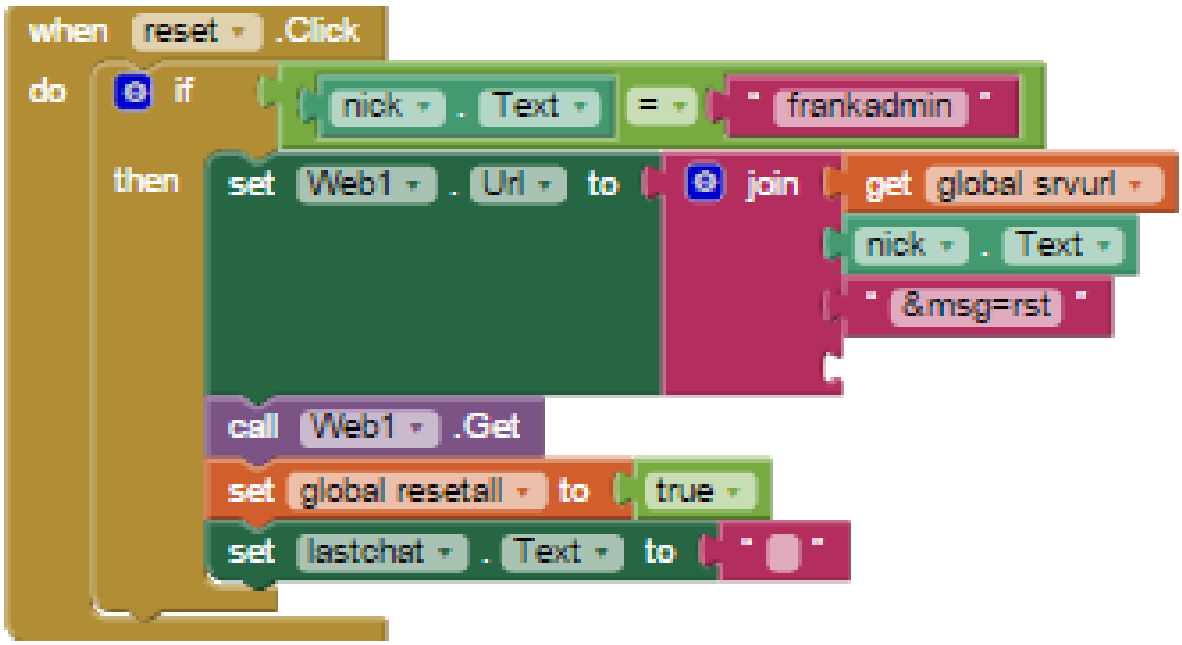
Η εφαρμογή μας λειτουργεί ως εξής: Αν δώσουμε σαν όνομα το frankestein μόνο τότε μπορούμε να στείλουμε μήνυμα. Επίσης μόνο ο χρήστης frankadmin μπορεί να σβήσει την οθόνη και να πραγματοποιήσει reset. Όταν ενεργοποιείται το clock στέλνεται η πληροφορία στο url και εκτελείται το chat.php αρχείο. Η εντολή .LostFocus σημαίνει ότι πατάμε να γράψουμε σε ένα textbox και στη συνέχεια όταν πατάμε πάνω σε άλλο σημείο πάνω στην οθόνη χάνεται η επαφή με το textbox, οπότε έχουμε .LostFocus. Το αντίστροφο είναι το .GotFocus.

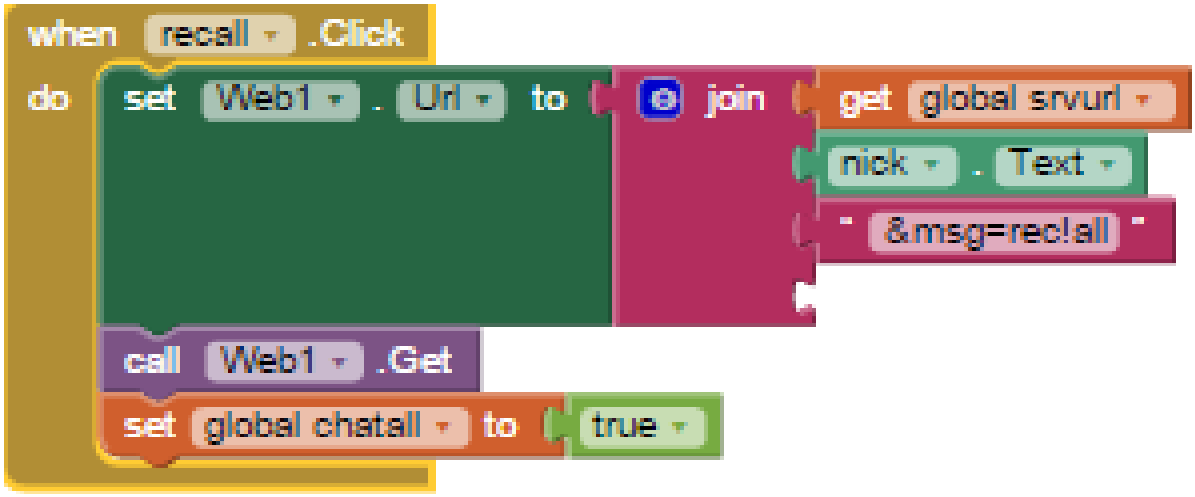
Τα προγραμματιστικά πλακίδια φαίνονται στις παρακάτω εικόνες.

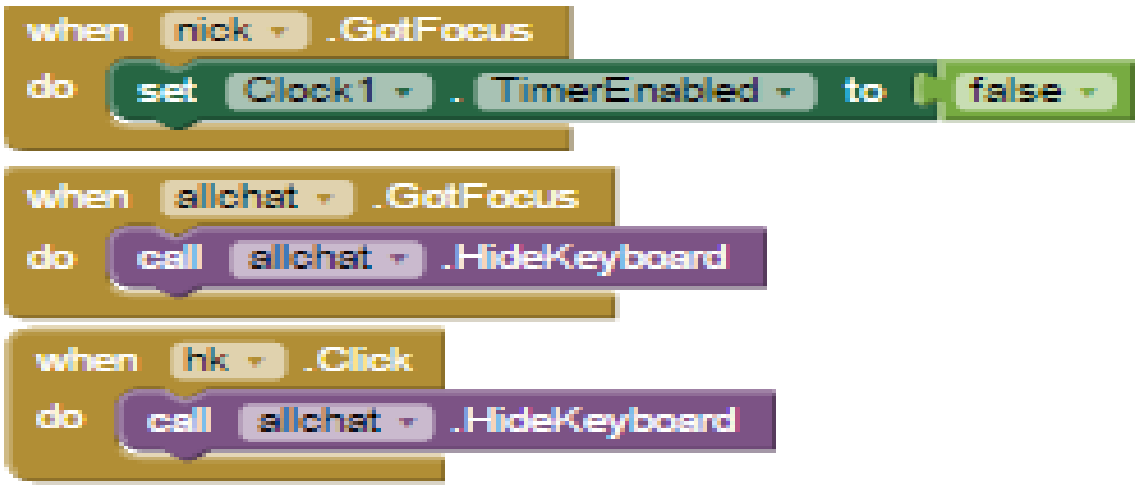

when Web1 .GotText
url responseCode responseType responseContent
do if get responseCode = 200
then if get global chatall = false
then if get global resetall = true
then set global resetall to false
set allchat . Text to " "
else if lastchat . Text ≠ get responseContent and trim get responseContent ≠ " asktime "
then set lastchat . Text to get responseContent
set allchat . Text to join get responseContent
allchat . Text
else set global chatall to false
set allchat . Text to get responseContent
else set Clock1 . TimerEnabled to false
set allchat . Text to join " ΛΑΘΟΣ ΣΤΗ ΣΥΝΔΕΣΗ "
get responseCode

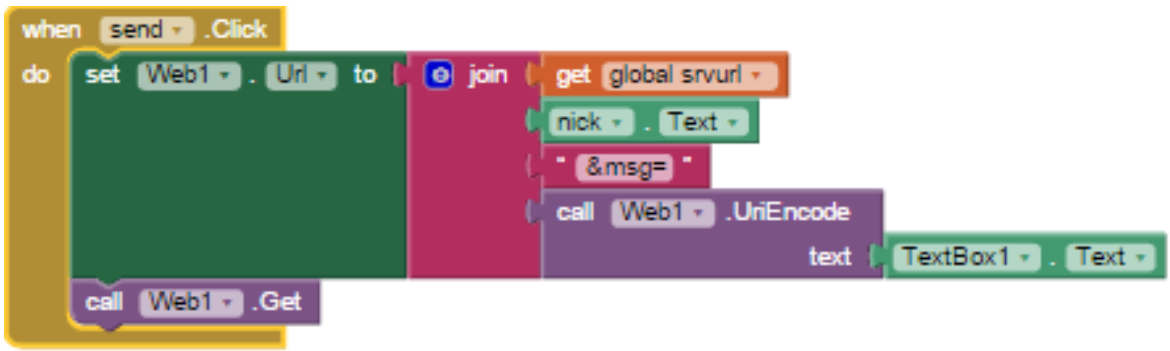

when nick .LostFocus
do if length nick . Text > 12
then if segment text nick . Text = " frankenstein "
start 1
length 12
then set Clock1 . TimerEnabled to true
else set Clock1 . TimerEnabled to false
set nick . Text to " "
else set Clock1 . TimerEnabled to false
set nick . Text to " "


when Clock1 .Timer
do set Web1 . Url to join get global srvurl
nick . Text
" &msg=asktime "
call Web1 .Get


when reset .Click
do if nick . Text = " frankadmin "
then set Web1 . Url to join get global srvurl
nick . Text
" &msg=rst "
call Web1 .Get
set global resetall to true
set lastchat . Text to " "


when recall .Click
do set Web1 . Url to join get global srvurl
nick . Text
" &msg=rec!all "
call Web1 .Get
set global chatall to true


when nick .GotFocus
do set Clock1 . TimerEnabled to false
when allchat .GotFocus
do call allchat .HideKeyboard
when hk .Click
do call allchat .HideKeyboard


when send .Click
do set Web1 . Url to join get global srvurl
nick . Text
" &msg= "
call Web1 .UriEncode
text TextBox1 . Text
call Web1 .Get

Κεφάλαιο 19

Οργάνωση και διαχείριση δεδομένων

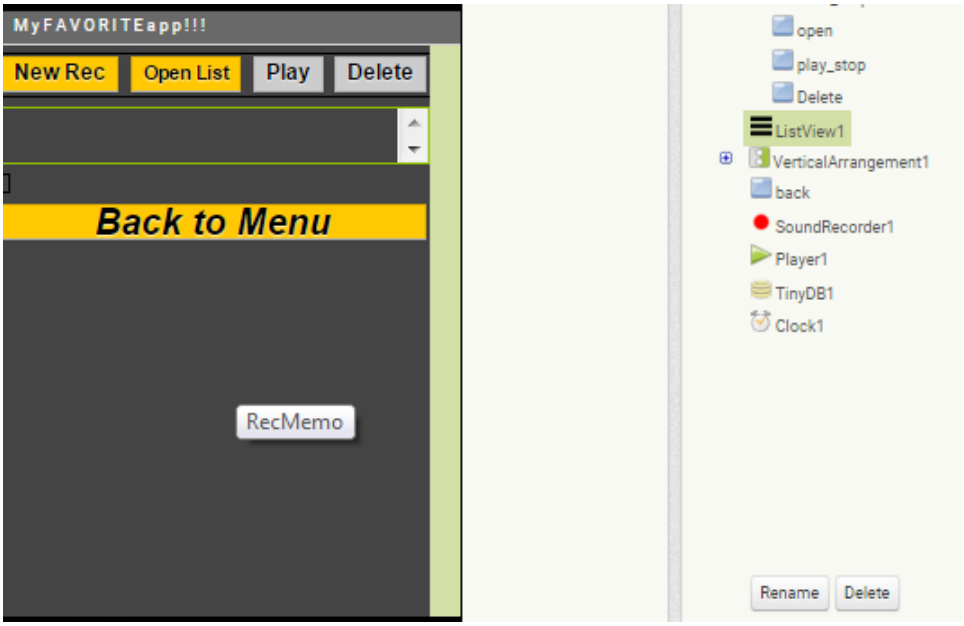
19. Οργάνωση και διαχείριση δεδομένων

Ενότητες Κεφαλαίου

- Διαχείριση λίστας
- Σχεδιασμός και δημιουργία απλής βάσης (TinyDB)
- Διαχείριση απλής βάσης (TinyDB)

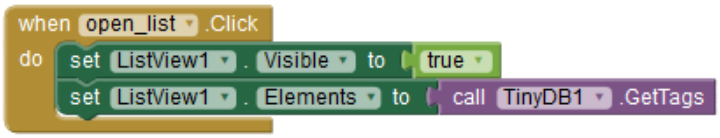
19.1 Διαχείριση λίστας

Είδαμε σε προηγούμενο κεφάλαιο την οθόνη recmemo. Πάμε σε αυτή την οθόνη και προσθέτουμε μία λίστα (Listview) από την κατηγορία userinterface. Στην παρακάτω εικόνα φαίνεται πως θα είναι η οθόνη.



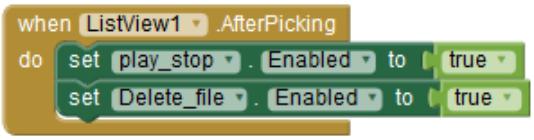
Εικόνα 19.1.1. Η οθόνη recmemo.

Whenopen_list.click: το επόμενο κουμπί θα εμφανίζει τις εγγραφές της βάσης μας σε μια λίστα. Γι' αυτό καθορίζουμε ως στοιχεία (***elements***) της λίστας τις ετικέτες (***getTags***) των εγγραφών της βάσης.



Εικόνα 19.1.2. Διαχείριση της λίστας.

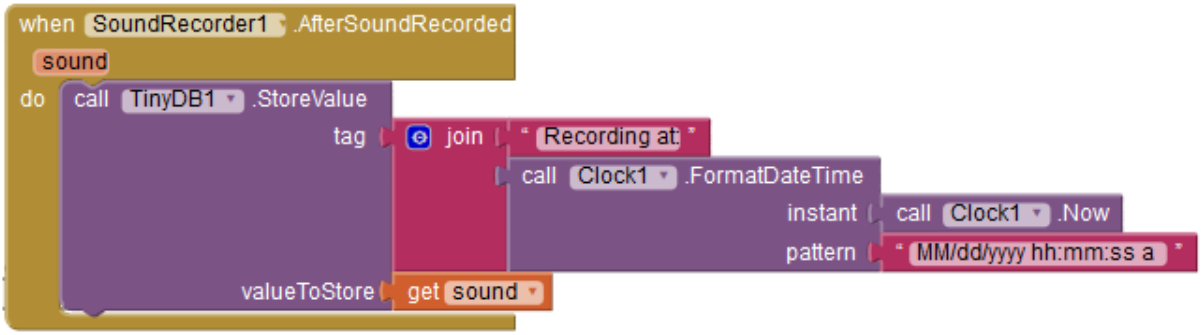
WhenListView1.AfterPicking: αυτή η ενέργεια συμβαίνει μόλις επιλέξουμε ένα στοιχείο (element) από τη λίστα, το οποίο αντιστοιχεί σε ηχητικό αρχείο της βάσης δεδομένων. Ενεργοποιεί τα δύο κουμπιά που θέλουμε να είναι ενεργά **μόνο** όταν υπάρχει επιλεγμένο αρχείο, είτε για να το αναπαράγουμε είτε για να το διαγράψουμε (και τα οποία είναι απενεργοποιημένα κατά την εκκίνηση της εφαρμογής).



Εικόνα 19.1.3. Επιλογή από τη λίστα.

19.2 Σχεδιασμός και δημιουργία απλής βάσης (TinyDB)

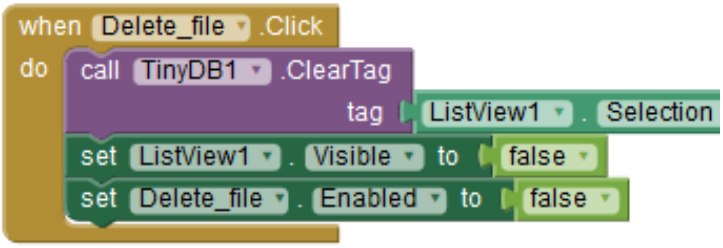
WhenSoundRecorder1.AfterSoundRecorded: σε κάθε stop του SoundRecorder, δημιουγείται ένα προσωρινό αρχείο ήχου, το οποίο πρέπει να το αποθηκεύσουμε στη βάση μας αλλιώς θα χαθεί. Καλούμε λοιπόν την *TinyDb.StoreValue* και ως τιμή προς αποθήκευση (***ValueToStore)*** ορίζουμε την μεταβλητή *Sound* της ενέργειας αυτής. Θέλουμε η αποθήκευση να γίνεται αυτόματα με μορφή ονόματος "Recordingat:" και την ημερομηνία/ώρα δημιουργίας του οπότε, η ετικέτα (***tag***) του αρχείου προς αποθήκευση θα έχει ένα κείμενο και την ημερομηνία. Για να επιτευχθεί αυτή η ένωση, δύο (ή περισσότερων) διαφορετικών τμημάτων κειμένου σε ένα, χρησιμοποιούμε την εντολή *Join* της ομάδας εντολών Text.



Εικόνα 19.2.1. Εισαγωγή δεδομένων στη βάση.

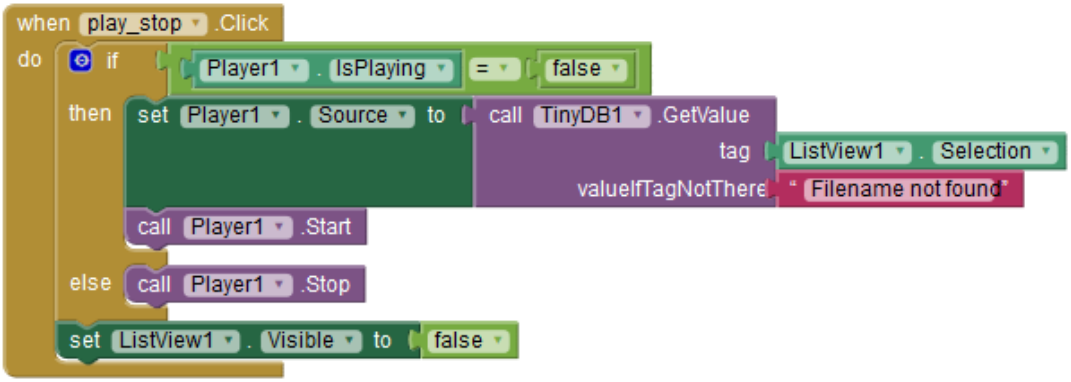
19.3 Διαχείριση απλής βάσης (TinyDB)

Whendelete_file.Click: Εδώ, καλούμε την λειτουργία ***ClearTag*** της βάσης, ώστε να διαγραφεί το επιλεγμένο αρχείο της λίστας (***selection***), από τη βάση δεδομένων. Προσοχή! Δεν διαγράφεται από τη μνήμη της συσκευής! Παράλληλα, κλείνουμε τη λίστα κι απενεργοποιούμε το κουμπί.



Εικόνα 19.3.1. Διαγραφή από τη βάση.

Whenplay_stop.Click: Αυτό το κουμπί το πατάμε όταν θέλουμε να ακούσουμε ή να σταματήσουμε το επιλεγμένο αρχείο. Γι' αυτό ελέγχουμε την ιδιότητα του Player, *IsPlaying*. Αν είναι *ψευδές*, δηλαδή δεν αναπαράγει, τότε συνδέουμε τον Player με το αρχείο της επιλογής μας (*source*) και ενεργοποιούμε την εντολή *Start*. Αλλιώς, αν είναι σε φάση αναπαραγωγής, δηλαδή *αληθές*, τότε απλά ενεργοποιούμε την εντολή *Stop*. Τέλος, κλείνουμε την λίστα.



Εικόνα 19.3.2. Αναπαραγωγή του ήχου.

Whenback.Click: Πατώντας το κουμπί αυτό ενεργοποιούμε την οθόνη με όνομα “Menu”, στην περίπτωση που χρησιμοποιούμε πολλές οθόνες και θέλουμε να επιστρέφει σε ένα αρχικό μενού. Αλλιώς μπορούμε να την αντικαταστήσουμε με “CloseApplication” εάν η εφαρμογή μας είναι ανεξάρτητη και θέλουμε να τερματίζει.

Εικόνα 19.3.3. Επιστροφή στο βασικό μενού.

Κεφάλαιο 20

Δημοσίευση μιας εφαρμογής

20. Δημοσίευση μιας εφαρμογής

Ενότητες Κεφαλαίου

- Ανοιχτή διάθεση του έργου
- Συμπλήρωση δηλωτικού (manifest)
- Διάθεση στο ευρύ κοινό (τρόποι)
- Δικαιώματα στον κώδικα
- Τεχνικές Προώθησης και Πωλήσεων ηλεκτρονικών προϊόντων

20.1 Ανοιχτή διάθεση του έργου

Η διάθεση ενός έργου μπορεί να είναι είτε δωρεάν είτε επί πληρωμή. Επίσης μια δωρεάν εφαρμογή μπορεί να διατίθεται ως έχει (στην APK της μορφή) ή με ανοιχτό τον κώδικα (ως aia project). Συχνά εφαρμογές που διαθέτουν ανοιχτό τον κώδικα είναι εκπαιδευτικές εφαρμογές ή εφαρμογές που δημιουργήθηκαν βασιζόμενες σε κώδικα από άλλες ανοιχτές εφαρμογές.

Μπορείτε να διαθέσετε ανοιχτή την εφαρμογή σας μέσω του AppInventorGallery που είδαμε σε προηγούμενα κεφάλαια ή προσφέροντας κατά περίπτωση πρόσβαση στο ανοιχτό έργο σας. Επίσης υπάρχουν διάφορες ιστοσελίδες και fora στα οποία μπορείτε να αναρτήσετε ένα aia project.

20.2 Συμπλήρωση δηλωτικού (manifest)

Με την ολοκλήρωση της εφαρμογής είναι σκόπιμη η συμπλήρωση του δηλωτικού της. Πλέον δεν θα τρέχει μέσω του AppInventor και είναι σωστό να προσδιορίζονται με σαφήνεια οι απαιτήσεις της. Όταν διαμορφώνουμε την εφαρμογή (αρχείο APK) αυτή πρέπει να περιλαμβάνει τα σχετικά στοιχεία.

Χρησιμοποιείστε την εμπειρία σας για να εντοπίσετε στο έργο σας που καταγράφονται τα στοιχεία αυτά, περιλαμβανομένων των εκδόσεων, πριν κλείσετε την εφαρμογή σας, δημιουργώντας ένα APK. Προσοχή, όμως, είναι αρκετά στοιχεία που δεν μπορείτε να αλλάξετε, όπως το ελάχιστο API στο οποίο απευθύνεται η εφαρμογή, αφού αυτά έχουν ρυθμιστεί κατά την ανάπτυξη του ίδιου του περιβάλλοντος.

20.3 Διάθεση στο ευρύ κοινό (τρόποι)

Είδαμε νωρίτερα πως γίνεται ανάρτηση στην AIGallery. Χιλιάδες χρήστες έχουν πρόσβαση εκεί, όπως και στα Fora του AppInventor2. Προφανώς αυτός είναι ένας τρόπος πρόσβασης στο ευρύ κοινό, ειδικά για μια ανοιχτή εφαρμογή. Απαιτεί την ανάρτηση κάποιων συμπληρωματικών πληροφοριών.

Με την προϋπόθεση ότι έχετε συμπληρωμένο δηλωτικό, την εφαρμογή σε μορφή APK και πρόκειται για πρωτότυπη εφαρμογή μπορείτε να αποκτήσετε ένα μοναδικό κλειδί για δημοσίευση στο google market. Αναζητείστε τον τρόπο στο Internet. Καλό είναι να έχετε συγκεντρώσει πληροφορίες και screenshots για την εφαρμογή σας, να έχετε καταγράψει τα δικαιώματα πρόσβασης που απαιτεί και να έχετε αποφασίσει αν τη διαθέτετε δωρεάν ή επί πληρωμή πριν προχωρήσετε. Όταν ολοκληρώσετε συγκρίνετε τις ενέργειές σας με όσα αναφέρονται στο κεφάλαιο 12 ειδικά σε σχέση με την ψηφιακή υπογραφή.

Ένας ακόμη τρόπος, και πάλι για πρωτότυπη εφαρμογή, είναι να αποκτήσετε έναν qr-code για μια πρωτότυπη εφαρμογή όπως εκείνος που χρησιμοποιήσαμε στο κεφάλαιο 13 για να εγκαταστήσουμε τον MIT emulator. Και πάλι απαιτούνται οι ενέργειες της προηγούμενης περίπτωσης. Αναζητείστε στο Internettον τρόπο για να αποκτήσετε τον κωδικό αυτό.

20.4 Δικαιώματα στον κώδικα

Όσο δουλεύουμε την εφαρμογή μας για εκπαιδευτικούς σκοπούς και την κρατάμε ανοιχτή, η κοινότητα ανάπτυξης του AppInventor δείχνει ανοχή στην επαναχρησιμοποίηση κώδικα. Είδαμε ότι στο Gallery συμπληρώνουμε και πληροφορίες για κώδικα που τυχόν χρησιμοποιήσαμε από άλλες εφαρμογές.

Όμως όταν μια εφαρμογή βγει στην αγορά οι όροι γίνονται πιο αυστηροί. Απαιτείται ένας βαθμός πρωτοτυπίας όπως ήδη αναφέραμε στην προηγούμενη παράγραφο και δεν μπορούμε να

αναπαράγουμε μια άλλη εφαρμογή ζητώντας αμοιβή, ειδικά αν η αρχική ήταν δωρεάν. Αντίστοιχα δεν υπάρχει λόγος να επιτρέψουμε την αντιγραφή του δικού μας, πρωτότυπου, κώδικα αν δεν έχουμε διαθέσει κάποια προηγούμενη έκδοση ανοιχτή οπότε τα πράγματα περιπλέκονται. Και πάλι ρίξτε μια ματιά στις σχετικές πηγές που αναφέρονται στο κεφάλαιο 12.

20.5 Τεχνικές Προώθησης και Πωλήσεων ηλεκτρονικών προϊόντων

Για την προώθηση της εφαρμογής σας στο google market (ή Play Store) είναι σκόπιμο να έχετε συγκεντρώσει το απαραίτητο για μια καλή παρουσίαση υλικό. Δείτε μια λίστα με το τι χρειάζεστε εδώ: http://developer.android.com/distribute/tools/launch-checklist.html.

Η σωστή κατηγοριοποίηση κατά την συμπλήρωση της παρουσίασης είναι πολύ σημαντική. Οι εφαρμογές που κυκλοφορούν στην ηλεκτρονική αγορά είναι πάμπολλες και είναι εύκολο να περάσει κάποια απαρατήρητη αν έχει μπει σε λάθος κατηγορία ή έχει ασαφή περιγραφή.

Υπάρχει ακόμα η επιλογή του in-gamebilling σε δωρεάν εφαρμογές, ιδιαίτερα παιχνίδια. Σε αυτές τις περιπτώσεις, ιδιαίτερα διαδεδομένες στα λεγόμενα browsergames, το παιχνίδι είναι δωρεάν αλλά υπάρχουν πληρωνόμενα αντικείμενα σε αυτό που παρέχουν πλεονεκτήματα έναντι των παιχτών που δεν επιλέγουν να τα πληρώσουν. Αυτή η πρακτική είναι αρκετά ενοχλητική και για αυτό το λόγο μόνο ιδιαίτερα ενδιαφέροντα παιχνίδια καταφέρνουν να την επιβάλουν.

Και για αυτά τα ζητήματα χρήσιμες είναι οι πηγές στο κεφάλαιο 12.

ΠΑΡΑΡΤΗΜΑ Α

Κατηγορίες Πόρων (πηγή Android Developer Guide: http://developer.android.com/guide/topics/resources/providing-resources.html#BestMatch)

Directory		Resource Type
`animator/`		XML files that define property animations.
`anim/`		XML files that define tween animations. (Property animations can also be saved in this directory, but the `animator/` directory is preferred for property animations to distinguish between the two types.)
`color/`		XML files that define a state list of colors. See Color State List Resource
`drawable/`		Bitmap files (`.png`, `.9.png`, `.jpg`, `.gif`) or XML files that are compiled into the following drawable resource subtypes: • Bitmap files • Nine-Patches (re-sizable bitmaps) • State lists • Shapes • Animation drawables • Other drawables See Drawable Resources.
`mipmap/`		Drawable files for different launcher icon densities. For more information on managing launcher icons with `mipmap/` folders, see Managing Projects Overview.
`layout/`		XML files that define a user interface layout. See Layout Resource.
`menu/`		XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. See Menu Resource.
`raw/`		Arbitrary files to save in their raw form. To open these resources with a raw`InputStream`, call `Resources.openRawResource()` with the resource ID, which is`R.raw.filename`. However, if you need access to original file names and file hierarchy, you might consider saving some resources in the `assets/` directory (instead of `res/raw/`). Files in `assets/` are not given a resource ID, so

		you can read them only using`AssetManager`.
`values/`		XML files that contain simple values, such as strings, integers, and colors. Whereas XML resource files in other `res/` subdirectories define a single resource based on the XML filename, files in the `values/` directory describe multiple resources. For a file in this directory, each child of the `<resources>` element defines a single resource. For example, a `<string>` element creates an `R.string` resource and a `<color>` element creates an `R.color` resource. Because each resource is defined with its own XML element, you can name the file whatever you want and place different resource types in one file. However, for clarity, you might want to place unique resource types in different files. For example, here are some filename conventions for resources you can create in this directory: • arrays.xml for resource arrays (typed arrays). • colors.xml for color values • dimens.xml for dimension values. • strings.xml for string values. • styles.xml for styles. See String Resources, Style Resource, and More Resource Types.
`xml/`		Arbitrary XML files that can be read at runtime by calling `Resources.getXML()`. Various XML configuration files must be saved here, such as a searchable configuration.

ΠΑΡΑΡΤΗΜΑ Β

Υποστηριζόμενοι προσδιορισμοί (πηγή Android Developer Guide: http://developer.android.com/guide/topics/resources/providing-resources.html#table2)

Configuration	Qualifier Values	Description

MCC and MNC	Examples:
	`mcc310`
	`mcc310-mnc004`
	`mcc208-mnc00`
	etc.
		The mobile country code (MCC), optionally followed by mobile network code (MNC) from the SIM card in the device. For example, `mcc310` is U.S. on any carrier, `mcc310-mnc004` is U.S. on Verizon, and `mcc208-mnc00` is France on Orange.
		If the device uses a radio connection (GSM phone), the MCC and MNC values come from the SIM card.
		You can also use the MCC alone (for example, to include country-specific legal resources in your application). If you need to specify based on the language only, then use the *language and region* qualifier instead (discussed next). If you decide to use the MCC and MNC qualifier, you should do so with care and test that it works as expected.
		Also see the configuration fields `mcc`, and `mnc`, which indicate the current mobile country code and mobile network code, respectively.

Language and region	Examples:
	`en`
	`fr`
	`en-rUS`
	`fr-rFR`
	`fr-rCA`
	etc.
		The language is defined by a two-letter ISO 639-1 language code, optionally followed by a two letter ISO 3166-1-alpha-2 region code (preceded by lowercase "`r`").
		The codes are *not* case-sensitive; the `r` prefix is used to distinguish the region portion. You cannot specify a region alone.
		This can change during the life of your application if the user changes his or her language in the system settings. See Handling Runtime Changes for information about how this can affect your application during runtime.
		See Localization for a complete guide to localizing your application for other languages.
		Also see the `locale` configuration field, which indicates the current locale.

Layout Direction	`ldrtl` `ldltr`	The layout direction of your application. `ldrtl` means "layout-direction-right-to-left". `ldltr` means "layout-direction-left-to-right" and is the default

		implicit value.
		
		This can apply to any resource such as layouts, drawables, or values.
		
		For example, if you want to provide some specific layout for the Arabic language and some generic layout for any other "right-to-left" language (like Persian or Hebrew) then you would have:
		
		`res/`
		
		`layout/`
		
		`main.xml (Default layout)`
		
		`layout-ar/`
		
		`main.xml (Specific layout for Arabic)`
		
		`layout-ldrtl/`
		`main.xml (Any "right-to-left" language, except for Arabic, because the "ar" language qualifier has a higher precedence.)`
		
		Note: To enable right-to-left layout features for your app, you must set`supportsRtl` to `"true"` and set `targetSdkVersion` to 17 or higher.
		
		Added in API level 17.

smallestWidth	`sw<N>dp`
	
	Examples:
	`sw320dp`
	`sw600dp`
	`sw720dp`
	etc.
		The fundamental size of a screen, as indicated by the shortest dimension of the available screen area. Specifically, the device's smallestWidth is the shortest of the screen's available height and width (you may also think of it as the "smallest possible width" for the screen). You can use this qualifier to ensure that, regardless of the screen's current orientation, your application has at least `<N>` dps of width available for its UI.
		
		For example, if your layout requires that its smallest dimension of screen area be at least 600 dp at all times, then you can use this qualifer to create the layout resources, `res/layout-sw600dp/`. The system will use these resources only when the smallest dimension of available screen is at least 600dp, regardless of whether the 600dp side is the user-perceived height or width. The smallestWidth is a fixed screen size characteristic of the device; **the device's smallestWidth does not change when the screen's**

		orientation changes. The smallestWidth of a device takes into account screen decorations and system UI. For example, if the device has some persistent UI elements on the screen that account for space along the axis of the smallestWidth, the system declares the smallestWidth to be smaller than the actual screen size, because those are screen pixels not available for your UI. Thus, the value you use should be the actual smallest dimension *required by your layout* (usually, this value is the "smallest width" that your layout supports, regardless of the screen's current orientation). Some values you might use here for common screen sizes: • 320, for devices with screen configurations such as: ○ 240x320 ldpi (QVGA handset) ○ 320x480 mdpi (handset) ○ 480x800 hdpi (high-density handset) • 480, for screens such as 480x800 mdpi (tablet/handset). • 600, for screens such as 600x1024 mdpi (7" tablet). • 720, for screens such as 720x1280 mdpi (10" tablet). When your application provides multiple resource directories with different values for the smallestWidth qualifier, the system uses the one closest to (without exceeding) the device's smallestWidth. *Added in API level 13.* Also see the `android:requiresSmallestWidthDp` attribute, which declares the minimum smallestWidth with which your application is compatible, and the `smallestScreenWidthDp` configuration field, which holds the device's smallestWidth value. For more information about designing for different screens and using this qualifier, see the Supporting Multiple Screens developer guide.

Available width	`w<N>dp`
	
	Examples:
	`w720dp`
	`w1024dp`
		Specifies a minimum available screen width, in `dp` units at which the resource should be used—defined by the `<N>` value. This configuration value will change when the orientation changes between landscape and portrait to match the current actual width.
		
		When your application provides multiple resource directories with different

	etc.	values for this configuration, the system uses the one closest to (without exceeding) the device's current screen width. The value here takes into account screen decorations, so if the device has some persistent UI elements on the left or right edge of the display, it uses a value for the width that is smaller than the real screen size, accounting for these UI elements and reducing the application's available space. *Added in API level 13.* Also see the `screenWidthDp` configuration field, which holds the current screen width. For more information about designing for different screens and using this qualifier, see the Supporting Multiple Screens developer guide.

Available height	`h<N>dp`
	
	Examples:
	`h720dp`
	`h1024dp`
	etc.
		Specifies a minimum available screen height, in "dp" units at which the resource should be used—defined by the `<N>` value. This configuration value will change when the orientation changes between landscape and portrait to match the current actual height.
		
		When your application provides multiple resource directories with different values for this configuration, the system uses the one closest to (without exceeding) the device's current screen height. The value here takes into account screen decorations, so if the device has some persistent UI elements on the top or bottom edge of the display, it uses a value for the height that is smaller than the real screen size, accounting for these UI elements and reducing the application's available space. Screen decorations that are not fixed (such as a phone status bar that can be hidden when full screen) are *not* accounted for here, nor are window decorations like the title bar or action bar, so applications must be prepared to deal with a somewhat smaller space than they specify.
		
		Added in API level 13.
		
		Also see the `screenHeightDp` configuration field, which holds the current screen width.
		
		For more information about designing for different screens and using this qualifier, see the Supporting Multiple Screens developer guide.

Screen size	`small`
	`normal`
	`large`
		• `small`: Screens that are of similar size to a low-density QVGA screen. The minimum layout size for a small screen is approximately 320x426 dp units. Examples are QVGA low-density and VGA high density.
		• `normal`: Screens that are of similar size to a medium-density

	`xlarge`	HVGA screen. The minimum layout size for a normal screen is approximately 320x470 dp units. Examples of such screens a WQVGA low-density, HVGA medium-density, WVGA high-density. • `large`: Screens that are of similar size to a medium-density VGA screen. The minimum layout size for a large screen is approximately 480x640 dp units. Examples are VGA and WVGA medium-density screens. • `xlarge`: Screens that are considerably larger than the traditional medium-density HVGA screen. The minimum layout size for an xlarge screen is approximately 720x960 dp units. In most cases, devices with extra-large screens would be too large to carry in a pocket and would most likely be tablet-style devices. *Added in API level 9.* **Note:** Using a size qualifier does not imply that the resources are *only* for screens of that size. If you do not provide alternative resources with qualifiers that better match the current device configuration, the system may use whichever resources are the best match. **Caution:** If all your resources use a size qualifier that is *larger* than the current screen, the system will **not** use them and your application will crash at runtime (for example, if all layout resources are tagged with the `xlarge` qualifier, but the device is a normal-size screen). *Added in API level 4.* See Supporting Multiple Screens for more information. Also see the `screenLayout` configuration field, which indicates whether the screen is small, normal, or large.
Screen aspect	`long` `notlong`	• `long`: Long screens, such as WQVGA, WVGA, FWVGA • `notlong`: Not long screens, such as QVGA, HVGA, and VGA *Added in API level 4.* This is based purely on the aspect ratio of the screen (a "long" screen is wider). This is not related to the screen orientation. Also see the `screenLayout` configuration field, which indicates whether the screen is long.
Round screen	`round` `notround`	• `round`: Round screens, such as a round wearable device • `notround`: Rectangular screens, such as phones or tablets *Added in API level 23.*

		Also see the `isScreenRound()` configuration method, which indicates whether the screen is round.
Screen orientation	`port` `land`	• `port`: Device is in portrait orientation (vertical) • `land`: Device is in landscape orientation (horizontal) This can change during the life of your application if the user rotates the screen. See Handling Runtime Changes for information about how this affects your application during runtime. Also see the `orientation` configuration field, which indicates the current device orientation.

UI mode	`car`
	`desk`
	`television`
	`appliancewatch`
		• `car`: Device is displaying in a car dock
		• `desk`: Device is displaying in a desk dock
		• `television`: Device is displaying on a television, providing a "ten foot" experience where its UI is on a large screen that the user is far away from, primarily oriented around DPAD or other non-pointer interaction
		• `appliance`: Device is serving as an appliance, with no display
		• `watch`: Device has a display and is worn on the wrist
		Added in API level 8, television added in API 13, watch added in API 20.
		For information about how your app can respond when the device is inserted into or removed from a dock, read Determining and Monitoring the Docking State and Type.
		This can change during the life of your application if the user places the device in a dock. You can enable or disable some of these modes using`UiModeManager`. See Handling Runtime Changes for information about how this affects your application during runtime.

Night mode	`night` `notnight`	• `night`: Night time • `notnight`: Day time *Added in API level 8.* This can change during the life of your application if night mode is left in auto mode (default), in which case the mode changes based on the time of day. You can enable or disable this mode using `UiModeManager`. SeeHandling Runtime Changes for information about how this affects your application during runtime.

Screen pixel density (dpi)	`ldpi`
	`mdpi`
	`hdpi`
		• `ldpi`: Low-density screens; approximately 120dpi.
		• `mdpi`: Medium-density (on traditional HVGA) screens; approximately 160dpi.

	`xhdpi`
	`xxhdpi`
	`xxxhdpi`
	`nodpi`
	`tvdpi`
		• `hdpi`: High-density screens; approximately 240dpi.
		• `xhdpi`: Extra-high-density screens; approximately 320dpi. *Added in API Level 8*
		• `xxhdpi`: Extra-extra-high-density screens; approximately 480dpi. *Added in API Level 16*
		• `xxxhdpi`: Extra-extra-extra-high-density uses (launcher icon only, see thenote in *Supporting Multiple Screens*); approximately 640dpi. *Added in API Level 18*
		• `nodpi`: This can be used for bitmap resources that you do not want to be scaled to match the device density.
		• `tvdpi`: Screens somewhere between mdpi and hdpi; approximately 213dpi. This is not considered a "primary" density group. It is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps and the system will scale them as appropriate. This qualifier was introduced with API level 13.
		
		There is a 3:4:6:8:12:16 scaling ratio between the six primary densities (ignoring the tvdpi density). So, a 9x9 bitmap in ldpi is 12x12 in mdpi, 18x18 in hdpi, 24x24 in xhdpi and so on.
		
		If you decide that your image resources don't look good enough on a television or other certain devices and want to try tvdpi resources, the scaling factor is 1.33*mdpi. For example, a 100px x 100px image for mdpi screens should be 133px x 133px for tvdpi.
		
		Note: Using a density qualifier does not imply that the resources are*only* for screens of that density. If you do not provide alternative resources with qualifiers that better match the current device configuration, the system may use whichever resources are the best match.
		
		See Supporting Multiple Screens for more information about how to handle different screen densities and how Android might scale your bitmaps to fit the current density.

Touchscreen type	`notouch` `finger`	• `notouch`: Device does not have a touchscreen. • `finger`: Device has a touchscreen that is intended to be used through direction interaction of the user's finger. Also see the `touchscreen` configuration field, which indicates the type of touchscreen on the device.
Keyboard availability	`keysexposed`	• `keysexposed`: Device has a keyboard available. If the device has

	`keyshidden` `keyssoft`	a software keyboard enabled (which is likely), this may be used even when the hardware keyboard is *not* exposed to the user, even if the device has no hardware keyboard. If no software keyboard is provided or it's disabled, then this is only used when a hardware keyboard is exposed. • `keyshidden`: Device has a hardware keyboard available but it is hidden*and* the device does *not* have a software keyboard enabled. • `keyssoft`: Device has a software keyboard enabled, whether it's visible or not. If you provide `keysexposed` resources, but not `keyssoft` resources, the system uses the `keysexposed` resources regardless of whether a keyboard is visible, as long as the system has a software keyboard enabled. This can change during the life of your application if the user opens a hardware keyboard. See Handling Runtime Changes for information about how this affects your application during runtime. Also see the configuration fields `hardKeyboardHidden` and`keyboardHidden`, which indicate the visibility of a hardware keyboard and and the visibility of any kind of keyboard (including software), respectively.

Primary text input method	`nokeys`
	`qwerty`
	`12key`
		• `nokeys`: Device has no hardware keys for text input.
		• `qwerty`: Device has a hardware qwerty keyboard, whether it's visible to the user or not.
		• `12key`: Device has a hardware 12-key keyboard, whether it's visible to the user or not.
		Also see the `keyboard` configuration field, which indicates the primary text input method available.

Navigation key availability	`navexposed` `navhidden`	• `navexposed`: Navigation keys are available to the user. • `navhidden`: Navigation keys are not available (such as behind a closed lid). This can change during the life of your application if the user reveals the navigation keys. See Handling Runtime Changes for information about how this affects your application during runtime. Also see the `navigationHidden` configuration field, which indicates whether navigation keys are hidden.

Primary non-touch navigation method	`nonav`
	`dpad`
	`trackball`
		• `nonav`: Device has no navigation facility other than using the touchscreen.
		• `dpad`: Device has a directional-pad (d-pad) for navigation.

	`wheel`	• `trackball`: Device has a trackball for navigation. • `wheel`: Device has a directional wheel(s) for navigation (uncommon). Also see the `navigation` configuration field, which indicates the type of navigation method available.

Platform Version (API level)	Examples:
	`v3`
	`v4`
	`v7`
	etc.
		The API level supported by the device. For example, `v1` for API level 1 (devices with Android 1.0 or higher) and `v4` for API level 4 (devices with Android 1.6 or higher). See the Android API levels document for more information about these values.

ΠΑΡΑΡΤΗΜΑ Γ

Αναλυτικές Οδηγίες για την εκτέλεση επιλεγμένων Δραστηριοτήτων στο Eclipse

ΚΕΦΑΛΑΙΟ 1

Δραστηριότητα 1.2

Κάντε κλικ στο κουμπί «New». Στο παράθυρο επιλογής βοηθού (wizard)που θα ανοίξει, επιλέξτε το φάκελο «Android» και επιλέξτε «Android Application Project». Προχωρήστε.

Στο παράθυρο «New Android Application» θα δείτε πεδία που πρέπει να ρυθμίσετε. Ανάμεσα σε αυτά το όνομα της εφαρμογής, του έργου και του πακέτου. Τα δύο πρώτα μπορούν να είναι τα ίδια ενώ το τρίτο καλό είναι να σχετίζεται. Επίσης θα σας επιτρέψει να επιλέξετε το SDK της ελάχιστης προς τα πίσω συμβατότητας, το SDK στόχο και το SDK με το οποίο θα γίνει η μετάφραση. Τα δύο τελευταία μπορούν να είναι τα ίδια όπως και το πρώτο αν δεν σας ενδιαφέρει να παίζει η εφαρμογή σας σε άλλες εκδόσεις. Τέλος μπορείτε να επιλέξετε την εμφάνιση (θέμα) της εφαρμογής σας.

Φυσικά μπορείτε να αφήσετε τις επιλογές SDK ως έχουν. Αργότερα θα μπορείτε να χρησιμοποιήσετε την εμπειρία σας για να επιλέγετε τις σχετικές ρυθμίσεις ώστε να προσφέρουν καλύτερη ισορροπία μεταξύ της παρεχόμενης εμπειρίας και της ευρύτητας των συσκευών (και άρα των εκδόσεων) στις οποίες θα τρέχει η εφαρμογή σας. Προχωρήστε.

Στην επόμενη οθόνη «Configure Project» επιλέξτε να δημιουργηθεί εικονίδιο εκκίνησης (launcher icon) για την εφαρμογή σας, μια δραστηριότητα (activity) και να δημιουργηθεί το έργο σας στο workspace σας. Προχωρήστε.

Στο παράθυρο «Launcher Icon» μπορείτε να επιλέξετε το εικονίδιο εκκίνησης και να ρυθμίσετε τις ιδιότητες εμφάνισης. Προχωρήστε.

Στο παράθυρο «Create Activity» μπορείτε να ρυθμίσετε τη λειτουργία του ενιαίου παράθυρου διεπαφής της εφαρμογής σας. Αφήστε επιλεγμένη τη δημιουργία («Create Activity») και κατά προτίμηση επιλέξτε « Blank Activity». Προχωρήστε.

Στην επόμενη οθόνη «Blank Activity» θα δείτε μια επισκόπηση των λεπτομερειών της (κύριας) Δραστηριότητας του έργου πριν από τη δημιουργία του. Μπορείτε να αλλάξετε τα ονόματα των δραστηριοτήτων και τη διάταξη, αλλά αυτό θα δυσκολέψει ίσως επόμενες εργαστηριακές δραστηριότητες. Ολοκληρώστε και θα δημιουργηθεί το έργο σας.

Το Eclipse θα δημιουργήσει το έργο σας, ανοίγοντας το αρχείο δραστηριότητας. Μην ανησυχείτε πάρα πολύ για το περιεχόμενο των αρχείων. Ρίξτε μόνο μια ματιά στον κατάλογο (directory) του έργου. Επιβεβαιώστε ότι υπάρχουν τα στοιχεία που αναφέραμε στην παράγραφο 1.1

ΚΕΦΑΛΑΙΟ 3

Δραστηριότητα 3.1

Διαλέξτε έναν τύπο σχεδίασης. Ο απλούστερος είναι ο λεγόμενος LinearLayout για προφανείς λόγους: τακτοποιεί τις όψεις που θέλουμε να φαίνονται είτε σε οριζόντια είτε σε κατακόρυφη διάταξη. Με οριζόντιο προσανατολισμό τοποθετεί όψεις από αριστερά προς τα δεξιά

ενώ με κατακόρυφο από πάνω προς τα κάτω. Για να το κάνει αυτό το στοιχείο σχεδίασης χρησιμοποιεί τα χαρακτηριστικά/ παραμέτρους layout_width και layout_height.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

</LinearLayout>
```

Για να προσθέσετε ένα νέο χαρακτηριστικό, προσθέστε μια νέα γραμμή μετά από τον ορισμό του «layout_height». Τα χαρακτηριστικά εισάγονται πληκτρολογώντας πρώτα «android:». Καθώς πληκτρολογείτε την άνω κάτω τελεία, το Eclipse θα εμφανίσει ένα κατάλογο σχετικών χαρακτηριστικών. Μπορείτε είτε να συνεχίσετε να πληκτρολογείτε για να περιορίσετε τη λίστα προτάσεων είτε με κύλιση να επιλέξετε ένα με το ποντίκι σας. Επιλέξτε το χαρακτηριστικό «android: gravity». Υπάρχουν πολλά ακόμα χαρακτηριστικά όπως φαίνεται στη λίστα.

Εισάγετε «center_horizontal» ως τιμή για την gravity ώστε όλα τα στοιχεία που περιλαμβάνονται στο σχεδιαστικό μας στοιχείο να κεντράρονται ως προς την οριζόντια διάστασή του:

```
android:gravity="center_horizontal"
```

Ας προσθέσουμε τώρα κάποιες όψεις (Views) στο σχεδιαστικό στοιχείο μας. Όπως είπαμε οι όψεις είναι τα ορατά στοιχεία της διεπαφής χρήστη. Στον κώδικά μας οι όψεις τοποθετούνται στις γραμμές μεταξύ του τελευταίου χαρακτηριστικού (android:orientation="vertical" >) και του κλεισίματος του σχεδιαστικού στοιχείου (</LinearLayout>). Αν ξεκινήσουμε σε κάποια από αυτές τις γραμμές πληκτρολογώντας «<» το Eclipse και πάλι θα παρουσιάσει λίστα διαθέσιμων κατάλληλων στοιχείων για να επιλέξουμε.

Επιλέγοντας «TextView» θα εμφανιστεί ο ακόλουθος κώδικας: <TextView />. Οι όψεις συνήθως δεν χρειάζονται διαφορετικά στοιχεία για το κλείσιμό τους, αρκεί ο χαρακτήρας «/».

Δώστε στη νέα όψη μας τη δυνατότητα να προσαρμόζεται στο μέγεθος του περιεχομένου της δίνοντας τόσο στο χαρακτηριστικό layout_width όσο και στο layout_height την τιμή «wrap_content» πληκτρολογώντας «android:» και επιλέγοντας από τη λίστα προτάσεων. Πρόσθεσε ένα κείμενο όπως το «Hello there» στα χαρακτηριστικά με τον ίδιο τρόπο. Ο κώδικας της όψης κειμένου σου θα είναι τώρα κάπως έτσι:

```
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    android:text="Hello there"/>
```

Σώστε το αρχείο.

Λογικά θα λάβετε μια προειδοποίηση (warning) του τύπου «Hardcoded string ... should use @string resource». Μπορείτε αν θέλετε να την αγνοήσετε όμως ο λόγος που εμφανίζεται είναι ότι το Eclipse θεωρεί προτιμότερο οι συμβολοσειρές (string) που χρησιμοποιούμε να ορίζονται στους πόρους δεδομένων ως τιμές. ΑΝ θέλετε να το διορθώσετε, πράγμα που θα εκτιμήσετε όταν ο κώδικάς σας μεγαλώσει κάπως, βρείτε στον υποκατάλογο res/values το αρχείο strings.xm και ανοίξτε το από την κατάλληλη ετικέτα.

Μέσα στο αρχείο, το Eclipse έχει ήδη προσθέσει κάποιες βασικές συμβολοσειρές. Πριν το κλείσιμο (εγγραφή « </resources> ») προσθέστε μια καινούρια με όνομα π.χ. «hello» και τιμή την συμβολοσειρά «Hello there». Ουσιαστικά θα προσθέσουμε κάτι σαν :

<string name="hello">Hello there</string>

Τώρα μπορούμε να χρησιμοποιήσουμε τη συμβολοσειρά μας όπου θέλουμε αναφερόμενοι σε αυτή με το όνομα «hello». Αν επιστρέψουμε στην όψη μας μπορούμε να τροποποιήσουμε την τελευταία ιδιότητα ως εξής:

android:text="@string/hello"

Αν σώσουμε πάλι η προειδοποίηση δεν θα εμφανιστεί πλέον.

Θα ολοκληρώσουμε το σχεδιαστικό μας στοιχείο προσθέτοντας ένα κουμπί (button). Πρώτα όμως ας πάμε να προσθέσουμε στο αρχείο res/values/strings.xml το κείμενο που θα φαίνεται πάνω στο κουμπί μας με τη γραμμή:

<string name="click">Click Me!</string>

Επιστρέφοντας στο σχεδιαστικό μας στοιχείο και χρησιμοποιώντας μόνο τα χαρακτηριστικά που ήδη έχουμε χρησιμοποιήσει για το textView φτιάξτε ένα Button. Ο κώδικας θα μοιάζει κάπως έτσι:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/click" />
```

Αποθηκεύστε.

Δραστηριότητα 3.4

Στο κύριο αρχείο διάταξης στο Eclipse επιλέξτε την ετικέτα επιμελητή XML . Εντοπίστε τον κώδικα που γράψατε για το κουμπί (Button) και προσθέστε το χαρακτηριστικό ID χρησιμοποιώντας την σύνταξη που ακολουθεί και μετά αποθηκεύστε:

android:id="@+id/myButton"

Πηγαίνετε στο κύριο αρχείο δραστηριότητας της εφαρμογής σας και δημιουργείστε μια γραμμή κάτω από την δήλωση της κλάσης. Είναι η πρώτη γραμμή και θα πρέπει να γράφει κάτι όπως:

public class MainActivity extends Activity {

Προσθέστε τη δήλωση μεταβλητής:

private Button theButton;

Το Eclipse, αν δεν έχετε ήδη εισάγει τον τύπο Button θα διαμαρτυρηθεί με κάποιο μήνυμα του τύπου «Button cannot be resolved to a type». Για να εισάγουμε τον έτοιμο τύπο Button του Android, περνάμε το ποντίκι πάνω από τη λέξη Button και το Eclipse δίνει και πάλι μια λίστα από προτάσεις. Επιλέξτε «Import 'Button' (android.widget)».

Επιστρέψτε στη σχεδίαση όπου μπορείτε να ανακτήσετε μια αναφορά στην αντίστοιχη όψη και να την αποθηκεύσετε στη μεταβλητή που έχετε ήδη δημιουργήσει. Στη μέθοδο onCreate στη δραστηριότητα (Activity) αμέσως μετά τη γραμμή όπου έχετε ορίσει τη διάταξη:

setContentView(R.layout.activity_main);

Προσθέστε τη γραμμή ανάκτησης του Button ως εξής:

theButton = (Button)findViewById();

Μέσα στις παρενθέσεις του «findViewById()» πληκτρολογήστε «R.». Το Eclipse θα εμφανίσει, για άλλη μια φορά, μια λίστα πιθανών τύπων πόρων. Επιλέξτε «id». Πληκτρολογήστε άλλη μία τελεία. Τώρα το Eclipse θα εμφανίσει μια λίστα υπαρχουσών τιμών ID. Επιλέξτε αυτήν που έχουμε δώσει στο Button («myButton»).

Έχετε τώρα αναθέσει την αναφορά στην όψη button στη νέα μεταβλητή που δημιουργήσατε διακρίνοντας την όψη χρησιμοποιώντας την ταυτότητά της (ID).

Δραστηριότητα 3.5

Πρώτα πρέπει να ορίσετε πώς η κλάση Activity θα υλοποιήσει τη συγκεκριμένη διεπαφή. Αυτό γίνεται επεκτείνοντας την αρχική δήλωση της κλάσης με τη διατύπωση «implements OnClickListener». Έτσι τώρα η πρώτη γραμμή του αντίστοιχου αρχείου θα μοιάζει κάπως έτσι:

```
public class MainActivity extends Activity implements OnClickListener {
```

Το Eclipse θα διαμαρτυρηθεί και πάλι για τον άγνωστο τύπο OnClickListener. Όπως πριν, διορθώστε το λάθος επιλέγοντας να εισάγετε τον τύπο (επιλογή «Import 'OnClickListener' (android.view.View)»). Αυτό όμως δεν αρκεί. Το Eclipse περιμένει να υλοποιήσουμε μια μέθοδο και αυτό θα κάνουμε.

Επιστρέφοντας στη μέθοδο onCreate εντοπίστε τη γραμμή όπου κατά την προηγούμενη δραστηριότητα αναθέσατε την αναφορά στην όψη Button στη μεταβλητή που είχατε ορίσει:

```
theButton = (Button)findViewById(R.id.myButton);
```

Αμέσως μετά προσθέστε μια γραμμή αναμονής για γεγονότα ως εξής:

```
theButton.setOnClickListener(this);
```

Η γραμμή αυτή απαιτεί από την εφαρμογή να παρακολουθεί για κλικ πάνω στο Button. Εντός των παρενθέσεων προσδιορίζεται το αντικείμενο χειριστής των κλικ. Στην περίπτωσή μας πρόκειται για την ενεργό υλοποίηση της ίδιας της κλάσης Activity («this»). Η θέση στην οποία υλοποιούνται η αναμονή και η ανταπόκριση στα γεγονότα, η μέθοδος στην οποία εντάσσονται και το αντικείμενο χειριστής είναι κρίσιμα για τη λειτουργία μιας Android εφαρμογής, περισσότερα όμως για αυτό αργότερα.

Απομένει η διαχείριση των συμβάντων. Για το σκοπό αυτό θα δημιουργήσουμε τη μέθοδο onClick αμέσως μετά από το κλείσιμο της μεθόδου onCreate:

```
public void onClick(View v){
        //εδώ πηγαίνει ο κώδικας που χειρίζεται τα κλικ
}
```

Και πάλι θα χρειαστεί να εισάγουμε τον τύπο «View» κάνοντας την επιλογή «Import 'View' (android.view)» . Όταν ένα γεγονός τύπου κλικ εμφανιστεί το περιεχόμενο της μεθόδου θα εκτελεστεί. Το v (που είναι τύπου View) είναι παράμετρος της μεθόδου που χρησιμοποιείται για να ληφθεί μια αναφορά στην όψη (View) όπου έγινε το κλικ ώστε να γίνει η ταυτοποίηση.

Πρέπει τώρα να ταυτοποιηθεί η όψη που έχει επιλεγεί με το κλικ. Η παρακολούθηση των κλικ γίνεται από μία μόνο ρύθμιση. Όμως μέσα στον κώδικα διαχείρισης η εφαρμογή μπορεί τώρα να διαχωρίσει μεταξύ πολλαπλών όψεων και άρα να διαχειριστεί κλικ σε αυτές, ελέγχοντας αν η αναφορά στη μεταβλητή που μόλις φτιάξαμε αφορά στην συγκεκριμένη όψη. Για παράδειγμα:

```
if(v.getId()==theButton.getId()){
        //εδώ προγραμματίζουμε την ανταπόκριση σε κλικ στο κουμπί
}
```

Το συγκεκριμένο μπλόκ θα εκτελεστεί μόνο εφόσον το κουμπί «πατηθεί» (κλικ). Η δυνατότητα αυτή γίνεται κρίσιμη όταν έχουμε πολλαπλά διαδραστικά στοιχεία που μπορούν να «πατηθούν».

Προγραμματίστε την ανταπόκριση στο «πάτημα» του κουμπιού ώστε όταν έχει πατηθεί να γράφει επάνω του Pressed δηλαδή «πατημένο». Ο κώδικάς σας θα πρέπει να μοιάζει κάπως έτσι:

```
public void onClick(View v){
```

```
        if(v.getId()==theButton.getId()){
                theButton.setText("Pressed");
        }
}
```

ΚΕΦΑΛΑΙΟ 5

Δραστηριότητα 5.2

Μεταφερθείτε στον κατάλογο drawables ή αν δεν υπάρχει, δημιουργείστε τον. Ότι δημιουργήσουμε μέσα σε αυτόν θα φαίνεται σε όλες τις συσκευές που εξυπηρετεί η εφαρμογή μας, ασχέτως API και διαστάσεων οθόνης. Αυτό δεν είναι γενικά καλή πρακτική.

Με δεξί κλικ μέσα στον κατάλογο επιλέξτε να δημιουργήσετε νέο αρχείο και προσδιορείστε το ως «Android XML File». Στον βοηθό (wizard) που θα εμφανιστεί επιλέξτε από τους τύπους να κατασκευάσετε ένα αρχείο σχήματος (shape). Σε τέτοια αρχεία ορίζουμε μια εικόνα προσδιορίζοντας με σήμανση χαρακτηριστικά του σχήματος και της εμφάνισής τους. Ονομάστε το αρχείο «my_shape.xml».

Μπορείτε τώρα να επιλέξετε από το φάσμα διαθέσιμων γενικών τύπων σχημάτων που περιλαμβάνουν γραμμές, παραλληλόγραμμα, ελλείψεις και δακτύλιους. Επιλέξτε ένα παραλληλόγραμμο (rectangle) επεξεργαζόμενοι το ριζικό (root) στοιχείο σχήμα (shape). Αυτό θα δώσει τον ακόλουθο κώδικα:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

</shape>
```

Το παραλληλόγραμμο αυτό δεν έχει ακόμα καθορισμένες ιδιότητες. Προσθέστε ιδιότητες ορίζοντας μια διαβάθμιση, τύπο γωνιών κλπ. Είναι πιο εύκολο από ότι φαίνεται να προσθέσετε μερικές καθώς το Eclipse, ενώ πληκτρολογείτε θα σας δίνει προτάσεις για διαθέσιμα στοιχεία και χαρακτηριστικά οπότε με κάποιες δοκιμές μπορείτε να φτιάξετε το δικό σας παραλληλόγραμμο. Αποθηκεύστε το τελικό αποτέλεσμα. Το αποτέλεσμα αυτό θα πρέπει να δείχνει κάπως έτσι:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <gradient
        android:angle="270"
        android:centerColor="#FFFFFF00"
        android:endColor="#FF0000FF"
        android:startColor="#FFFF0000"
    android:type="linear" />

    <corners android:radius="10dp" />

    <stroke
        android:width="2dp"
        android:color="#FF339966" />
</shape>
```

ΚΕΦΑΛΑΙΟ 7

Δραστηριότητα 7.1.1

Στην αρχή της κλάσης κύριας δραστηριότητας (main Activity) και πριν τη μέθοδο onCreate επιλέξτε ένα όνομα για τις προτιμήσεις σας π.χ.:

public static final String MY_APP_PREFS = “MyAppPrefs”;

Λόγω public static η μεταβλητή αυτή θα είναι διαθέσιμη σε όλες τις κλάσεις μέσα στην εφαρμογή.

Στην onClick μετά το σημείο που ορίζετε το κείμενο πάνω στο κουμπί να είναι “Pressed” χρησιμοποιείστε το όνομα της μεταβλητής για να ανακτήσετε τις προτιμήσεις. Την ώρα που θα γράφετε την επόμενη γραμμή θα χρειαστεί να εισάγετε και την κλάση android.content. Shared Preferences με το γνωστό τρόπο:

SharedPreferences thePrefs = getSharedPreferences(MY_APP_PREFS, 0);

Η πρώτη παράμετρος είναι το όνομα που ορίσατε και η δεύτερη ο τρόπος (mode) που θα χρησιμοποιούμε ως προεπιλογή. Για να επεξεργαστείτε τις προτιμήσεις σας και να ορίσετε τιμές χρειαζόμαστε έναν editor:

SharedPreferences.Editor prefsEd = thePrefs.edit();

Γράψτε μια τιμή στις προτιμήσεις σας που να έχει σχέση με το αν πατήθηκε το κουμπί. Επειδή αυτό είτε έχει γίνει είτε όχι η τιμή θα είναι boolean. Η εντολή εγγραφής θα δείχνει κάπως έτσι:

prefsEd.putBoolean(“btnPressed”, true);

Και για να αποθηκευτεί η επεξεργασία σας προσθέστε:

prefsEd.commit();

Τώρα μπορείτε να ρυθμίσετε την εφαρμογή σας ώστε αν κάποτε πατήθηκε το κουμπί να φαίνεται «Pressed» επάνω του και κάθε επόμενη φορά που ανοίγει η εφαρμογή. Την onCreate μετά τον υπάρχοντα κώδικα ανακτήστε τις SharedPreferences. Αν προσθέσετε τον παρακάτω κώδικα και «τρέξετε» την εφαρμογή, άπαξ και πατηθεί μια φορά το κουμπί θα φαίνεται πάντα πατημένο:

```
SharedPreferences thePrefs = getSharedPreferences(MY_APP_PREFS, 0);
boolean pressed = thePrefs.getBoolean(“btnPressed”, false);
if (pressed) theButton.setText(“Pressed”);
```

Επιβεβαιώστε το αποτέλεσμα.

Δραστηριότητα 7.1.2

Προσθέστε μετά τη δημιουργία κώδικα για γράψιμο στο φάκελο (πάντα μέσα στο try μπλόκ). Ο κώδικάς σας θα δείχνει κάπως έτσι:

```
String fileContent = “my data file content”;
fileOut.write(fileContent.getBytes());
```

Κλείστε το φάκελο:

```
fileOut.close();
```

Τέλος ανοίξτε το αρχείο και ανακτήστε τα δεδομένα σας χρησιμοποιώντας έναν BufferedReader. Το αποτέλεσμα θα πρέπει να δείχνει κάπως έτσι:

```
try{
    FileInputStream fileIn = openFileInput(“my_file”);
     InputStreamReader streamIn = new InputStreamReader(fileIn);
    BufferedReader fileRead = new BufferedReader(streamIn);
    StringBuilder fileBuild = new StringBuilder(&amp;quot;&amp;quot;);
```

```
            String fileLine=fileRead.readLine();
            while(fileLine!=null){
                  fileBuild.append(fileLine+&amp;quot;\n&amp;quot;);
                   fileLine=fileRead.readLine();
            }
            String fileText = fileBuild.toString();
            streamIn.close();
      }
      catch(IOException ioe){
             Log.e("APP_TAG", "IO Exception", ioe);
      }
```

ΚΕΦΑΛΑΙΟ 8

Δραστηριότητα 8.2.1

Στην κλάση της κύριας δραστηριότητας προσθέστε μια μεταβλητή στην αρχή της κλάσης η οποία θα χρησιμοποιείται για τη μετάδοση του μηνύματος καταγραφής. Ονομάστε τη LOG_TAG. Ο κώδικάς σας θα πρέπει να μοιάζει κάπως έτσι:

```
private final String LOG_TAG = "MainActivity";
```

Στη μέθοδο onClick, πριν τον ορισμό του κειμένου πάνω στο κουμπί προσθέστε τον κώδικα:

```
Log.v(LOG_TAG, "button clicked");
```

Θα χρειαστεί, όπως έχουμε κάνει και αλλού, να εισάγετε το «android.util.Log» στην κλάση. Κατά την εγγραφή στο Log είναι διαθέσιμες αρκετές μέθοδοι που αντανακλούν το σκοπό του μηνύματος. Εδώ χρησιμοποιήσαμε το v (από το verbose). Διατίθενται ακόμα τα d – από το debug, το i από το information, το w από το warning και το e από το error.

Τρέξτε την εφαρμογή χρησιμοποιώντας το κουμπί Run. Κρατείστε ανοιχτή την όψη LogCat στο Eclipse και πατήστε το κουμπί στη διεπαφή χρήστη της συσκευής σας. Ανάμεσα στα άλλα μηνύματα στο LogCat θα πρέπει να φανεί ένα μήνυμα που στο Application θα δείχνει com.example.myfirstapp, στο Tag θα δείχνει MainActivity, και στο Text θα δείχνει button clicked.

Προχωρήστε στην ανάπτυξη αντίστοιχων μηνυμάτων σε σημαντικά σημεία στην εφαρμογή σας ώστε να μπορείτε να παρακολουθήσετε την εξέλιξή της όσο τρέχει, από τις σχετικές εγγραφές.

ΚΕΦΑΛΑΙΟ 12

Δραστηριότητα 12.3.2

Στο Eclipse, επιλέγουμε το έργο της εφαρμογής στον Package Explorer, επιλέγουμε File και μετά Export. Αναπτύξτε το φάκελο Android, επιλέξτε Export Android Application και πατήστε Next.

Σε αυτό το σημείο το Eclipse θα υπογραμμίσει όσα λάθη συνάντησε κατά τη διαδικασία της οικοδόμησης (build) και τα οποία πρέπει να αντιμετωπιστούν πριν συνεχίσετε. Αν δεν υπάρχουν άλλα λάθη επιλέξτε Next για να συνεχίσετε.

Στο παράθυρο Keystore Selection, αναζητείστε τη θέση του αρχείου keystore που φτιάξατε στην προηγούμενη δραστηριότητα και εισάγετε το password. Επιλέξτε Next για να συνεχίσετε.

Επιλέξτε μια θέση και ένα όνομα για τον φάκελο APK της εφαρμογής. Αυτό το αρχείο θα αναρτήσετε στο Google Play και οι χρήστες θα κατεβάσουν στις συσκευές τους. Το Eclipse θα

φροντίσει πλέον αυτόματα την υπογραφή. Μετά το Finish το αρχείο APK βρίσκετε στο σημείο που υποδείξατε.

Το αρχείο αυτό μπορείτε πλέον να το αντιγράψετε σε μια συσκευή Android. Μόλις το έχετε αντιγράψει στη συσκευή, βρείτε και επιλέξτε το αρχείο APK , και ακολουθήστε τις οδηγίες για να το εγκαταστήσετε.

Εάν έχετε υπογράψει την αίτηση σωστά , το σύστημα θα πρέπει να είναι σε θέση να εγκαταστήσει την εφαρμογή και επομένως θα μπορέσετε να εκτελέσετε την έκδοση της εφαρμογής σας στη συσκευή.

Στο εξής, για κάθε νέα έκδοση του πακέτου θα χρειάζεστε το ίδιο κλειδί δημοσίευσης οπότε μην το χάσετε.