ΕΠΙΜΟΡΦΩΣΗ ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
το έργο συγχρηματοδοτείται 80% από το ΕΚΤ και 20% από Εθνικούς Πόρους
Ε.Π. "Κοινωνία της Πληροφορίας"
Πράξη «Δράσεις Επιμόρφωσης Εκπαιδευτικών Πληροφορικής»
Υποέργο 1 «Επιμόρφωση Εκπαιδευτικών Πληροφορικής»

Έργο	**<u>"Υλοποίηση Επιμόρφωσης Εκπαιδευτικών Πληροφορικής"</u>** στο πλαίσιο του Υποέργου 1: Επιμόρφωση Εκπαιδευτικών Πληροφορικής της πράξης: Δράσεις Επιμόρφωσης Εκπαιδευτικών Πληροφορικής **Του ΥΠΕΠΘ**
Μέρος	**Μέρος Α** Τεχνολογίες αιχμής, επικαιροποίηση και ανανέωση γνώσεων πληροφορικής
Ενότητα	**[Α6] Προηγμένο Περιβάλλον Αντικειμενοστρεφούς Προγραμματισμού Java**
Σύντομη Περιγραφή	Σκοπός αυτής της ενότητας είναι η εισαγωγή και η κατανόηση των βασικών αρχών του αντικειμενοστρεφούς προγραμματισμού χρησιμοποιώντας τη γλώσσα Java. Οι επιμορφούμενοι θα εξοικειωθούν με τη χρήση της πλατφόρμας εργασίας Eclipse και με τον τρόπο δημιουργίας γραφικών διεπαφών χρήστη.
Συγγραφική Ομάδα	Επιστημονικός Υπεύθυνος: **Γιάννης Τζίτζικας** (tzitzik@csd.uoc.gr) Επίκουρος Καθηγητής, Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης Συγγραφική Ομάδα **Παναγιώτης Παπαδάκος** (papadako@csd.uoc.gr) **Νικόλαος Αρμεναντζόγλου** (armenan@csd.uoc.gr) **Γιάννης Μαρκετάκης** (marketak@csd.uoc.gr) Μεταπτυχιακοί Φοιτητές, Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περιεχόμενα

1. Εισαγωγή

Η ανάγκη για δημιουργία όλο και πιο σύνθετου λογισμικού και η επιθυμία μας για συγγραφή κώδικα που να μπορεί να επαναχρησιμοποιηθεί και να κατανοηθεί εύκολα μας οδήγησε στο αντικειμενοστρεφές (object-oriented) υπόδειγμα προγραμματισμού. Το αντικειμενοστρεφές υπόδειγμα προγραμματισμού μας προσφέρει τους απαραίτητους αφαιρετικούς μηχανισμούς για να επιτύχουμε κάτι τέτοιο. Επιγραμματικά, μας προσφέρει την έννοια της κλάσης (η οποία ενθυλακώνει δεδομένα και τις συναφείς με αυτά λειτουργίες) και μηχανισμούς που επιτρέπουν την εύκολη επαναχρησιμοποίηση και προσαρμογή κλάσεων.

Σύμφωνα με το αντικειμενοστρεφές υπόδειγμα, η επιδιωκόμενη λειτουργικότητα του λογισμικού επιτυγχάνεται μέσω της δημιουργίας αντικειμένων τα οποία συνεργάζονται μεταξύ τους. Τα αντικείμενα είναι στιγμιότυπα των κλάσεων που έχουμε ορίσει εμείς ή είναι ήδη ορισμένες στις βιβλιοθήκες της γλώσσας. Μια κλάση είναι στην ουσία ένας σύνθετος τύπος δεδομένων ο οποίος εκτός από δεδομένα έχει και λειτουργίες διαχείρισης των δεδομένων του. Μια κλάση μπορεί να δηλωθεί ως επέκταση (ή αλλιώς εξειδίκευση) μιας άλλης και αυτό συνεπάγεται κληρονομικότητα των αντίστοιχων ιδιοτήτων (δεδομένων και λειτουργιών) της κλάσης. Συνάμα μας δίνεται η δυνατότητα προσαρμογής των κληρονομημένων ιδιοτήτων (π.χ. μέσω της δυνατότητας επικάλυψης/υποσκελισμού των κληρονομημένων μεθόδων).

Η Java είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού (η οποία δημιουργήθηκε στις αρχές του 1991 από την εταιρία Sun). Ιδιαίτερο χαρακτηριστικό της γλώσσας είναι ότι βασίζεται στην αρχιτεκτονική εικονικής μηχανής (Java Virtual Machine) η οποία επιτρέπει στον κώδικα μας να είναι ανεξάρτητος από τα ιδιαίτερα χαρακτηριστικά του εκάστοτε λειτουργικού συστήματος. Αυτό σημαίνει ότι δεν χρειάζεται να συντηρούμε διαφορετικές εκδόσεις του κώδικα ανάλογα με το λειτουργικό για το οποίο προορίζεται η εφαρμογή μας, άρα επιτυγχάνουμε μεταφερσιμότητα (portability).

Τέλος αξίζει να σημειωθεί ότι η Java συνοδεύεται από μια πολύ πλούσια βιβλιοθήκη έτοιμων κλάσεων **API** (Application Programming Interface) και λειτουργιών, την οποία μπορούμε να εκμεταλλευτούμε ώστε να επιταχύνουμε τη διαδικασία συγγραφής λογισμικού.

Παρακάτω επιχειρούμε μια πολύ σύντομη εισαγωγή στις βασικές έννοιες της γλώσσας Java. Λόγω της περιορισμένης έκτασης κάποια θέματα δεν σχολιάζονται καθόλου ενώ άλλα παρουσιάζονται σε απλουστευμένη μορφή. Για περισσότερα ο αναγνώστης πρέπει να καταφύγει στις πηγές που αναφέρονται στο τέλος του κειμένου.

2. Συγγραφή – Μεταγλώττιση και Εκτέλεση Προγραμμάτων

Εν συντομία, η διαδικασία συγγραφής λογισμικού έχει ως εξής. Γράφουμε τα προγράμματα μας σε αρχεία κειμένου με επέκταση "**.java**", (π.χ. myFirstProgram.java). Κατόπιν, τα μεταφράζουμε σε μια ενδιάμεση γλώσσα (**bytecode**), χρησιμοποιώντας το μεταγλωττιστή (**javac**). Ο μεταφραστής javac είναι διαθέσιμος στο **JDK** (Java Development Kit), το οποίο παρέχεται δωρεάν από την εταιρία Sun. Αποτέλεσμα της μετάφρασης είναι αρχεία με επέκταση "**.class**" (εδώ myFirstProgram.class), τα οποία μπορούν να εκτελεστούν με τη χρήση της εντολής "**java"**, (εδώ '*java myFirstProgram*') σε οποιοδήποτε λειτουργικό σύστημα έχει εγκατεστημένη την Εικονική Μηχανή της Java . Τα παραπάνω βήματα μπορούν να γίνουν είτε από κονσόλα είτε με τη βοήθεια ενός **IDE** (Integrated Development Enviroment) το οποίο μας προσφέρει και πολλές άλλες ευκολίες, π.χ. γρήγορη αποσφαλμάτωση (debugging) και εύκολη οργάνωση σε πακέτα ("**.jars"**) της εφαρμογής μας. Για το λόγο αυτό στην ενότητα 8 γίνεται μια σύντομη εισαγωγή σε ένα πολύ διαδεδομένο IDE, το **Eclipse**.

Ακολουθεί η κλασσική εφαρμογή «Hello World» γραμμένη σε Java. Πληκτρολογείστε σε ένα αρχείο κειμένου με όνομα *HelloWorld.java* το ακόλουθο κείμενο:

```
1 class HelloWorld {
2   public static void main (String args[]) {
3    System.out.println("Hello World!");
4   }
5 }
```

Για να μεταγλωττίσετε το πρόγραμμα, σιγουρευτείτε ότι είστε στον ίδιο φάκελο με το HelloWorld.java και πληκτρολογείστε '*javac HelloWorld.java*'. Όταν το πρόγραμμα μεταφραστεί επιτυχώς, ο μεταφραστής τοποθετεί την εκτελούμενη έξοδο σε ένα αρχείο που ονομάζεται HelloWorld.class στον ίδιο φάκελο. Μπορείτε να τρέξετε το πρόγραμμα σας πληκτρολογώντας '*java HelloWorld*'. Συγχαρητήρια! Μόλις γράψατε το πρώτο σας πρόγραμμα σε Java.

3. Βασικά στοιχεία της Java

Σε αυτό το κεφάλαιο θα μιλήσουμε για τα σημαντικά στοιχεία της γλώσσας προγραμματισμού Java.

3.1. Σχόλια

Τα σχόλια αποτελούν αναπόσπαστο κομμάτι κάθε προγράμματος (απαραίτητο για την κατανόηση και επέκταση του κώδικα από άλλους προγραμματιστές). Σχόλια μπορούν να γραφτούν σε οποιοδήποτε σημείο του αρχείου και αγνοούνται από το μεταγλωττιστή της Java. Υπάρχουν δύο τρόποι για να γράψουμε σχόλια. Σύμφωνα με τον πρώτο γράφουμε το σχόλιο μεταξύ των «/*» και «*/», ενώ με βάση το δεύτερο το γράφουμε μετά από «//». Στη δεύτερη περίπτωση το σχόλιο δεν μπορεί να υπερβαίνει τη μία γραμμή. Μία παραλλαγή του πρώτου τρόπου είναι και τα Javadoc, τα οποία επιτρέπουν την εμφάνιση των σχολίων μέσω HTML σελίδων. Οι HTML σελίδες παράγονται μέσω του εργαλείου **javadoc** (εδώ 'javadoc HelloWorld.java'). Στην περίπτωση αυτή τα σχόλια πρέπει να περικλείονται από «/**» και «*/» και μπορούμε να χρησιμοποιήσουμε και άλλες συγκεκριμένες λέξεις (τις οποίες δεν θα αναλύσουμε περεταίρω). Παρακάτω βλέπουμε τα τρία είδη σχολίων:

```
1 /* Σχόλιο πολλών
2   Γραμμών */
3 //Σχόλιο μονής γραμμής
4 /** Αυτό είναι ένα σχόλιο τύπου Javadoc */
```

3.2. Βασικοί τύποι δεδομένων

Η Java χρησιμοποιεί 8 βασικούς τύπους δεδομένων, οι οποίοι είναι οι εξής:

- **byte:** καταλαμβάνει 8 bits και χρησιμοποιείται για την αναπαράσταση ακεραίων του διαστήματος [-128, 127].
- **short:** καταλαμβάνει 16 bits και χρησιμοποιείται για την αναπαράσταση ακεραίων του διαστήματος [-32.768, 32.767].
- **int:** καταλαμβάνει 32 bits και χρησιμοποιείται για την αναπαράσταση ακεραίων του διαστήματος [-2.147.483.648, 2.147.483.647]. Αποτελεί έναν από τους πιο συχνά χρησιμοποιούμενους τύπους δεδομένων.
- **long:** καταλαμβάνει 64 bit και χρησιμοποιείται για την αναπαράσταση ακεραίων του διαστήματος [-9.223.372.036.854.775.808, 9.223.372.036.854.775.807]. Χρησιμοποιείται όταν θέλουμε πολύ μεγάλο εύρος ακεραίων.
- **float:** καταλαμβάνει 32 bit και υλοποιεί το standard IEEE 754 για αριθμούς κινητής υποδιαστολής μονής ακρίβειας. Χρησιμοποιείται για την παράσταση πραγματικών αριθμών.
- **double:** καταλαμβάνει 64 bit και υλοποιεί το standard IEEE 754 για αριθμούς κινητής υποδιαστολής διπλής ακρίβειας. Το χρησιμοποιούμε όταν θέλουμε να παραστήσουμε πραγματικούς αριθμούς μεγαλύτερης ακρίβειας.
- **boolean:** μπορεί να πάρει μόνο δύο τιμές, *true* ή *false*. Χρησιμοποιείται σε λογικές εκφράσεις και ελέγχους.
- **char:** χρησιμοποιείται για την παράσταση 16 bit Unicode χαρακτήρων, και χρησιμοποιείται για την παράσταση και μη λατινικών χαρακτήρων (π.χ. ελληνικών).

Εκτός από τα παραπάνω, η Java προσφέρει ειδική υποστήριξη για συμβολοσειρές (δηλαδή strings) στο API της (java.lang.String), για το οποίο θα μιλήσουμε παρακάτω.

3.3. Δήλωση μεταβλητών

Για να χρησιμοποιήσουμε μια μεταβλητή, πρέπει να καθορίσουμε το όνομα και τον τύπο της. Ο τύπος μιας μεταβλητής καθορίζει τον τύπο των τιμών που μπορεί να κρατήσει. Παραδείγματα δήλωσης (και αρχικοποίησης) μεταβλητών ακολουθούν.

```
1 int c;        // Δήλωση ακέραιας μεταβλητής
2 float f = 0.0; /* Δήλωση μεταβλητής κινητής υποδιαστολής και ανάθεση
3                   της τιμής 0.0  */
4 char c= 'λ'; // Δήλωση μεταβλητής χαρακτήρα και ανάθεση χαρακτήρα 'λ
5 String s = "Java"; // Δήλωση συμβολοσειράς χαρακτήρων
```

Παρατήρηση: Στο τέλος κάθε δήλωσης (όπως και εντολής) γράφουμε το χαρακτήρα'**;**'.

Γενικά θα μπορούσαμε να διακρίνουμε τις μεταβλητές στις εξής κατηγορίες
(α) **τοπικές μεταβλητές**, δηλώνονται στα σώματα των μεθόδων (οι μέθοδοι αποτελούν τις λειτουργίες μιας κλάσης, όπως ορίσαμε στην εισαγωγή)
(β) **παράμετροι μεθόδων**, δηλώνονται στην επικεφαλίδα των μεθόδων
(γ) **μεταβλητές στιγμιότυπων (instance variables)**, δηλώνονται στις κλάσεις και κάθε αντικείμενο (στιγμιότυπο της κλάσης) μπορεί να έχει διαφορετική τιμή σε αυτήν τη μεταβλητή
(δ) **στατικές μεταβλητές κλάσης (static variables)**, δηλώνονται στις κλάσεις αλλά δεν αφορούν τα στιγμιότυπα της, αλλά την ίδια την κλάση.

3.4. Δήλωση σταθερών

Σταθερές μπορούμε να ορίσουμε χρησιμοποιώντας τη λέξη κλειδί **final**. Στις σταθερές μπορούμε να αναθέσουμε τιμή μόνο μία φορά. Αν προσπαθήσουμε να αλλάξουμε την τιμή μιας σταθεράς, θα διαμαρτυρηθεί ο μεταγλωττιστής της Java. Παράδειγμα δήλωσης σταθεράς δίδεται παρακάτω:

```
1 class HelloWorld {
2  public static void main (String args[]) {
3       final String breakTheRules = "Hello from Java!"; // Η σταθερά
4       /*breakTheRules = "Hello World!";*/ // Λάθος!
5       System.out.println(breakTheRules); // Εκτύπωση της σταθεράς
6  }
7 }
```

3.5. Πίνακες

Παραδείγματα δήλωσης, αρχικοποίησης και χρήσης πινάκων:

```
1 int array[]; // Δήλωση πίνακα ακεραίων
2 array = new int[12]; // Ορισμός πίνακα ακεραίων 12 στοιχείων
3 array[0] = 1; // Βάζουμε σαν πρώτο στοιχείο του πίνακα τον αριθμό 1
4 int arrayNew[] = {1, 2, 3};  // Δήλωση και αρχικοποίηση πίνακα ακεραίων
5 int ar2D[][] = new int[4][5]; //Δήλωση και ορισμός δισδιάστατου πίνακα
```

Το **new** είναι μια δεσμευμένη λέξη στη Java και χρησιμοποιείται όχι μόνο για τη δημιουργία πινάκων (για τη δέσμευση του χώρου μνήμης που απαιτούν), αλλά και για την δημιουργία στιγμιότυπων των κλάσεων όπως θα δούμε παρακάτω.

3.6. Τελεστές

Η Java παρέχει ένα πολύ πλούσιο σύνολο τελεστών, οι οποίοι μπορούν να χωριστούν στις παρακάτω κατηγορίες:

- Αριθμητικοί
- Σύγκρισης
- Λογικοί
- Bitwise

Από αυτούς οι πιο συχνά χρησιμοποιούμενοι είναι οι τρεις πρώτοι. Παρακάτω θα τους δούμε αναλυτικά.

Αριθμητικοί (Πράξεις ανάμεσα σε αριθμούς)

- + Πρόσθεση
- - Αφαίρεση
- * Πολλαπλασιασμός
- / Διαίρεση
- % Υπόλοιπο διαίρεσης
- ++ Αύξηση κατά ένα
- -- Μείωση κατά ένα

Τους αριθμητικούς τελεστές μπορούμε να τους χρησιμοποιήσουμε και σε συνδυασμό με τον τελεστή ανάθεσης (=). Συγκεκριμένα

- += Ανάθεση με αύξηση
- -= Ανάθεση με μείωση
- *= Ανάθεση με πολλαπλασιασμό
- /= Ανάθεση με διαίρεση
- %= Ανάθεση με υπόλοιπο

Σύγκρισης (Σχέση ανάμεσα σε δύο τελεστέους)

- == Ισότητα
- != Ανισότητα
- > Μεγαλύτερο από
- < Μικρότερο από
- >= Μεγαλύτερο από ή ίσο
- <= Μικρότερο από ή ίσο

Λογικοί (Εφαρμόζονται σε Boolean εκφράσεις, παρουσιάζονται οι πιο σημαντικοί)

- == Ισότητα
- != Ανισότητα
- && Σύζευξη (λογικό ΚΑΙ)
- || Διάζευξη (λογικό Ή)

Προσοχή: Το σύμβολο = δεν συμβολίζει έλεγχο ισότητας αλλά εκχώρηση (ο έλεγχος ισότητας γίνεται με το ==).

Προσοχή: Στους τελεστές υπάρχει σειρά προτεραιότητας. Με την χρήση παρενθέσεων μπορούμε να καθορίσουμε την επιδιωκόμενη σειρά εκτέλεσης των πράξεων.

4. Ροή προγράμματος

Σε κάθε πρόγραμμα είναι απαραίτητο να ελέγχουμε τη ροή του ανάλογα με το αν ισχύουν κάποιες συνθήκες και να την ανακατευθύνουμε κατάλληλα. Υπάρχουν πολλοί τρόποι να κατευθύνουμε τη ροή ενός προγράμματος, οι οποίοι μπορούν να χωριστούν σε δύο μεγάλες κατηγορίες, τις δομές ελέγχου και τις δομές επανάληψης.

4.1. Δομές ελέγχου

Οι δομές ελέγχου μας επιτρέπουν τον καθορισμό της ροής εκτέλεσης των εντολών του προγράμματός μας.

4.1.1. If else

Η **if** είναι η εντολή διακλάδωσης της Java. Εάν είναι αληθής η συνθήκη ελέγχου (αυτή που αναγράφεται μετά το if), τότε, εκτελείται η επόμενη εντολή ή block εντολών (ένα block εντολών περικλείεται από άγκιστρα «{» «}») . Το **else** είναι προαιρετικό και καθορίζει την εντολή (ή block εντολών) που πρέπει να εκτελεστούν εάν η συνθήκη ελέγχου δεν είναι αληθής. Ως συνθήκη ελέγχου μπορεί να είναι οποιαδήποτε λογική έκφραση.

Προσοχή: Μόνο μία εντολή μπορεί να υπάρχει αμέσως μετά από μία if ή else. Αν θέλουμε να συμπεριλάβουμε παραπάνω εντολές, θα πρέπει να δημιουργήσουμε ένα block από εντολές.

Συνάμα μπορούμε να συνδυάσουμε ένα else και ένα if δημιουργώντας ένα else if και να το χρησιμοποιήσουμε για να εξετάσουμε μια πληθώρα από περιπτώσεις. Επιπλέον μπορούμε να έχουμε φωλιασμένες if και else μέσα σε block κάποιας άλλης if και else.

Ένα παράδειγμα χρήσης των if και else είναι το παρακάτω.

```
1 int a, b, c, d, i, j, k;
2 // .. Άλλες δηλώσεις
3 if(i == 10)          // Αν το i είναι ίσο με 10
4 {                          // Αρχή block
5       if(j < 20) a = b;    // Αν το j είναι μικρότερο από το 20, a = b
6       else if(k > 100) c = d;/* Αλλιώς αν το k είναι μεγαλύτερο του
7                            100, c = d */
8       else a = c;          // Αλλιώς a = c
9 }                          // Τέλος block
10 else a = d;         // Αν η πρώτη if δεν ισχύει, a = d
```

4.1.2. Switch

Η λέξη κλειδί **switch** μας επιτρέπει να δηλώσουμε πολλαπλές διακλαδώσεις. Παρέχει ένα γρήγορο τρόπο για να στρέψουμε την εκτέλεση του προγράμματος μας σε διαφορετικά τμήματα του κώδικά μας, με βάση την τιμή μιας έκφρασης (εκ τούτου αποτελεί καλύτερη εναλλακτική από τη χρήση μιας πολύ μεγάλης σειράς if-else-if…). Η συνθήκη ελέγχου μιας switch μπορεί να είναι τύπου byte, short, int ή char. Για να δηλώσουμε περιπτώσεις χρησιμοποιούμε την δήλωση **case.** Η δήλωση **default,** μπορεί να χρησιμοποιηθεί για την περίπτωση που δεν ισχύει κανένα από τα δηλωμένα case και είναι προαιρετική. Επίσης η δήλωση **break** μπορεί να χρησιμοποιηθεί για να βγούμε από μία switch ή από ένα βρόγχο όπως θα δούμε παρακάτω, και είναι και αυτή προαιρετική. Παραλείποντας το break σε μια case η

εκτέλεση του προγράμματος θα συνεχίσει με την επόμενη case. Ένα παράδειγμα χρήσης της switch είναι το παρακάτω:

```
1 int i;
2 switch(i) {  // Αρχή block switch
3      case 0: // Σε περίπτωση που το i είναι 0
4              i++;    // Αύξησέ το κατά 1
5              break;  // Βγες από την switch
6      case 1: // Σε περίπτωση που το i είναι 1
7      case 2: // Ή σε περίπτωση που το i είναι 2
8              i--;    // Μείωσέ το κατά 1
9              break;  // Βγες από την switch
10     default:        // Σε οποιαδήποτε άλλη περίπτωση
11             break;  // Βγες από την switch
12}                    // Τέλος block switch
```

4.2. Δομές επανάληψης

Οι δομές επανάληψης μας επιτρέπουν να καθορίζουμε βρόγχους, δηλαδή εντολές που θέλουμε να εκτελούνται επαναληπτικά όσο η συνθήκη ελέγχου ικανοποιείται. Η συνθήκη μπορεί να είναι οποιαδήποτε λογική έκφραση, όπως και παραπάνω.

4.2.1. While

Ο βρόγχος **while**, είναι ο πιο βασικός βρόγχος που παρέχει η Java. Επαναλαμβάνει μία εντολή ή ένα block εντολών όσο η συνθήκη επανάληψης είναι αληθής. Όταν η συνθήκη γίνει ψευδής, η επανάληψη τερματίζεται. Ένα παράδειγμα χρήσης της while είναι το παρακάτω, όπου το block της while θα εκτελεστεί 10 φορές.

```
1 int n = 10;
2 while (n > 0) {              // Εφόσον το n είναι μεγαλύτερο του 0
3      n--;                    // Μείωσε το n κατά 1
4 }
```

4.2.2. Do while

Στην περίπτωση της while που είδαμε παραπάνω, αν η συνθήκη αρχικά είναι ψευδής, τότε το σώμα του while δεν θα εκτελεστεί. Μερικές φορές όμως είναι επιθυμητό κάποιος κώδικας να μπορεί να εκτελεστεί ακόμα και αν η συνθήκη μας είναι αρχικά ψευδής. Δηλαδή θέλουμε να ελέγξουμε αν η συνθήκη μας ισχύει στο τέλος του βρόγχου. Για να το πετύχουμε αυτό χρησιμοποιούμε το βρόγχο **do while**. Ένα παράδειγμα χρήσης του, φαίνεται παρακάτω.

```
1 int n = 10;
2 do {                 // Εκτέλεσε
3      n--;            // Μείωσε το n κατά 1
4 } while (n > 0);     // Εφόσον το n είναι μεγαλύτερο του 0
5 // Τελικά το n είναι -1
```

Προσοχή: Μετά τη συνθήκη στο while βάζουμε '**;**'.

4.2.3. For

Ένας βρόγχος **for** έχει την εξής δομή:

for(*Αρχικοποίηση***;** *Συνθήκη ελέγχου***;** *Βήμα επανάληψης***) {**
Σώμα for
}

Αρχικά εκτελείται το τμήμα αρχικοποίησης, το οποίο είναι υπεύθυνο για την ανάθεση τιμών στις μεταβλητές ελέγχου. Στη συνέχεια ελέγχεται η συνθήκη, και αν είναι αληθής ακολουθεί η εκτέλεση του σώματος του βρόγχου, διαφορετικά ο βρόγχος τερματίζει. Μετά την εκτέλεση του σώματος του βρόγχου εκτελείται το βήμα επανάληψης, το οποίο συνήθως αυξάνει ή μειώνει τη μεταβλητή ελέγχου. Στη συνέχεια ελέγχεται πάλι η συνθήκη και εφόσον παραμένει αληθής επαναλαμβάνεται η εκτέλεση του σώματος του βρόγχου. Ένα παράδειγμα του βρόγχου for μπορείτε να δείτε παρακάτω.

Σημείωση: Μερικές φορές είναι χρήσιμο να μπορούμε να επιβάλλουμε μία πρόωρη επανάληψη του βρόγχου όπως θα δούμε στο παράδειγμα. Αυτό μπορεί να γίνει σε όλες τις επαναληπτικές δομές χρησιμοποιώντας τη λέξη κλειδί **continue**. Σε περίπτωση που θέλουμε τη πρόωρη διακοπή του βρόγχου, χρησιμοποιούμε την δήλωση break που γνωρίσαμε νωρίτερα.

```
1 int n = 10;
2 for(int i = 0;       // Δήλωση και αρχικοποίηση μεταβλητής ελέγχου
3         i < 10;      // Συνθήκη ελέγχου
4         i++ ){       // Αύξηση κατά ένα μεταβλητής ελέγχου
5                      // Εφόσον i < 10
6      if(n == 1)      // Αν το n είναι 1
7        continue;     // Συνέχισε στην επόμενη επανάληψη
8      else if(n == 2)// Αν το n είναι 2
9        break;        // Διέκοψε το βρόγχο
10     n--;            // Αλλιώς μείωσε το n κατά 1
11 }
```

5. Κλάσεις – Αντικείμενα

Οι κλάσεις και τα αντικείμενα είναι τα πιο σημαντικά συστατικά μιας αντικειμενοστρεφούς γλώσσας προγραμματισμού. Μια **κλάση** είναι ένας τύπος δεδομένων ο οποίος ορίζεται από τον προγραμματιστή και αποτελείται από:
α) άλλους (απλούς ή σύνθετους) τύπους δεδομένων που συχνά αναφέρονται με το όνομα *γνωρίσματα*, *πεδία* ή *μεταβλητές στιγμιοτύπου* και
β) λειτουργίες που συχνά αναφέρονται με το όνομα *μέθοδοι*.
Τα πεδία και οι μέθοδοι μιας κλάσης αποκαλούνται **μέλη** της. Πρέπει να τονίσουμε, ότι το όνομα μιας κλάσης πρέπει να είναι μοναδικό.

Ένα **αντικείμενο** είναι ένα στιγμιότυπο μιας κλάσης. Αν για παράδειγμα η κλάση μας είναι οι άνθρωποι, το αντικείμενο Μανούσος αποτελεί ένα στιγμιότυπο της.

Μια κλάση στη Java ορίζεται σε ένα αρχείο που (συνήθως) έχει ως όνομα το όνομα της κλάσης και κατάληξη ".java". Παρακάτω δίνεται ένα παράδειγμα μιας κλάσης Vehicle (όχημα) η οποία ορίζεται στο αρχείο "Vehicle.java".

```
1 class Vehicle {      // Ορισμός κλάσης vehicle
2       // Δήλωση πεδίων
3       String manufacturer;
4       String platesNumber;
5 }
```

Μπορούμε να θεωρήσουμε την κλάση Vehicle ως ένα σύνθετο τύπο δεδομένων για τη διαχείριση οχημάτων. Παρατηρούμε ότι περιλαμβάνει τη δήλωση δύο πεδίων (μεταβλητών στιγμιοτύπων). Το πρώτο έχει όνομα manufacturer και είναι τύπου συμβολοσειράς (String) το οποίο μας είναι χρήσιμο για να καταχωρούμε τη μάρκα των συγκεκριμένων οχημάτων που θα δημιουργήσουμε, ενώ το άλλο έχει όνομα platesNumber και θα μας επιτρέψει την καταχώρηση των πινακίδων των οχημάτων. Για τα αντικείμενα που θα δημιουργήσουμε ως στιγμιότυπα αυτής κλάσης Vehicle θα μπορούμε να θέσουμε και να διαβάσουμε τις τιμές αυτών των γνωρισμάτων. Για παράδειγμα:

```
1 Vehicle myCar;
2 myCar = new Vehicle();
3 Vehicle yourCar = new Vehicle();
4 myCar.manufacturer = "BMW";
5 myCar.platesNumber = "HPA0007"
6 System.out.println("Τύπος αυτοκινήτου " + myCar. Manufacturer);
7 System.out.println("Αρ. Πινακίδων  " + myCar. platesNumber);
```

Στη γραμμή 1 δηλώνουμε μια μεταβλητή τύπου Vehicle.
Στη γραμμή 2 δημιουργούμε ένα νέο αντικείμενο τύπου Vehicle και η μεταβλητή myCar κρατά μια αναφορά προς αυτό.
Στη γραμμή 3 δηλώνουμε μια νέα μεταβλητή η οποία δείχνει σε ένα νέο αντικείμενο που δημιουργούμε.
Στις γραμμές 4 και 5 θέτουμε τιμές στα πεδία manufacturer και platesNumber του πρώτου αντικειμένου, ενώ στις γραμμές 6 και 7 εκτυπώνουμε στην κονσόλα τις τιμές αυτών των πεδίων.

Μια κλάση μπορεί να ορίζει και μεθόδους (λειτουργίες) για τη διαχείριση των αντικειμένων της. Ακολουθεί μια πιο πλούσια έκδοση της κλάσης Vehicle.

```
1 class Vehicle {      // Ορισμός κλάσης vehicle
```

```
2        // Δήλωση πεδίων
3       String manufacturer;
4       String platesNumber = "Unknown";      // Αρχική τιμή, άγνωστη πινακίδα
5       // Η προεπιλεγμένη κατασκευάστρια μέθοδος
6       Vehicle() {
7               this("Unknown");// Κλήση 2ης κατασκευάστριας μεθόδου
8       }
9       // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί τον κατασκευαστή
10      Vehicle(String manufacturer) {
11              this.manufacturer = manufacturer;     // Αρχικοποίηση
12      }       // Τέλος κατασκευάστριας μεθόδου
13      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί όλα τα πεδία
14      Vehicle(String manufacturer, String platesNumber) {
15              // Αρχικοποιήσεις
16              this.manufacturer = manufacturer;
17              this.platesNumber = platesNumber;
18      }       // Τέλος κατασκευάστριας μεθόδου
19      // Μέθοδος που επιστρέφει τον κατασκευαστή
20      String getManufacturer(){
21              return manufacturer;
22      }       // Τέλος μεθόδου
23      // Μέθοδος που δίνει τιμή στον κατασκευαστή και δεν επιστρέφει τίποτα
24      void setManufacturer(String manufacturer){
25              this.manufacturer = manufacturer;
26      }       // Τέλος μεθόδου
27      // Μέθοδος που επιστρέφει την πινακίδα
28      String getNumberPlates (){
29              return platesNumber;
30      }       // Τέλος μεθόδου
31      // Μέθοδος που δίνει τιμή στην πινακίδα και δεν επιστρέφει τίποτα
32      void setNumberPlate(String platesNumber){
33              this.platesNumber = platesNumber;
34      }       // Τέλος μεθόδου
35      // Μέθοδος που τυπώνει πληροφορίες για το όχημα
36      String info() {
37              String s = "Vehicle: " + manufacturer + " " + platesNumber;
38              return s;
38      }
39}     // Τέλος κλάσης
```

Η παραπάνω κλάση εκτός των πεδίων της ορίζει και 7 μεθόδους. Τρεις από αυτές είναι **κατασκευάστριες μέθοδοι** (**constructors**). Μια κατασκευάστρια μέθοδος είναι μια μέθοδος η οποία μας βοηθά στη δημιουργία αντικειμένων (π.χ. στη αρχικοποίηση των πεδίων τους). Μια κατασκευάστρια μέθοδος πρέπει να έχει το ίδιο όνομα με την κλάση στην οποία ανήκει. Μπορεί να έχει παραμέτρους οι οποίες δηλώνονται όπως οι μεταβλητές, στην επικεφαλίδα της μεθόδου (ανάμεσα στις παρενθέσεις και χωρίζονται μεταξύ τους με '**,**'). Μία κλάση μπορεί να έχει πολλές κατασκευάστριες μεθόδους, οι οποίες διαφέρουν ως προς το πλήθος και το είδος των παραμέτρων και χρησιμοποιούνται για την διαφορετική αρχικοποίηση των πεδίων της. Η προεπιλεγμένη κατασκευάστρια μέθοδος είναι αυτή που δεν παίρνει καμία παράμετρο. Η δήλωσή της δεν είναι απαραίτητη, αφού δημιουργείται αυτόματα από τον μεταγλωττιστή όταν δεν υπάρχει. Η δυνατότητα να έχουμε πολλές μεθόδους με το ίδιο όνομα αλλά διαφορετικό πλήθος ή/και τύπο παραμέτρων ονομάζεται **υπερφόρτωση** (στα αγγλικά **overloading**).

Στο παράδειγμά μας ορίζουμε τρεις κατασκευάστριες μεθόδους. Η πρώτη που είναι και η προεπιλεγμένη (γραμμή 6), έχει την ιδιότητα ότι καλεί με τη σειρά της την δεύτερη κατασκευάστρια μέθοδο (γραμμή 7), έχοντας σαν παράμετρο το αλφαριθμητικό "Unknown". Αυτό γίνεται χρησιμοποιώντας τη λέξη κλειδί **this**, η οποία αποτελεί έναν τρόπο αναφοράς στο στιγμιότυπο που έχουμε δημιουργήσει (δημιουργείται αυτή τη στιγμή). Η δεύτερη κατασκευάστρια μέθοδος (γραμμή 10) έχει σαν παράμετρο ένα αλφαριθμητικό το οποίο αρχικοποιεί το πεδίο manufacturer της κλάσης. Προσέξτε τη χρήση του this σαν τρόπο αναφοράς στο πεδίο της κλάσης

manufacturer λόγω της σκίασης του από τη παράμετρο του constructor με το ίδιο όνομα. Τέλος στη γραμμή 14, ορίζεται η τρίτη κατασκευάστρια μέθοδος η οποία παίρνει 2 αλφαριθμητικά και αρχικοποιεί τα αντίστοιχα πεδία της κλάσης.

Θυμηθείτε τις εντολές που είχαμε γράψει προηγουμένως:

```
1 Vehicle myCar;
2 myCar = new Vehicle();
3 Vehicle yourCar = new Vehicle();
4 myCar.manufacturer = "BMW";
5 myCar.platesNumber = "HPA0007"
```

Βάσει της εμπλουτισμένης (με κατασκευάστριες μεθόδους) κλάσης μπορούμε να επιτύχουμε ακριβώς το ίδιο με την εντολή:

```
Vehicle myCar = new Vehicle("BMW", "HPA0007");
```

Εκτός των τριών κατασκευαστριών μεθόδων η παραπάνω κλάση έχει και 4 μεθόδους. Οι μέθοδοι μπορεί να έχουν και αυτές παραμέτρους και να επιστρέφουν μια τιμή. Σε περίπτωση που μία μέθοδος δεν χρειάζεται να επιστρέψει κάποια τιμή, χρησιμοποιείται η λέξη κλειδί **void**. Ο γενικός ορισμός μίας μεθόδου είναι ο ακόλουθος:

```
επιστρεφόμενος_τύπος όνομα_μεθόδου(τύπος_ορίσματος όνομα_ορίσματος, ……){
        σώμα μεθόδου
        return (επιστρεφόμενη_τιμή) // εκτός εάν ο επιστρεφόμενος τύπος είναι void
}
```

Για παράδειγμα η μέθοδος info() επιστρέφει μια συμβολοσειρά. Στο σώμα της παρατηρείστε τη χρήση του τελεστή + στη γραμμή 37 ως τρόπο συνένωσης συμβολοσειρών (χαρακτηριστικό της υπερφόρτωσης του τελεστή +). Στη συνέχεια δείχνουμε ένα παράδειγμα χρήσης της κλάσης Vehicle.

```
1 public class MyProgram {
2       //Ένα πρόγραμμα ξεκινά πάντα με την κλήση της main
3       public static void main(String[] args) {
4               Vehicle vehicle1 = new Vehicle(); // Δημιουργία αντικειμένου
5               Vehicle vehicle2 = new Vehicle("Mazerati", "HPA-0000");
6               // Αναφορά σε μεταβλήτή στιγμιοτύπου και ανάθεση νέας τιμής
7               vehicle1.manufacturer = new String("Peugeuot");
8               System.out.println(vehicle1.info()); // Κλήση μεθόδων
9               System.out.println(vehicle2.info()); // Κλήση μεθόδων
10      }       // Τέλος της main
11 }
```

Ορίζουμε μία νέα κλάση MyProgram, η οποία περιλαμβάνει μία static μέθοδο **main()**. Κάθε πρόγραμμα σε Java ξεκινάει την εκτέλεση του με μία κλήση σε αυτήν. Μέσα στο σώμα της main() και συγκεκριμένα στις γραμμές 4 και 5 δημιουργούνται δύο αντικείμενα τύπου Vehicle, χρησιμοποιώντας δύο διαφορετικές κατασκευάστριες μεθόδους. Στη συνέχεια δείχνουμε πως κάνουμε αναφορές σε μεταβλητές στιγμιοτύπου και πώς γίνεται η κλήση μεθόδων. Μπορούμε να αναφερθούμε σε ένα πεδίο ή να καλέσουμε μία μέθοδο ενός αντικειμένου, χρησιμοποιώντας το όνομα του αντικειμένου ακολουθούμενο από μια τελεία και το όνομα της μεταβλητής ή της μεθόδου. Για παράδειγμα στη γραμμή 6 γίνεται ανάθεση νέας τιμής στο πεδίο manufacturer του αντικειμένου vehicle1 και μετέπειτα εκτελούνται κλήσεις στις μεθόδους info και των δύο αντικειμένων, εκτυπώνοντας παράλληλα τις επιστρεφόμενες τιμές τους.

5.1. Οργάνωση κλάσεων

Πριν την κατασκευή ενός προγράμματος είναι σημαντικό να ανακαλύψουμε και να οργανώσουμε τις απαιτούμενες κλάσεις καθώς και τον τρόπο επικοινωνίας τους. Η δομή του προγράμματος αποτελεί μια από τις σημαντικότερες προϋποθέσεις για την σωστή υλοποίησή του. Η Java παρέχει μηχανισμούς που βοηθούν στη σωστή οργάνωση και δόμηση όπως είναι τα **πακέτα**, οι **περιορισμοί προσπέλασης**, οι **διεπαφές** και η **κληρονομικότητα** που θα περιγράψουμε σε αυτό το κεφάλαιο.

5.1.1. Πακέτα

Όπως προαναφέραμε το όνομα μιας κλάσης πρέπει να είναι μοναδικό. Σε μία μεγάλη εφαρμογή, η οποία μπορεί να αποτελείται από πάρα πολλές κλάσεις, είναι πολύ πιθανόν να υπάρξουν κλάσεις με το ίδιο όνομα. Η Java μας δίνει τη δυνατότητα να αποφύγουμε τέτοιου είδους συγκρούσεις μέσω της χρήσης των πακέτων. Χρησιμοποιώντας αυτό το μηχανισμό, μπορούμε να ορίσουμε κλάσεις με το ίδιο όνομα, αρκεί βέβαια αυτές να βρίσκονται σε διαφορετικά πακέτα. Επίσης η χρήση των πακέτων μπορεί να βοηθήσει στην καλύτερη οργάνωση (ομαδοποίηση) των κλάσεων μιας εφαρμογής, αφού κλάσεις που έχουν παρόμοια λειτουργικότητα μπορούν να συμπεριλαμβάνονται στο ίδιο πακέτο.

Για να ορίσουμε το πακέτο μέσα στο οποίο θέλουμε να περιέχεται μια κλάση απλά προσθέτουμε τη λέξη κλειδί **package** και το όνομα του πακέτου, στην πρώτη γραμμή του πηγαίου κώδικα της κλάσης. Αν δεν ορίσουμε κάποιο package τότε η κλάση μας θα αποθηκευτεί στο προεπιλεγμένο πακέτο το οποίο δεν έχει όνομα. Μια τυπική δήλωση ενός πακέτου είναι η εξής:

```
package myPackage;
```

Στην περίπτωση όπου μια κλάση χρειαστεί τη λειτουργικότητα μιας άλλης, η οποία βρίσκεται σε ένα διαφορετικό πακέτο, πρέπει να εισάγει στον κώδικά της το πακέτο αυτό. Αυτό επιτυγχάνετε χρησιμοποιώντας τη λέξη κλειδί **import** και το όνομα του πακέτου. Για παράδειγμα:

```
import otherPackage;
```

5.1.2. Περιορισμοί προσπέλασης

Η Java μας δίνει τη δυνατότητα να ορίσουμε αν τα πεδία και οι μέθοδοι μιας κλάσης θα είναι προσπελάσιμα από άλλες κλάσεις ή όχι, καθορίζοντας τον τρόπο αλληλεπίδρασης μεταξύ τους. Φανταστείτε για παράδειγμα μια κλάση να μπορεί να αλλάζει τα πεδία μιας άλλης χωρίς αυτό να είναι επιθυμητό. Για το λόγο αυτό έχουν οριστεί λέξεις κλειδιά οι οποίες θέτουν περιορισμούς προσπέλασης στα μέλη μιας κλάσης. Παρακάτω βλέπουμε έναν πίνακα με τους περιορισμούς πρόσβασης που παρέχονται.

Περιορισμοί Προσπέλασης	**Τύποι προσπέλασης (λέξεις κλειδιά)**			
	public	**private**	**protected**	**Μη ορισμένος**
Ίδια Κλάση	Ναι	Ναι	Ναι	Ναι
Ίδιο Πακέτο	Ναι	Όχι	Ναι	Ναι

Διαφορετικό Πακέτο	Ναι	Όχι	*Ναι για κλάσεις που την κληρονομούν, αλλιώς όχι(δες 5.1.3. Κληρονομικότητα)	Όχι

Μια κλάση έχει μόνο δυο τύπους προσπέλασης, το **public** και το **προεπιλεγμένο** στο οποίο δεν χρησιμοποιούμε καμία λέξη κλειδί. Όταν μια κλάση είναι public, τότε είναι προσπελάσιμη από οποιοδήποτε κώδικα ενώ αν έχει οριστεί ως default, είναι προσπελάσιμη μόνο από κώδικα ο οποίος ανήκει στο ίδιο πακέτο.

Παρακάτω βλέπουμε την κλάση Vehicle, με τους κατάλληλους περιορισμούς προσπέλασης και ενημερωμένη main. Προσέξτε τη γραμμή 7 στη main. Πλέον ο μεταγλωττιστής θα διαμαρτυρηθεί αν προσπαθήσουμε να προσπελάσουμε τη μεταβλητή manufacturer, αφού είναι private. Ο σωστός τρόπος για να αλλάξουμε την τιμή της είναι μέσω της χρήσης της κατάλληλης public μεθόδου.

```
1 class Vehicle {      // Ορισμός κλάσης vehicle
2        // Δήλωση πεδίων
3      private String manufacturer;
4      private String platesNumber = "Unknown";     // Αρχική τιμή, άγνωστη πινακίδα
5      // Μία κατασκευάστρια μέθοδος
6      public Vehicle() {
7              this("Unknown");       // Κλήση 2ης κατασκευάστριας μεθόδου
8      }
9      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί τον κατασκευαστή
10     public Vehicle(String manufacturer) {
11             this.manufacturer = manufacturer;     // Αρχικοποίηση
12     }       // Τέλος κατασκευάστριας μεθόδου
13     // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί όλα τα πεδία
14     public Vehicle(String manufacturer, String platesNumber) {
15             // Αρχικοποιήσεις
16             this.manufacturer = manufacturer;
17             this.platesNumber = platesNumber;
18     }       // Τέλος κατασκευάστριας μεθόδου
19     // Μέθοδος που επιστρέφει τον κατασκευαστή
20     public String getManufacturer(){
21             return manufacturer;
22     }       // Τέλος μεθόδου
23     // Μέθοδος που δίνει τιμή στον κατασκευαστή και δεν επιστρέφει τίποτα
24     public void setManufacturer(String manufacturer){
25             this.manufacturer = manufacturer;
26     }       // Τέλος μεθόδου
27     // Μέθοδος που επιστρέφει την πινακίδα
28     public String getPlatesNumber(){
29             return platesNumber;
30     }       // Τέλος μεθόδου
31     // Μέθοδος που δίνει τιμή στην πινακίδα και δεν επιστρέφει τίποτα
32     public void setPlatesNumber(String platesNumber){
33             this.platesNumber = platesNumber;
34     }       // Τέλος μεθόδου
35     // Μέθοδος που τυπώνει πληροφορίες για το όχημα
36     public String info() {
37             String s = "Vehicle: " + manufacturer + " " + platesNumber;
38             return s;
39     }
40 }    // Τέλος κλάσης
```

```
1 public class MyProgram {
2      //Ένα πρόγραμμα ξεκινά πάντα με την κλήση της main
3      public static void main(String[] args) {
4              Vehicle vehicle1 = new Vehicle(); // Δημιουργία αντικειμένου
5              Vehicle vehicle2 = new Vehicle("Mazerati", "HPA-0000");
6              // Αναφορά σε μεταβλήτή στιγμιοτύπου και ανάθεση νέας τιμής
7              //vehicle1.manufacturer = new String("Peugeuot"); //Λάθος, private
8              vehicle1.setManufacturer("Peugeuot"));
9              System.out.println(vehicle1.info()); // Κλήση μεθόδων
10             System.out.println(vehicle2.info()); // Κλήση μεθόδων
11     }       // Τέλος της main
12 }
```

5.1.3. Κληρονομικότητα (Inheritance)

Η κληρονομικότητα αποτελεί ένα από τα σημαντικότερα χαρακτηριστικά του αντικειμενοστρεφούς προγραμματισμού. Για παράδειγμα μπορούμε να δημιουργήσουμε μια γενική κλάση η οποία ορίζει μια σειρά από κοινά χαρακτηριστικά (πεδία ή λειτουργίες) για ένα ευρύ σύνολο αντικειμένων. Αυτή η κλάση μπορεί κατόπιν να κληρονομηθεί από άλλες, πιο εξειδικευμένες κλάσεις, προσθέτοντας η κάθε μια εκείνα τα χαρακτηριστικά (πεδία ή λειτουργίες) τα οποία την διαφοροποιούν από τις άλλες. Στην ορολογία της Java αυτή η κλάση ονομάζεται υπερκλάση και αυτές που την κληρονομούν ονομάζονται υποκλάσεις. Γενικά μπορούμε να δημιουργούμε ιεραρχίες κλάσεων πολλών επιπέδων. Αν μία κλάση έχει δηλωθεί ως final, τότε δεν μπορεί να κληρονομηθεί από άλλες κλάσεις. Ένας περιορισμός που έχουμε στη Java είναι ότι μια κλάση μπορεί να έχει το πολύ μια υπερκλάση (αλλά απεριόριστο αριθμό υποκλάσεων). Αυτό ονομάζεται single inheritance.

Οι public και protected μέθοδοι της υπερκλάσης κληρονομούνται αυτόματα και στην υποκλάση, ενώ η υποκλάση μπορεί να αναφερθεί απευθείας στα public και protected πεδία της υπερκλάσης. Στο παράδειγμα μας παρατηρείστε ότι χρησιμοποιούμε το όνομα της υποκλάσης ακολουθούμενο από τη λέξη κλειδί **extends** και το όνομα της υπερκλάσης την οποία κληρονομεί, όπως φαίνεται στα παραδείγματα παρακάτω. Συγκεκριμένα δηλώνουμε την κλάση Car ως υποκλάση της κλάσης Vehicle. Στη νέα κλάση προσθέτουμε τις ιδιότητες του κυβισμού, της ιπποδύναμης και του αριθμού πορτών. Επίσης μπορούμε να εξειδικεύσουμε ακόμα περισσότερο την κλάση Car, δημιουργώντας την κλάση Truck, η οποία έχει ένα επιπλέον πεδίο σε σχέση με την κλάση Car, το πεδίο loadCapacity (ικανότητα φόρτωσης) το οποίο μπορεί να εκφράζεται σε κιλά.

```
1 public class Car extends Vehicle{
2       private int cc = 0;    // Νέα πεδία της Car σε σχέση με τη Vehicle
3       private int horsePower = 0;
4       private int numberOfDoors = 0;
5       // Μία κατασκευάστρια μέθοδος
6       public Car() {
7               super();       // Κλήση κατασκευάστριας μεθόδου της υπερκλάσης Vehicle
8       }
9       // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί τον κατασκευαστή
10      public Car(String manufacturer) {
11              super(manufacturer);   // Κλήση κατασκευάστριας υπερκλάσης Vehicle
12      }       // Τέλος κατασκευάστριας μεθόδου
13      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί όλα τα πεδία
14      public Car(String manufacturer, String platesNumber, int cc,
15      int horsePower, int doors) {
16              // Κλήση constructor υπερκλάσης, Vehicle
17              super(manufacturer, platesNumber);
18              // Αρχικοποιήσεις
19              this.cc = cc;
20              this.horsePower = horsePower;
21              numberOfDoors = doors;
22      }       // Τέλος κατασκευάστριας μεθόδου
23      // Μέθοδος που επιστρέφει τον κυβισμό
24      public int getCC(){
25              return cc;
26      }       // Τέλος μεθόδου
27      // Μέθοδος που δίνει τιμή στον κυβισμό και δεν επιστρέφει τίποτα
28      public void setCC(int cc){
29              this.cc = cc;
30      }       // Τέλος μεθόδου
31      // Μέθοδος που επιστρέφει την ιπποδύναμη
32      public int getHorsePower(){
33              return horsePower;
34      }       // Τέλος μεθόδου
```

```
35      // Μέθοδος που δίνει τιμή στην ιπποδύναμη και δεν επιστρέφει τίποτα
36      public void setHorsePower(int horsePower){
37              this.horsePower = horsePower;
38      }       // Τέλος μεθόδου
39      // Μέθοδος που επιστρέφει τον αριθμό πορτών
40      public int getNumberOfDoors(){
41              return numberOfDoors;
42      }       // Τέλος μεθόδου
43      // Μέθοδος που δίνει τιμή στον αριθμό πορτών και δεν επιστρέφει τίποτα
44      public void setNumberOfDoors(int doors){
45              numberOfDoors = doors;
46      }       // Τέλος μεθόδου
47}
```

Στη γραμμή 1 ορίζουμε την κλάση Car η οποία κληρονομεί την κλάση Vehicle και στις γραμμές 2-4 ορίζουμε ως private τα νέα πεδία της. Στη γραμμή 6 καλείται η κατασκευάστρια μέθοδος της κλάσης Car, η οποία χρησιμοποιώντας τη λέξη κλειδί **super** καλεί την κατασκευάστρια μέθοδο της υπερκλάσης της, Vehicle. Αντίστοιχα και για τις άλλες κατασκευάστριες μεθόδους. Επιπλέον, χρησιμοποιώντας τη ως super.<όνομα μέλους>, μπορούμε να καλέσουμε μεθόδους ή πεδία της υπερκλάσης.

Στη συνέχεια δηλώνουμε τις μεθόδους για την ανάθεση και την επιστροφή των πεδίων της.

```
1 public class Truck extends Car {
2       //Νέα πεδία σε σχέση με τη Car
3       private static int numberOfTrucks = 0;        //Static μεταβλητή
4       private int loadCapacity = 0;
5       // Κατασκευάστρια μέθοδος
6       public Truck() {
7               super(); // Κλήση constructor υπερκλάσης Car
8               numberOfTrucks++;       // Καινούργιο φορτηγό
9       }
10      // Ακόμα μία κατασκευάστρια μέθοδος
11      public Truck(String manufacturer, int capacity) {
12              super(manufacturer);   // Κλήση constructor υπερκλάσης Car
13              loadCapacity = capacity;
14              numberOfTrucks++;       // Καινούργιο φορτηγό
15      }
16      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί όλα τα πεδία
17      public Truck(String manufacturer, String platesNumber, int cc, int horsePower,
18              int doors, int capacity) {
19              // Κλήση constructor υπερκλάσης, Car
20              super(manufacturer, platesNumber, cc, horsePower, doors);
21              loadCapacity = capacity;
22              numberOfTrucks++;       // Καινούργιο φορτηγό
23      }       // Τέλος κατασκευάστριας μεθόδου
24      // Επιστρέφει τον αριθμό φορτηγών που έχουν δημιουργηθεί
25      public static int getNumberOfTrucks() {
26              return numberOfTrucks;
27      }       // Τέλος μεθόδου
28      // Μέθοδος που επιστρέφει τον αριθμό πορτών
29      public int getLoadCapacity(){
30              return loadCapacity;
31      }       // Τέλος μεθόδου
32      // Μέθοδος που δίνει τιμή στον αριθμό πορτών και δεν επιστρέφει τίποτα
33      public void setLoadCapacity(int capacity){
34              loadCapacity = capacity;
35      }       // Τέλος μεθόδου
36 }
```

Όπως έχουμε δει, για να προσπελάσουμε ένα μέλος μιας κλάσης, πρέπει πρώτα να δημιουργήσουμε ένα αντικείμενό της. Χρησιμοποιώντας τη λέξη κλειδί **static** μπορούμε να δηλώσουμε μέλη (πεδία ή μεθόδους) τα οποία μπορούμε να χρησιμοποιήσουμε χωρίς να έχουμε δημιουργήσει ένα αντικείμενο της κλάσης (στην ουσία τα μέλη αυτά αφορούν την κλάση και όχι τα στιγμιότυπα της). Μια τέτοια

περίπτωση παρατηρούμε στη κλάση Truck. Παρατηρήστε στη γραμμή 3 και 25 τη δήλωση της μεταβλητής numberOfTrucks και αντίστοιχα της μεθόδου getNumberOfTrucks ως static. Η numberOfTrucks χρησιμοποιείται για να κρατά το πλήθος των αντικειμένων που έχουν δημιουργηθεί και για να το επιτύχουμε αυξάνουμε την τιμή της κατά ένα όταν καλείται κάποια κατασκευάστρια μέθοδος.

```
1 public class MyProgram {
2  //Ένα πρόγραμμα ξεκινά πάντα με την κλήση της main
3  public static void main(String a[]) {
4       // Δημιουργία 3 φορτηγών
5       Truck t1 = new Truck();
6       Truck t2 = new Truck();
7       Truck t3 = new Truck();
8       // Δημιουργία αυτοκινήτου.
9       Car c1 = new Car("Audi","HPA-007",1800,120,4);
10      System.out.println(c1.info());
10      // Ένα φορτηγό είναι αυτοκίνητο
11      Car c2 = t1; // Αναφορά στο t1
12      //Truck t4 = c2; // Λάθος (Ένα αυτοκίνητο δεν είναι φορτηγό
13      t1.setManufacturer("Volvo");
14      System.out.println(c2.getManufacturer());
15      System.out.println("Υπάρχουν " + Truck.getNumberOfTrucks() + " φορτηγά");
16  }   // Τέλος της main
17}
```

Στην ενημερωμένη main δημιουργούνται αρχικά τρία Trucks. Στη συνέχεια δημιουργείται ένα Car, του οποίου τυπώνουμε το αποτέλεσμα της κλήσης της μεθόδου info(). Έπειτα στη γραμμή 11 κάνουμε ανάθεση της μεταβλητής t1 τύπου Truck, στη μεταβλητή c2 τύπου Car. Αυτό είναι σωστό διότι ένα Truck είναι και Car, αφού το κληρονομεί. Ανάποδα ένα Car δεν είναι Truck και ο μεταγλωττιστής θα διαμαρτυρηθεί (γραμμή 12). Σημειώστε εδώ ότι οι μεταβλητές κρατάνε μία αναφορά (reference) στο αντικείμενο και όχι μία αντιγραφή του αντικειμένου. Για αυτό στη γραμμή 14 τυπώνεται η τιμή Volvo, αφού στην προηγούμενη γραμμή έχει γίνει ανάθεση αυτής της τιμής στο αντικείμενο μέσω του t1. Αντίστοιχα και οι παράμετροι σε μία μέθοδο περνάνε τις τιμές με reference. Τέλος στη γραμμή 15 θα τυπωθεί το μήνυμα "Υπάρχουν 3 φορτηγά", αφού έχουν κληθεί οι κατασκευάστριες μέθοδοι της κλάσης Truck τρεις φορές, και άρα η static μεταβλητή numberOfTrucks έχει την τιμή 3. Πιο συγκεκριμένα η έξοδος της main είναι η παρακάτω:

```
Vehicle: Audi HPA-007
Volvo
Υπάρχουν 3 φορτηγά
```

5.1.4. Διεπαφές (Interfaces)

Μια **διεπαφή** (**interface**) αποτελείται από ένα όνομα και από ένα σύνολο επικεφαλίδων μεθόδων. Για κάθε μέθοδο ορίζουμε μόνο το όνομα της, τον επιστρεφόμενο τύπο της και τις παραμέτρους της. Τα σώματα των μεθόδων δεν δηλώνονται στη διεπαφή. Μία διεπαφή ορίζεται χρησιμοποιώντας τη λέξη κλειδί **interface.** Παρακάτω βλέπουμε τον ορισμό μιας διεπαφής Ownership.

```
1 public interface Ownership{ //Δήλωση της διεπαφής
2       public Object getOwner();//Οι μέθοδοι που πρέπει να υλοποιηθούν
3       public void setOwner(Object owner);
4       public int getPrice();
5       public void setPrice(int price);
6 }
```

Παρατηρείστε ότι οι παραπάνω μέθοδοι δεν έχουν σώματα. Τα σώματα των μεθόδων ορίζονται στις κλάσεις οι οποίες δηλώνουν ότι υλοποιούν την διεπαφή. Κάθε κλάση που δηλώνει ότι υλοποιεί τη συγκεκριμένη διεπαφή, θα πρέπει να υλοποιεί και τις τέσσερεις μεθόδους της. Στη συνέχεια δίνεται η υλοποίηση της κλάσης Vehicle ώστε να υλοποιεί τη διεπαφή Ownership. Παρατηρήστε ότι αυτό γίνεται γράφοντας τη λέξη κλειδί **implements** και στη συνέχεια το όνομα της διεπαφής.

Προσέξτε ότι η setOwner και getOwner δέχεται ως παράμετρο και επιστρέφει ένα αντικείμενο τύπου Object, το οποίο αποτελεί και την υπερκλάση κάθε αντικειμένου στη Java.

```
1 class Vehicle implements Ownership{ // Ορισμός κλάσης vehicle
2        // Δήλωση πεδίων
3       protected String manufacturer;
4       private String platesNumber = "Unknown";      // Αρχική τιμή
5       private String owner = "Unknown";
6       private int price;
7       // Μία κατασκευάστρια μέθοδος
8       public Vehicle() {
9               this("Unknown");       // Κλήση 2ης κατασκευάστριας μεθόδου
10      }
11      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί τον κατασκευαστή
12      public Vehicle(String manufacturer) {
13              this.manufacturer = manufacturer;     // Αρχικοποίηση
14      }       // Τέλος κατασκευάστριας μεθόδου
15      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί όλα τα πεδία
16      public Vehicle(String manufacturer , String platesNumber) {
17              // Αρχικοποιήσεις
18              this.manufacturer = manufacturer;
19              this.platesNumber = platesNumber;
20      }       // Τέλος κατασκευάστριας μεθόδου
21      // Μέθοδος που επιστρέφει τον κατασκευαστή
22      public String getManufacturer(){
23              return manufacturer;
24      }       // Τέλος μεθόδου
25      // Μέθοδος που δίνει τιμή στον κατασκευαστή και δεν επιστρέφει τίποτα
26      public void setManufacturer(String manufacturer){
27              this.manufacturer = manufacturer;
28      }       // Τέλος μεθόδου
29      // Μέθοδος που επιστρέφει την πινακίδα
30      public String getPlatesNumber(){
31              return platesNumber;
32      }       // Τέλος μεθόδου
33      // Μέθοδος που δίνει τιμή στην πινακίδα και δεν επιστρέφει τίποτα
34      public void setPlatesNumber(String platesNumber){
35              this.platesNumber = platesNumber;
36      }       // Τέλος μεθόδου
37      // Μέθοδος που τυπώνει πληροφορίες για το όχημα
38      public String info() {
39              String s = "Vehicle: " + manufacturer + " " + platesNumber;
40              return s;
41      }
42      // Υλοποίηση μεθόδων του interface
43      public String getOwner(){
44              return owner;
45      }
46      public void setOwner(Object owner){
47              // Εάν το αντικείμενο owner είναι κλάσης String
48              if(owner instanceof String)
49                      this.owner = (String) (owner); // Μετατροπή από Object σε String
50              else
51                      System.out.println("Δεν μπορεί να εισαχθεί ο ιδιοκτήτης");
52      }
53      public int getPrice(){
54              return price;
55      }
56      public void setPrice(int price){
57              this.price = price;
58      }
59}     // Τέλος κλάσης
```

Θα μπορούσαμε να πούμε ότι η δήλωση «implements Owner» σημαίνει ότι η Vehicle κληρονομεί την υποχρέωση υλοποίησης των μεθόδων που περιέχει η διεπαφή Owner.

Μια κλάση έχει την δυνατότητα να υλοποιήσει όσες διεπαφές επιθυμεί ο προγραμματιστής της (π.χ. class Boat implements Owner, Serializable, Clonable). Θα μπορούσαμε να πούμε ότι αυτή η δυνατότητα έρχεται να απαλύνει τον περιορισμό της απλής κληρονομικότητας.

Στο παράδειγμα παραπάνω προσέξτε επίσης και τη χρήση της λέξης κλειδί **instanceof** στη γραμμή 48, η οποία μας ενημερώνει αν ένα αντικείμενο είναι στιγμιότυπο μιας κλάσης (στην περίπτωσή μας αν το αντικείμενο owner είναι τύπου String). Στη συνέχεια θα θέλαμε να αναθέσουμε ένα αντικείμενο τύπου Object σε μία μεταβλητή τύπου String, κάτι το οποίο όμως απαγορεύεται. Η κλάση Object όμως, όπως είπαμε είναι η υπερκλάση κάθε αντικειμένου στη γλώσσα Java και η instanceof στον έλεγχο της if μας έχει διαβεβαιώνει ότι η owner είναι τύπου String. Άρα μπορούμε να χρησιμοποιήσουμε το τελεστή αλλαγής τύπου (casting) **(type)**, όπως φαίνεται στη γραμμή 49.

Η συγκεκριμένη διεπαφή θα μπορούσε να χρησιμοποιηθεί σε οποιαδήποτε κλάση που έχει κάποιο ιδιοκτήτη, π.χ. Vehicle, Hotel, House κ.ο.κ.. Τι κερδίζουμε; Μπορούμε να ορίσουμε κλάσεις και μεθόδους που να διαχειρίζονται αντικείμενα τύπου Ownership. Αυτές τις κλάσεις και μεθόδους θα μπορούμε να τις χρησιμοποιήσουμε με αντικείμενα τύπου Hotel, Vehicle, House (γενικά με στιγμιότυπα κλάσεων που υλοποιούν το interface Ownership). Ένα μικρό παράδειγμα ακολουθεί:

```
1 public class Contract {
2       Ownership property;
3       Person seller;
4       Person buyer;
5       Contract(Person seller, Person buyer, Ownership property) {
6               this.seller = seller;
7               this.buyer = buyer;
8               this.property = property;
9       }
10      void makeContract() {
11              if (property.getOwner() == seller)
12                      property.setOwner(buyer);
13              else
14                      System.out.println("Cannot make this contract!");
15      }
16}
```

5.1.5. Αφηρημένες κλάσεις

Υπάρχουν περιπτώσεις στις οποίες θέλουμε να δηλώσουμε μία υπερκλάση η οποία δεν χρειάζεται να έχει υλοποιημένες όλες τις μεθόδους της. Αυτό μπορούμε να το πετύχουμε χρησιμοποιώντας αφηρημένες κλάσεις. Σε αντίθεση με τις διεπαφές, οι αφηρημένες κλάσεις επιτρέπουν την υλοποίηση μεθόδων οι οποίες όμως δεν έχουν δηλωθεί σαν αφηρημένες. Οι αφηρημένες κλάσεις είναι χρήσιμες σε περίπτωση που θέλουμε να υλοποιήσουμε κλάσεις οι οποίες έχουν ίδιες μεθόδους (όχι μόνο από άποψη υπογραφών, αλλά και λειτουργικότητας). Αν χρησιμοποιούσαμε διεπαφές θα έπρεπε να υλοποιήσουμε τις μεθόδους αυτές σε κάθε κλάση, γράφοντας τον ίδιο κώδικα σε κάθε μία. Χρησιμοποιώντας όμως αφηρημένες κλάσεις, αυτός ο κοινός κώδικας γράφεται μόνο στην αφηρημένη κλάση την οποία στη συνέχεια κληρονομούν. Για να οριστεί ότι μία κλάση είναι αφηρημένη, χρησιμοποιείται η λέξη

κλειδί **abstract** τόσο στη δήλωση της κλάσης όσο και στις μεθόδους που πρέπει να υλοποιήσουν οι κλάσεις που την κληρονομούν. Σε περίπτωση που οι κλάσεις που την κληρονομούν δεν υλοποιήσουν κάποια από τις abstract μεθόδους, ο μεταγλωττιστής θα διαμαρτυρηθεί. Επίσης δεν μπορούμε να δημιουργήσουμε στιγμιότυπα μιας αφηρημένης κλάσης. Παρακάτω βλέπουμε την έκδοση της Vehicle η οποία κληρονομεί την abstract OwnershipAbstract.

```
1 public abstract class OwnershipAbstract {
2       private int price = 0;
3       public int getPrice() {
4               return price;
5       }
6       public void setPrice(int price) {
7               this.price = price;
8       }
9       //Οι μέθοδοι που πρέπει να υλοποιηθούν από τις κλάσεις που την κληρονομούν
10      abstract public Object getOwner();
11      abstract public void setOwner(Object owner);
12 }
```

```
1 class Vehicle extends OwnershipAbstract{   // Ορισμός κλάσης vehicle
2        // Δήλωση πεδίων
3       protected String manufacturer;
4       private String platesNumber = "Unknown";     // Αρχική τιμή
5       private String owner = "Unknown";
6       // Μία κατασκευάστρια μέθοδος
7       public Vehicle() {
8               this("Unknown");       // Κλήση 2ης κατασκευάστριας μεθόδου
9       }
10      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί τον κατασκευαστή
11      public Vehicle(String manufacturer) {
12              this.manufacturer = manufacturer;     // Αρχικοποίηση
13      }       // Τέλος κατασκευάστριας μεθόδου
14      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί όλα τα πεδία
15      public Vehicle(String manufacturer , String platesNumber) {
16              // Αρχικοποιήσεις
17              this.manufacturer = manufacturer;
18              this.platesNumber = platesNumber;
19      }       // Τέλος κατασκευάστριας μεθόδου
20      // Μέθοδος που επιστρέφει τον κατασκευαστή
21      public String getManufacturer(){
22              return manufacturer;
23      }       // Τέλος μεθόδου
24      // Μέθοδος που δίνει τιμή στον κατασκευαστή και δεν επιστρέφει τίποτα
25      public void setManufacturer(String manufacturer){
26              this.manufacturer = manufacturer;
27      }       // Τέλος μεθόδου
28      // Μέθοδος που επιστρέφει την πινακίδα
29      public String getPlatesNumber(){
30              return platesNumber;
31      }       // Τέλος μεθόδου
32      // Μέθοδος που δίνει τιμή στην πινακίδα και δεν επιστρέφει τίποτα
33      public void setPlatesNumber(String platesNumber){
34              this.platesNumber = platesNumber;
35      }       // Τέλος μεθόδου
36      // Μέθοδος που τυπώνει πληροφορίες για το όχημα
37      public String info() {
38              String s = "Vehicle: " + manufacturer + " " + platesNumber;
39              return s;
40      }
41      public String getOwner(){
42              return owner;
43      }
44      public void setOwner(Object owner){
45              // If object is a string
46              if(owner instanceof String)
47                      this.owner = (String) (owner);// Μετατροπή από Object σε String
48              else
49                      System.out.println("Δεν μπορεί να εισαχθεί ο ιδιοκτήτης");
50      }
51}     // Τέλος κλάσης
```

5.2. Πολυμορφισμός (Polymorphism)

Ο πολυμορφισμός στη Java επιτυγχάνεται με στατικό και δυναμικό τρόπο. Ο πρώτος ονομάζεται υπερφόρτωση (overloading) και ο δεύτερος επικάλυψη ή υποσκελισμός (overriding).

5.2.1. Υπερφόρτωση (Overloading)

Η υπερφόρτωση μιας μεθόδου ή της κατασκευάστριας μεθόδου μιας κλάσης αποτελεί έναν τύπο στατικού πολυμορφισμού όπως ήδη αναφέραμε. Υπερφόρτωση είναι η εκ νέου δήλωση μιας ήδη υπάρχουσας μεθόδου με το ίδιο όνομα αλλά με διαφορετικό πλήθος ή τύπο παραμέτρων, και τον ίδιο τύπο επιστρεφόμενης τιμής.

Στα προηγούμενα παραδείγματα, υπερφόρτωση χρησιμοποιείται στις κατασκευάστριες μεθόδους των κλάσεων. Όπως θα έχετε παρατηρήσει υπάρχουν κατασκευάστριες μέθοδοι οι οποίες δεν δέχονται καμία παράμετρο, άλλες που δέχονται μία συμβολοσειρά και άλλες που δέχονται 2 συμβολοσειρές. Αντίστοιχα, θα μπορούσαμε να ορίσουμε μεθόδους με διαφορετικές παραμέτρους αλλά προσοχή, την ίδια επιστρεφόμενη τιμή.

5.2.2. Υποσκελισμός (Overriding)

Μια υποκλάση κληρονομώντας μια υπερκλάση, αυτομάτως μπορεί να χρησιμοποιήσει και τις public και protected μεθόδους της. Υπάρχει όμως η πιθανότητα η υποκλάση να θέλει να εκτελέσει μια συγκεκριμένη ενέργεια που κληρονομεί, με διαφορετικό τρόπο. Σε αυτή την περίπτωση η υποκλάση έχει τη δυνατότητα να παραμερίσει την υλοποίηση της υπερκλάσης για τη συγκεκριμένη μέθοδο, υλοποιώντας τη με διαφορετικό τρόπο και κρατώντας τα χαρακτηριστικά (όνομα, ορίσματα και επιστρεφόμενο τύπο) της υπερκλάσης ίδια. Αυτό ονομάζεται υποσκελισμός και μπορεί να γίνει με την προϋπόθεση ότι η μέθοδος δεν έχει δηλωθεί final στην υπερκλαση. Ο υποσκελισμός αποτελεί μορφή δυναμικού πολυμορφισμού και εφαρμόζεται κατά το χρόνο εκτέλεσης του προγράμματος. Ακολουθεί παράδειγμα όπου η κλάση Car υποσκελίζει τη μέθοδο info που κληρονομεί από τη Vehicle.

```
1 public class Car extends Vehicle{
2       private int cc = 0;            // Νέα πεδία της Car σε σχέση με τη Vehicle
3       private int horsePower = 0;
4       private int numberOfDoors = 0;
5       // Μία κατασκευάστρια μέθοδος
6       public Car() {
7               super();        // Κλήση κατασκευάστριας υπερκλάσης Vehicle
8       }
9       // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί τον κατασκευαστή
10      public Car(String manufacturer) {
11              super(manufacturer);   // Κλήση κατασκευάστριας υπερκλάσης Vehicle
12      }       // Τέλος κατασκευάστριας μεθόδου
13      // Ακόμα μία κατασκευάστρια μέθοδος, αρχικοποιεί όλα τα πεδία
14      public Car(String manufacturer, String platesNumber, int cc,
15              int horsePower, int doors) {
16              // Κλήση constructor υπερκλάσης, Vehicle
17              super(manufacturer, platesNumber);
18              // Αρχικοποιήσεις
19              this.cc = cc;
20              this.horsePower = horsePower;
21              numberOfDoors = doors;
22      }       // Τέλος κατασκευάστριας μεθόδου
23      // Μέθοδος που επιστρέφει τον κυβισμό
24      public int getCC(){
25              return cc;
```

```
26      }       // Τέλος μεθόδου
27      // Μέθοδος που δίνει τιμή στον κυβισμό και δεν επιστρέφει τίποτα
28      public void setCC(int cc){
29              this.cc = cc;
30      }       // Τέλος μεθόδου
31      // Μέθοδος που επιστρέφει την ιπποδύναμη
32      public int getHorsePower(){
33              return horsePower;
34      }       // Τέλος μεθόδου
35      // Μέθοδος που δίνει τιμή στην ιπποδύναμη και δεν επιστρέφει τίποτα
36      public void setHorsePower(int horsePower){
37              this.horsePower = horsePower;
38      }       // Τέλος μεθόδου
39      // Μέθοδος που επιστρέφει τον αριθμό πορτών
40      public int getNumberOfDoors(){
41              return numberOfDoors;
42      }       // Τέλος μεθόδου
43      // Μέθοδος που δίνει τιμή στον αριθμό πορτών και δεν επιστρέφει τίποτα
44      public void setNumberOfDoors(int doors){
45              numberOfDoors = doors;
46      }       // Τέλος μεθόδου
47      // Μέθοδος που τυπώνει πληροφορίες για το αυτοκίνητο, Overriding
48      public String info() {
49              String s = "Car: " + getManufacturer() + " " + getPlatesNumber();
50              s += " " + cc + " " + horsePower + " " + numberOfDoors;
51              return s;
52      }
53 }
```

Στη γραμμή 48 γίνεται ο υποσκελισμός της μεθόδου info της Vehicle, έτσι ώστε να μας δώσει πληροφορίες για τα κυβικά, την ιπποδύναμη και τον αριθμό πορτών, χαρακτηριστικών διαθέσιμων στην κλάση Car.

```
1 public class MyProgramOverriding{
2       //Ένα πρόγραμμα ξεκινά πάντα με την κλήση της main
3       public static void main(String a[]) {
4               // Δημιουργία οχήματος
5               Vehicle t1 = new Vehicle("Kawasaki", "HPA-000");
6               // Δημιουργία αυτοκινήτου.
7               Car c1 = new Car("Audi","HPA-007",1800,120,4);
8               System.out.println(t1.info());
9               System.out.println(c1.info());
10      }       // Τέλος της main
11}
```

Η εξοδος του προγράμματος είναι η εξής:

```
Vehicle: Kawasaki HPA-000
Car: Audi HPA-007 1800 120 4
```

5.3. Ενθυλάκωση – ADTs

Η ενθυλάκωση είναι ένας μηχανισμός με τον οποίο τα πεδία και οι μέθοδοι ενός αντικειμένου κρατούνται κρυμμένα μέσα στο ίδιο το αντικείμενο και ασφαλή από εξωτερικές παρεμβάσεις. Ο χρήστης του αντικειμένου χρειάζεται να γνωρίζει μόνο τον τρόπο επικοινωνίας μαζί του και όχι τις εσωτερικές του λειτουργίες. Η ιδιαίτερη ικανότητα αυτού του μηχανισμού είναι ότι όλοι γνωρίζουν πώς να προσπελάσουν ένα αντικείμενο χωρίς να φοβούνται τυχόν απροσδόκητα αποτελέσματα.

Στα παραδείγματα που ορίσαμε παραπάνω είδατε ότι τα πεδία μιας κλάσης ήταν ορισμένα ως private. Με αυτό τον τρόπο δεν μπορούσαμε να τα προσπελάσουμε απευθείας εκτός της κλάσης, π.χ. μέσα στη main. Για το λόγο αυτό, κάθε κλάση είχε τις κατάλληλες public μεθόδους get και set, έτσι ώστε να είναι δυνατή η διαχείριση των τιμών τους.

Χρησιμοποιώντας ενθυλάκωση μπορούμε να δημιουργήσουμε αφηρημένους τύπους δεδομένων, ή αλλιώς **ADTs (**Abstract Data Type). Ένα ADT είναι μια περιγραφή ενός συνόλου δεδομένων καθώς και του συνόλου των λειτουργιών που μπορούν να εκτελεστούν πάνω του. Ένας τέτοιος τύπος δεδομένων είναι αφηρημένος υπο την έννοια ότι είναι ανεξάρτητος από διάφορες συγκεκριμένες υλοποιήσεις. Οι χρήστες ενός ADT ασχολούνται μόνο με το πως θα αλληλεπιδράσουν μαζί του και όχι με το πώς αυτό υλοποιείται, αφού η υλοποίηση μπορεί να αλλάξει μελλοντικά. Αυτό σημαίνει ότι οποιεσδήποτε τροποποιήσεις στην υλοποίηση του ADT δεν επηρεάζουν το πρόγραμμα που το χρησιμοποιεί.

Τα θετικά των ADTs μπορούν να συνοψιστούν στα παρακάτω:

- Διαχωρίζουν το σκοπό και τη χρήση ενός μέρους του προγράμματος από την υλοποίησή του.
- Κρύβουν πληροφορίες όσον αφορά την υλοποίηση ενός μέρους του προγράμματος και τις κάνει μη προσβάσιμες από το προγραμματιστή.
- Μειώνουν την πολυπλοκότητα μεγάλων προγραμμάτων με το να καθορίζουν συστηματικά πως αλληλεπιδρούν τα διάφορα μέρη του προγράμματος μεταξύ τους.

Για παράδειγμα, ένα δυαδικό δέντρο ADT μπορεί να αναπαρασταθεί με πολλούς τρόπους: δυαδικό δέντρο, AVL δέντρο, red-black δέντρο, πίνακας, etc. Ανεξάρτητα όμως από την υλοποίηση, ένα δυαδικό δέντρο έχει πάντα κάποιες συγκεκριμένες λειτουργίες π.χ. (εισαγωγή, διαγραφή, εύρεση, κ.τ.λ.).

5.4. Εξαιρέσεις

Κατά την διάρκεια εκτέλεσης ενός προγράμματος μπορεί να εμφανιστούν λογικά λάθη τα οποία οδηγούν στον απρόσμενο τερματισμό του. Η Java μας δίνει την δυνατότητα να διαχειριστούμε τέτοια λάθη με το μηχανισμό των εξαιρέσεων.

Μια εξαίρεση είναι ένα αντικείμενο που περιέχει πληροφορίες για το σφάλμα. Στην περίπτωση όπου μια μέθοδος αντιμετωπίσει κάποιο πρόβλημα στο εσωτερικό της, παράγει ένα αντικείμενο εξαίρεσης. Οι περιπτώσεις από τις οποίες μπορεί να προέλθει μια εξαίρεση είναι πολλές και για το λόγο αυτό η Java έχει δημιουργήσει διαφορετικού τύπου εξαιρέσεις ανάλογα με το πρόβλημα. Αν για παράδειγμα προσπαθήσουμε να διαιρέσουμε έναν αριθμό με το μηδέν, κάτι που μαθηματικά είναι αδύνατο, θα έχουμε ένα *ArithmeticException*. Στην περίπτωση που προσπαθήσουμε να προσπελάσουμε ένα στοιχείο ενός πίνακα το οποίο είναι εκτός των ορίων του, θα έχουμε αντίστοιχα ένα *ArrayIndexOutOfBoundsException*.

Το πρόγραμμα πρέπει να χειρίζεται με λογικό τρόπο τις εξαιρέσεις οι οποίες προκύπτουν κατά το χρόνο εκτέλεσης. Η διαχείριση των εξαιρέσεων γίνεται μέσω των λέξεων κλειδιών: **try**, **catch**, **throw**, **throws**, και **finally**. Ο κώδικας ο οποίος υπάρχει πιθανότητα να προκαλέσει μια εξαίρεση περικλείεται μέσα σε ένα try block ακολουθούμενο από ένα catch block, στο οποίο περνάει η εκτέλεση του προγράμματος στην περίπτωση που συμβεί η εξαίρεση. Ένα try block μπορεί να ακολουθείται από παραπάνω του ενός catch blocks στην περίπτωση όπου θέλουμε να διαχειριστούμε παραπάνω από ενός είδους εξαιρέσεις. Επιπλέον, στην περίπτωση που πρέπει οπωσδήποτε να εκτελεστούν κάποιες ενέργειες ακόμα και αν προκληθεί

κάποια εξαίρεση, περικλείουμε τις ενέργειες αυτές μέσα σε ένα finally block. Εδώ βλέπουμε μια γενική φόρμα χειρισμού των εξαιρέσεων:

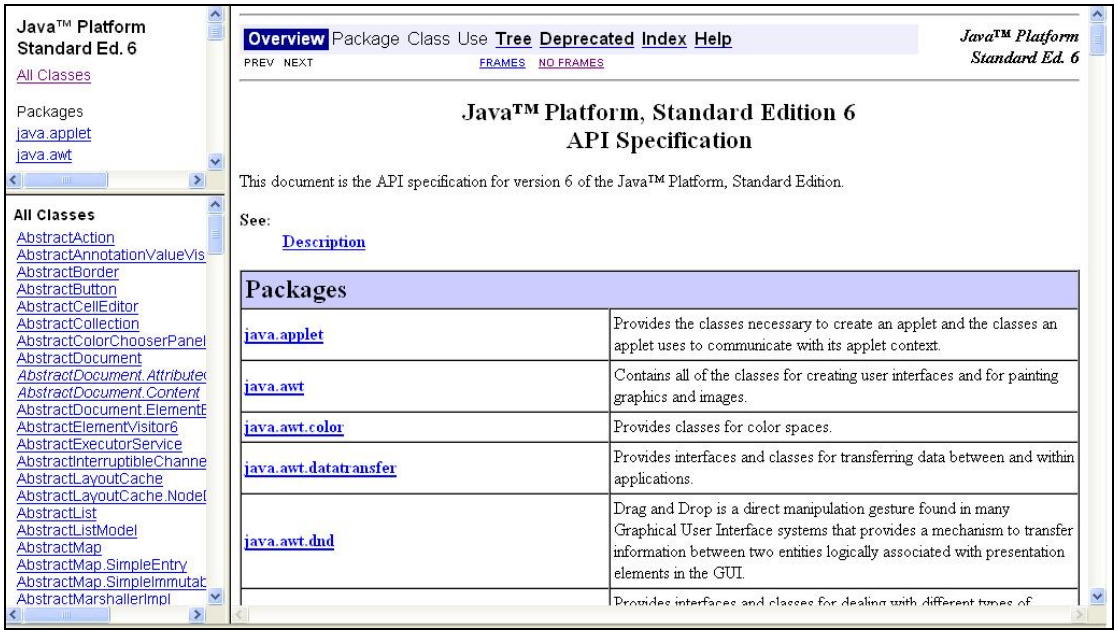
```
1 try{
2       //ο κώδικας που ενδέχεται να προκαλέσει μια εξαίρεση
3 }
4 catch(Τύπος_Εξαίρεσης e){ //π.χ. ArithmeticException
5       //διαχείριση της εξαίρεσης
6 }
7 finally{
8       //ο κώδικας που θα εκτελεστεί πριν τερματίσει το πρόγραμμα
9 }
```

Στις περιπτώσεις που περιγράψαμε μέχρι τώρα, το σύστημα δημιουργεί αυτόματα το αντικείμενο εξαίρεση. Για να μπορέσουμε εμείς να δημιουργήσουμε μια εξαίρεση χειροκίνητα θα πρέπει να χρησιμοποιήσουμε την λέξη κλειδί throw. Οποιαδήποτε μέθοδος η οποία δημιουργεί μια εξαίρεση θα πρέπει να περιγράφεται από την λέξη κλειδί throws και τον τύπο της εξαίρεσης. Η παρακάτω μέθοδος αποτελεί ένα παράδειγμα μεθόδου η οποία δημιουργεί μια εξαίρεση.

```
1 public void mymethod(int index, int []array)throws ArrayIndexOutOfBoundsException{
2       // Κάποιες ενέργειες
3       if(index>=0 && index<array.length){
4               // Κάποιες ενέργειες
5       }       //διαφορετικά δημιούργησε μια τέτοιου είδους εξαίρεση
6       throw new ArrayIndexOutOfBoundsException();
7 }
```

6. Η βιβλιοθήκη της Java (Java API)

Η εταιρία Sun για να διευκολύνει τους χρήστες της γλώσσας έχει δημιουργήσει την βιβλιοθήκη της Java η οποία περιέχει πολλές έτοιμες κλάσεις οργανωμένες σε πακέτα. Για να μπορέσουμε να χρησιμοποιήσουμε μια κλάση της βιβλιοθήκης θα πρέπει να γνωρίζουμε σε ποιο πακέτο βρίσκεται (υποενότητα 4.1.1). Στην εικόνα που ακολουθεί μπορούμε να δούμε τη βιβλιοθήκη της Java όπως περιγράφεται στην επίσημη ιστοσελίδα της εταιρίας (http://java.sun.com/javase/6/docs/api/).



Στο πάνω αριστερό πλαίσιο παρατηρούμε τα ονόματα των πακέτων. Επιλέγοντας κάποιο πακέτο θα εμφανιστούν οι κλάσεις που περιέχει στο πλαίσιο ακριβώς από κάτω. Αν στη συνέχεια επιλέξουμε μια κλάση θα μας εμφανίσει στο κεντρικό πλαίσιο του παραθύρου όλα τα μέλη της μαζί με τη περιγραφή τους. Έτσι η αναζήτηση των κατάλληλων κλάσεων γίνετε αρκετά εύκολη. Παρακάτω φαίνεται ένα παράδειγμα χρήσης της κλάσης *ArrayList,* μιας λίστας που έχει υλοποιηθεί χρησιμοποιώντας πίνακα και μερικών μεθόδων της.

```
1 import java.util.ArrayList; // Εισαγωγή ArrayList από το πακέτο java.util
2 import javax.swing.JOptionPane;
3 public abstract class ArrayListExample {
4
5       public static void main(String[] args){
6               Truck truck = new Truck("Volvo", 1000);
7               Car car = new Car("Toyota");
8               String name = JOptionPane.showInputDialog("Give the owner");
9               Vehicle vehicle1 = new Vehicle("Yugo");
10              vehicle1.setOwner(name);
11              Vehicle vehicle2 = new Vehicle();
12              // Κλάση του API
13              ArrayList list = new ArrayList();
14              // Εισάγουμε τα αντικείμενα
15              list.add(truck);
16              list.add(car);
17              list.add(vehicle1);
18              // Το μέγεθος της λίστας
19              list.size();   // Επιστρέφει τον αριθμό των στοιχείων της λίστας (3)
20              list.contains(truck);          //true, περιέχει το truck
21              list.contains(vehicle2);       //false, δεν περιέχει το vehicle2
22              list.indexOf(car);     // Η θέση που βρίσκεται το car (2)
```

```
23              list.get(0);   // Επιστρέφει το truck
24
25              for(int i = 0; i < list.size(); i++) {
26                      Vehicle temp = (Vehicle)list.get(i);
27                      System.out.println(temp.info() + temp.getOwner());
28              }
29
30              list.clear();  // Καθαρίζουμε τη λίστα
31      }
32 }
```

Στη βιβλιοθήκη της Java παρέχεται και ένα σύνολο κλάσεων για την δημιουργία γραφικών διεπαφών, στις οποίες θα αναφερθούμε αμέσως μετά. Στη γραμμή 8 χρησιμοποιούμε την κλάση JOptionPane για την είσοδο δεδομένων στο προγραμμα.

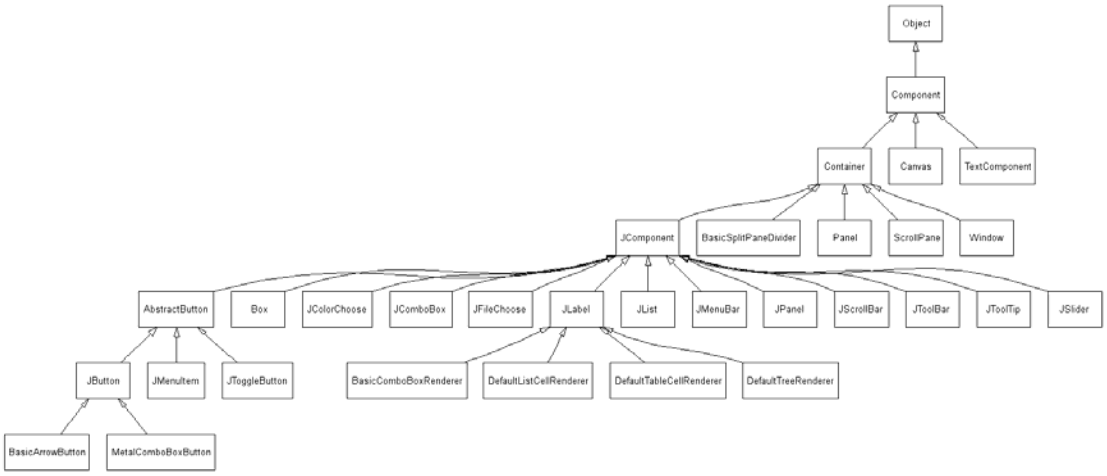
7. Γραφικές Διεπαφές Χρήστη – Swing

Ένα από τα πιο σημαντικά μέρη μιας εφαρμογής είναι εκείνο που είναι υπεύθυνο για την επικοινωνία με το χρήστη. Η επικοινωνία με το χρήστη μπορεί να γίνει είτε μέσω της γραμμής εντολών (κονσόλας), ή μέσω μιας γραφικής διεπαφής χρήστη (GUI - Graphical User Interface). Η δεύτερη κάνει ευκολότερη και πιο ελκυστική την είσοδο των δεδομένων και την εμφάνιση της εξόδου στο χρήστη. Η βιβλιοθήκη της Java παρέχει ένα σύνολο κλάσεων για την υλοποίηση και διαχείριση των γραφικών καθώς και για την υποστήριξη του πληκτρολογίου, του ποντικιού και άλλων συσκευών εισόδου.

Το Swing είναι ένα πακέτο της βιβλιοθήκης της Java το οποίο παρέχει τις κλάσεις για την δημιουργία αλληλεπιδραστικών γραφικών περιβαλλόντων. Τα GUIs δημιουργούνται συνδυάζοντας διάφορα επιμέρους συστατικά όπως:

- Πλαίσια: Είναι παράθυρα με τίτλο, μενού, κουμπιά ελαχιστοποίησης και μεγιστοποίησης.
- Υποδοχείς: Στοιχεία τα οποία μπορεί να περιέχουν και άλλα συστατικά.
- Κουμπιά: Περιοχές οι οποίες είναι συσχετισμένες με μία ενέργεια και τις οποίες ο χρήστης μπορεί να εκτελέσει κάνοντας κλικ με το ποντίκι.
- Πεδία κειμένου: Περιοχές στις οποίες ο χρήστης μπορεί να εισάγει κείμενο.

Το επόμενο διάγραμμα απεικονίζει κάποιες μόνο από τις κλάσεις που μας δίνει το Swing.



7.1 Η κλάση Component

Η κλάση Component (java.awt.Component) είναι μία αφηρημένη κλάση που περιέχει ένα σύνολο βασικών μεθόδων για την διαχείριση των διαφόρων συστατικών ενός GUI. Τα συνήθη συστατικά που χρησιμοποιούνται αποτελούν υποκλάσεις της Component και επομένως κληρονομούν τις μεθόδους της. Μερικές από τις σημαντικότερες είναι:

- ***setLocation****(int x,int y)* : void
  Η μέθοδος αυτή ορίζει την θέση ενός συστατικού. Οι τιμές *x* και *y* αποτελούν συντεταγμένες μετρούμενες σε pixels, ενώ ως τοποθεσία (0,0) ορίζεται η πάνω αριστερή γωνία της οθόνης.
- ***setSize****(int width,int height)* : void

Ορίζει το μέγεθος του συστατικού σε pixels.

- ***setBounds****(int x,int y,int width,int height)* : void
  Συνδυάζει τις μεθόδους ***setLocation*** και ***setBounds***. Προσδιορίζει τόσο την τοποθεσία του συστατικού (παράμετροι *x και y*) όσο και το μέγεθος του (παράμετροι *width* και *height*) σε pixels.
- ***setFont****(Font f)* : void
  Προσδιορίζει ποια θα είναι η μορφή της γραμματοσειράς που θα περιέχει το συστατικό.
- ***setBackground****(Color c)* : void
  Ορίζει ποιο θα είναι το χρώμα του παρασκηνίου του συστατικού. Το χρώμα επιλέγεται με χρήση της κλάσης Color (java.awt.Color). Εάν η συγκεκριμένη παράμετρος είναι null τότε ως χρώμα επιλέγεται το χρώμα του πατρικού συστατικού , εφόσον υπάρχει.
- ***setVisible****(boolean isVisible)* : void
  Καθορίζει εάν το συγκεκριμένο συστατικό θα είναι ορατό ή αόρατο, ανάλογα με την τιμή της isVisible.
- ***addKeyListener****(KeyListener kl)* : void
  Προσθέτει ένα ακροατή για συμβάντα πληκτρολογίου.
- ***addActionListener****(ActionListener al)* : void
  Προσθέτει ένα ακροατή για συμβάντα ενέργειας.

Τα πακέτα της βιβλιοθήκης της Java που χρησιμοποιούνται για τον προγραμματισμό με γραφικές διεπαφές είναι τα **java.awt**, **javax.swing** και **java.awt.event**. Περισσότερες αναφορές μπορούν να βρεθούν στη βιβλιοθήκη της Java. (http://java.sun.com/javase/6/docs/api/)

7.2 Υποδοχείς - Πλαίσια

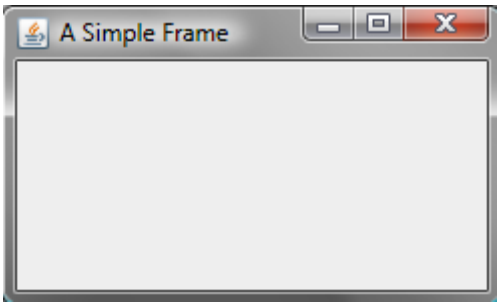
Ένας υποδοχέας είναι ένα αντικείμενο της κλάσης Container (java.awt.Container). Επομένως είναι επίσης αντικείμενο της κλάσης Component αφού η τελευταία είναι υπερκλάση της κλάσης Container. Έτσι λοιπόν γίνεται κατανοητό ότι οι υποδοχείς είναι και αυτοί με την σειρά τους συστατικά. Αυτό μας παρέχει την δυνατότητα να προσθέσουμε άλλα συστατικά σε ένα υποδοχέα ή άλλους υποδοχείς, δημιουργώντας με αυτό τον τρόπο σύνθετα GUI. Επιπρόσθετα κάθε γραφική διεπαφή του Swing πρέπει να περιέχει τουλάχιστον ένα βασικό υποδοχέα.

Ένας από τους βασικούς υποδοχείς είναι τα πλαίσια. Τα πλαίσια είναι παράθυρα τα οποία εμφανίζονται στην επιφάνεια εργασίας του χρήστη και αποτελούνται από τίτλο, γραμμή μενού, κουμπιά αλλαγής μεγέθους και κλεισίματος. Επιπλέον έχουν δυνατότητα αλλαγής μεγέθους και μετακίνησης. Μιας και τα πλαίσια είναι υποδοχείς μπορούν να περιέχουν μέσα και άλλα συστατικά. Η κλάση JFrame (javax.swing.JFrame) είναι η συνηθέστερη για την κατασκευή τέτοιων πλαισίων.

Το παρακάτω πρόγραμμα δημιουργεί ένα απλό πλαίσιο.

```
1 import javax.swing.*;
2 public class SimpleFrame extends JFrame{
3       public static void main(String[] args){
4               JFrame frame=new JFrame("A Simple Frame");
5               frame.setBounds(300,300,200,100);
6               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7               frame.setVisible(true);
8       }
```

```
9 }
```



Στη γραμμή 3 του παραπάνω προγράμματος δημιουργούμε ένα καινούργιο πλαίσιο (**JFrame**). Το όνομα που δίνουμε στην κατασκευάστρια μέθοδο του πλαισίου αποτελεί ουσιαστικά τον τίτλο του παραθύρου όπως φαίνεται και στην εικόνα παραπάνω. Στην επόμενη γραμμή ορίζουμε που θα βρίσκεται το πλαίσιο και ποιο θα είναι το μέγεθος του. Συγκεκριμένα το τοποθετούμε στη θέση 300,300 (δηλαδή 300 pixels χαμηλότερα και δεξιότερα από την πάνω αριστερή γωνία της οθόνης του υπολογιστή) με μέγεθος 200x100 pixels. Στη γραμμή 5 προσδιορίζουμε τις ενέργειες που θα συμβούν εάν πατήσουμε το κουμπί κλεισίματος του πλαισίου. Τέλος στην τελευταία γραμμή κάνουμε το πλαίσιο ορατό.

7.3 Συστατικά

Συστατικά είναι τα μέρη από τα οποία δομείται ένα GUI. Ένα σύνολο συστατικών τοποθετείται μέσα σε έναν υποδοχέα και κατά αυτό τον τρόπο συντίθεται μία ολοκληρωμένη γραφική διεπαφή. Όλα τα συστατικά που θα περιγράψουμε στη συνέχεια αποτελούν αντικείμενα υποκλάσεων της κλάσης Component (βλ. 7.1) και επομένως κληρονομούν το σύνολο των μεθόδων αυτής.

Η κλάση **JButton** (javax.swing.JButton) χρησιμοποιείται για την δημιουργία κουμπιών τα οποία ο χρήστης μπορεί να ενεργοποιήσει με το ποντίκι. Ο χρήστης μπορεί να συσχετίσει τις ενέργειες που θα εκτελούνται όταν πατιέται ένα κουμπί, χρησιμοποιώντας ακροατές συμβάντων (βλέπε 7.6). Επιπρόσθετα μπορεί να τοποθετήσει κείμενο η εικόνα μέσα σε ένα κουμπί ή συνδυασμό των δύο. Για την δημιουργία ενός αντικειμένου **JButton** προσφέρονται οι παρακάτω κατασκευάστριες μέθοδοι:

- ***JButton****()*
- ***JButton****(String text)*
- ***JButton****(Icon ic)*
- ***JButton****(String text,Icon ic)*

Η κλάση **JLabel** (javax.swing.JLabel) χρησιμοποιείται για την δημιουργία ετικετών που μπορούν να περιέχουν κείμενο, εικόνα ή συνδυασμό των δύο και παρέχει ένα πλήθος μεθόδων σχετικά με την στοίχιση του κειμένου ή της εικόνας που θα περιέχει. Οι ετικέτες δεν μπορούν να υποστούν επεξεργασία από το χρήστη του προγράμματος, παρά μόνο από τον προγραμματιστή της εφαρμογής. Για την δημιουργία μιας τέτοιας ετικέτας παρέχονται από την κλάση **JLabel** οι παρακάτω κατασκευάστριες μεθοδοι:

- ***JLabel****()*
- ***JLabel****(String text)*

- ***JLabel****(Icon ic)*
- ***JLabel****(String text,Icon ic)*

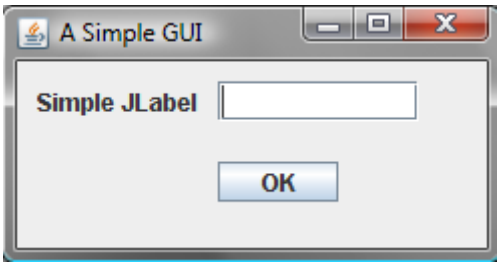
Η κλάση **JTextField** (javax.swing.JTextField) χρησιμοποιείται για την δημιουργία μιας περιοχής κειμένου, ενός επεξεργάσιμου δηλαδή πεδίου κειμένου. Ο χρήστης μπορεί να εισάγει σε αυτό την είσοδο και εφόσον έχει αναθέσει στο συστατικό αυτό ένα ακροατή συμβάντων όταν πατήσει το Enter θα εκτελεστεί το συγκεκριμένο γεγονός. Παρόμοια είναι και η κλάση **JTextArea** (javax.swing.JTextArea) μόνο που χρησιμοποιείται για την εισαγωγή μεγαλύτερων κειμένων. Για την δημιουργία JTextField παρέχονται οι παρακάτω κατασκευάστριες μέθοδοι:

- ***JTextField****()*
- ***JTextField****(String text)*
- ***JTextField****(int numOfColumns)*
- ***JTextField****(String text, int numOfColumns)*

Παρακάτω φαίνεται ένα παράδειγμα ενός πλαισίου με τρία συστατικά:

```
1 import javax.swing.*;
2 public class SimpleGui extends JFrame{
3       public static void main(String[] args){
4               JFrame frame=new JFrame("A Simple GUI");
5               frame.setSize(250,130);
6               frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7               frame.setVisible(true);
8               frame.setLayout(null);
9               JLabel label=new JLabel("Simple JLabel");
10              JTextField text=new JTextField();
11              JButton button=new JButton("OK");
12              label.setBounds(10,10,100,20);
13              text.setBounds(100, 10, 100, 20);
14              button.setBounds(100,50,60,20);
15              frame.add(label);
16              frame.add(text);
17              frame.add(button);
18      }
19}
```

Το παραπάνω πρόγραμμα δημιουργεί την γραφική διεπαφή που φαίνεται παρακάτω.



Στις γραμμές 3 έως 7 δημιουργούμε και τροποποιούμε το πλαίσιο. Συγκεκριμένα δημιουργούμε ένα πλαίσιο (αντικείμενο της κλάσης **JFrame**). Στη γραμμή 4 ορίζουμε ποιο θα είναι το μέγεθος του παραθύρου ενώ στην επόμενη ποια θα είναι η ενέργεια του προγράμματος όταν πατήσουμε το κουμπί κλεισίματος. Στη γραμμή 6 κάνουμε ορατό το πλαίσιο. Μπορούμε να αγνοήσουμε προσωρινά την γραμμή 7 μιας και πρόκειται για διαχειριστή διάταξης για τον οποίο θα μιλήσουμε στη συνέχεια. Κατόπιν φτιαχνουμε 3 συστατικά: μία ετικέτα (**JLabel**), μία περιοχή κειμένου (**JTextFiled**) και ένα κουμπί (**JButton**). Στις 3 τελευταίες γραμμές τοποθετούμε τα 3

συστατικά στον υποδοχέα που έχουμε ορίσει. Αξίζει να σημειώσουμε ότι και τα 3 συστατικά τοποθετούνται σε διαφορετικές θέσεις του ίδιου υποδοχέα.

7.4 Διάταξη Συστατικών

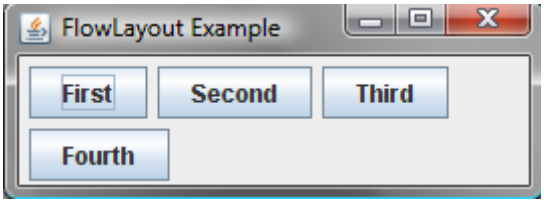
Μέχρι τώρα είδαμε πώς να δημιουργούμε διάφορα συστατικά και να τα συνδυάζουμε για την δημιουργία πιο σύνθετων γραφικών διεπαφών. Ένα σημαντικό κομμάτι προς αυτή την κατεύθυνση είναι η διάταξη των συστατικών μέσα σε ένα υποδοχέα. Ένας διαχειριστής διάταξης μας επιτρέπει να τοποθετήσουμε εύκολα τα συστατικά χωρίς να ενδιαφερόμαστε για τις επιμέρους πολύπλοκες λεπτομέρειες (λ.χ. μετρώντας τα pixel χρησιμοποιώντας την μέθοδο *setLocation()*). Επιπρόσθετα πολλοί διαφορετικοί διαχειριστές διάταξης μπορούν να τοποθετηθούν μαζί στην ίδια διασύνδεση.

Η **διάταξη ροής** είναι ο απλούστερος και συνάμα ο προεπιλεγμένος διαχειριστής διάταξης. Για αυτό το είδος διάταξης των συστατικών χρησιμοποιείται η κλάση **FlowLayout** (java.awt.FlowLayout), η οποία διατάσσει τα συστατικά από αριστερά προς τα δεξιά, μέχρι να μην υπάρχει χώρος για κάποιο συστατικό, οπότε συνεχίζει σε καινούργια γραμμή. Η σειρά διάταξης έχει άμεση σχέση με την σειρά με την οποία εισάγονται τα συστατικά στον υποδοχέα καθώς το πρώτο συστατικό που εισάγεται θα τοποθετηθεί τέρμα αριστερά, το δεύτερο ακριβώς δεξιότερα του πρώτου κ.ο.κ. Συνολικά ο προγραμματιστής μπορεί να επιλέξει το είδος της στοίχισης που επιθυμεί επιλέγοντας την αντίστοιχη μεταβλητή κατά την δημιουργία του διαχειριστή διάταξης. Η κατασκευάστρια μέθοδος της κλάσης **FlowLayout** παίρνει σαν όρισμα μία μεταβλητή που ορίζει το είδος της στοίχισης. Επιπρόσθετα υπάρχει άλλη μία κατασκευάστρια μέθοδος που δέχεται 2 επιπλέον ορίσματα τα οποία αναφέρονται στο οριζόντιο και το κάθετο διάκενο που θα υπάρχει ανάμεσα στα συστατικά (σε pixels). Συνολικά υπάρχουν 3 διαφορετικοί τρόποι στοίχισης:

- Αριστερά (με χρήση μεταβλητής *FlowLayout.Left*)
- Δεξιά (με χρήση μεταβλητής *FlowLayout.Right*)
- Κέντρο (με χρήση μεταβλητής *FlowLayout.Center*)

Ένα παράδειγμα χρήσης του παραπάνω διαχειριστή διάταξης φαίνεται στο παρακάτω πρόγραμμα.

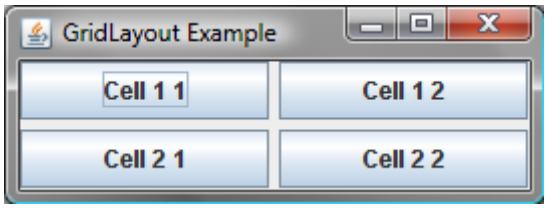
```
1 import javax.swing.*;
2 import java.awt.FlowLayout;
3 public class FlowExample extends JFrame{
4       public static void main(String[] args){
5               JFrame frame=new JFrame("FlowLayout Example");
6               JButton b1=new JButton("First");
7               JButton b2=new JButton("Second");
8               JButton b3=new JButton("Third");
9               JButton b4=new JButton("Fourth");
10              frame.setBounds(300,300,270,100);
11              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12              FlowLayout flow=new FlowLayout(FlowLayout.LEFT);
13              frame.setLayout(flow);
14              frame.add(b1);
15              frame.add(b2);
16              frame.add(b3);
17              frame.add(b4);
18              frame.setVisible(true);
19      }
20 }
```



Ο **διαχειριστής διάταξης πλέγματος** διατάσσει τα στοιχεία σε ένα πλέγμα γραμμών και στηλών. Το πρώτο συστατικό που εισάγεται στον υποδοχέα τοποθετείται στην πρώτη γραμμή και στην πρώτη στήλη, το επόμενο στην επόμενη στήλη της ίδιας γραμμής κ.ο.κ. Αυτό γίνεται μέχρι να γεμίσουν όλα τα κελιά μιας γραμμής οπότε και ξεκινάει η συμπλήρωση της επόμενης γραμμής. Τέτοιου είδους διατάξεις δημιουργούνται με την κλάση **GridLayout** (java.awt.GridLayout). Η κατασκευάστρια μέθοδος αυτής της κλάσης παίρνει σαν ορίσματα τον αριθμό των γραμμών και των στηλών του πλέγματος. Όμοια με την κλάση FlowLayout έχει μία επιπλέον κατασκευάστρια μέθοδο η οποία εκτός των προηγουμένων δέχεται σαν ορίσματα το οριζόντιο και το κάθετο διάκενο ανάμεσα στα συστατικά.

Ένα παράδειγμα χρήσης της παραπάνω διάταξης φαίνεται στο παρακάτω παράδειγμα:

```
1 import javax.swing.*;
2 import java.awt.GridLayout;
3 public class GridExample extends JFrame{
4       public static void main(String[] args){
5               JFrame frame=new JFrame("GridLayout Example");
6               JButton b11=new JButton("Cell 1 1");
7               JButton b12=new JButton("Cell 1 2");
8               JButton b21=new JButton("Cell 2 1");
9               JButton b22=new JButton("Cell 2 2");
10              frame.setBounds(300,300,270,100);
11              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12              GridLayout grid=new GridLayout(2,2,5,5);
13              frame.setLayout(grid);
14              frame.add(b11);
15              frame.add(b12);
16              frame.add(b21);
17              frame.add(b22);
18              frame.setVisible(true);
19      }
20 }
```

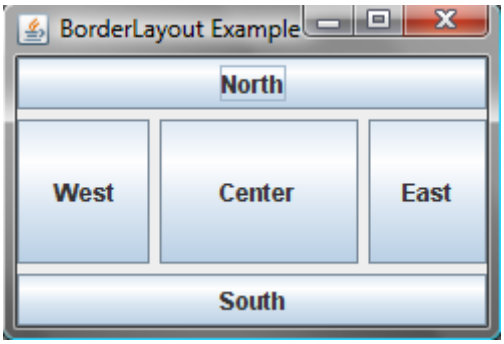


Η **διάταξη περιγράμματος** δημιουργείται με χρήση της κλάσης **BorderLayout** (java.awt.BorderLayout). Ο τρόπος με τον οποίο διατάσσει τα συστατικά είναι διαιρώντας τον υποδοχέα σε 5 διαφορετικούς τομείς: βόρειο, νότιο, ανατολικό, δυτικό και κεντρικό. Κάθε συστατικό τοποθετείται στον αντίστοιχο τομέα προσθέτοντας μια επιπλέον παράμετρο στη μέθοδο *add()* του υποδοχέα η οποία προσδιορίζει τον αντίστοιχο τομέα. Οι τιμές που μπορεί να πάρει αυτή η επιπλέον παράμετρος είναι: ("North", "South", "East", "West", "Center"). Αυτός ο διαχειριστής διάταξης περιλαμβάνει δύο κατασκευάστριες μεθόδους, η πρώτη δεν

περιλαμβάνει καθόλου ορίσματα και απλά δημιουργεί την διάταξη, ενώ η δεύτερη περιλαμβάνει παραμέτρους για το οριζόντιο και το κάθετο διάκενο.

Ένα παράδειγμα χρήσης της παραπάνω διάταξης φαίνεται στο παρακάτω πρόγραμμα:

```
1 import javax.swing.*;
2 import java.awt.BorderLayout;
3 import java.awt.event.ActionListener;
4 import java.awt.event.ActionEvent;
5 public class BorderExample extends JFrame{
6       public static void main(String[] args){
7               JFrame frame=new JFrame("BorderLayout Example");
8               JButton bNorth=new JButton("North");
9               JButton bSouth=new JButton("South");
10              JButton bWest=new JButton("West");
11              JButton bEast=new JButton("East");
12              JButton bCenter=new JButton("Center");
13              frame.setBounds(300,300,250,170);
14              frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15              BorderLayout border=new BorderLayout(5,5);
16              frame.setLayout(border);
17              frame.add("North",bNorth);
18              frame.add("South",bSouth);
19              frame.add("West",bWest);
20              frame.add("East",bEast);
21              frame.add("Center",bCenter);
22              frame.setVisible(true);
23      }
24 }
```



7.5 Ταμπλό (Τομείς)

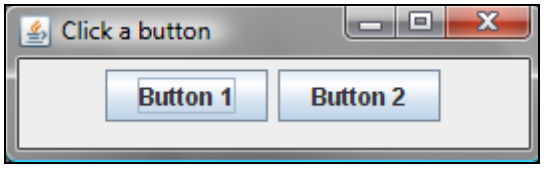
Κάθε GUI αποτελείται από τουλάχιστον έναν υποδοχέα στον οποίο τοποθετούνται όλα τα συστατικά όπως παρατηρήσαμε παραπάνω με την κλάση **JFrame.** Ωστόσο όταν η διεπαφή αποτελείται από πολλά διαφορετικά συστατικά, κάθε ένα από τα οποία πρέπει να βρίσκεται σε συγκεκριμένη θέση είναι συχνά βολικό να χρησιμοποιούμε τομείς. Οι τομείς είναι υποδοχείς στους οποίους τοποθετούνται τα συστατικά. Η κλάση **JPanel** (javax.swing.JPanel) είναι ένα τέτοιο είδος υποδοχέα. Για την δημιουργία ενός αντικειμένου τομέα χρειάζεται μία κλήση της κατασκευάστριας μεθόδου *JPanel()*. Κατόπιν για να εισάγουμε ένα συστατικό στον υποδοχέα χρησιμοποιούμε την μέθοδο *add()*, μιας και η μέθοδος αυτή κληρονομείται από την υπερκλάση **Component**. Κατόπιν τοποθετούμε τους διάφορους υποδοχείς όπως ακριβώς τοποθετούσαμε τα συστατικά.

7.6 Ακροατές Συμβάντων

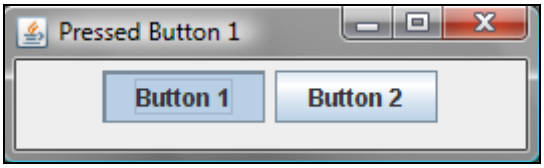
Προηγουμένως είδαμε ότι ένα GUI μπορεί να περιέχει κουμπιά, περιοχές εισαγωγής κειμένου καθώς και άλλα συστατικά με τα οποία ο χρήστης μπορεί να εισάγει

δεδομένα σε ένα πρόγραμμα. Για να μπορεί το πρόγραμμα να αξιοποιήσει αυτά τα δεδομένα θα πρέπει αρχικά να μπορεί να καταλάβει πότε έχει συμβεί ένα γεγονός, πότε για παράδειγμα έχει πατηθεί ένα κουμπί. Αυτό η Java το επιτυγχάνει με τους ακροατές συμβάντων. Κάθε ακροατής μπορεί να χειρίζεται ένα συγκεκριμένο είδος συμβάντος ενώ ένα πρόγραμμα μπορεί να περιέχει όσους ακροατές χρειάζεται. Όταν για παράδειγμα θέλουμε να κάνουμε κάποιες ενέργειες ανάλογα με το αν ο χρήστης πατήσει ένα κουμπί του GUI, δεν έχουμε παρά να προσθέσουμε ένα ακροατή συμβάντος στο κουμπί. Κατόπιν μπορούμε με την βοήθεια των μεθόδων που μας παρέχει ο ακροατής να πράξουμε τις ενέργειες που επιθυμούμε. Παρακάτω υπάρχει ένα παράδειγμα χρήσης των του ακροατή για συμβάντα ενέργειας.

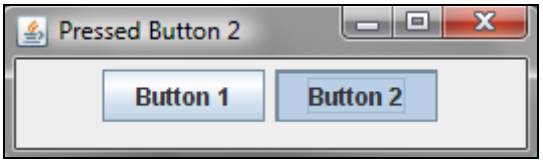
```
1 import javax.swing.*;
2 import java.awt.FlowLayout;
3 import java.awt.event.ActionListener;
4 import java.awt.event.ActionEvent;
5  public class ActionExample extends JFrame implements ActionListener{
6       JButton b1=new JButton("Button 1");
7       JButton b2=new JButton("Button 2");
8
9       public ActionExample(){
10              super("Click a button");
11              setBounds(300,300,270,80);
12              setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13              FlowLayout flow=new FlowLayout(FlowLayout.CENTER);
14              setLayout(flow);
15              b1.addActionListener(this);
16              b2.addActionListener(this);
17              add(b1);
18              add(b2);
19              setVisible(true);
20      }
21
22      public void actionPerformed(ActionEvent event){
23              Object buttonPressed=event.getSource();
24              if(buttonPressed==b1)
25                      setTitle("Pressed Button 1");
26              else if(buttonPressed==b2)
27                      setTitle("Pressed Button 2");
28      }
29
30      public static void main(String[] args){
31              ActionExample example=new ActionExample();
32      }
33}
```



Εάν πατήσουμε το κουμπί Button 1



Παρόμοια αν πατήσουμε το Button 2



7.7 Μικροεφαρμογές της Java, JApplet

Μια μικροεφαρμογή είναι μία Java εφαρμογή η οποία εκτελείται μέσα σε ένα περιηγητή του ιστού (Firefox, Internet Explorer, Opera). Ουσιαστικά λοιπόν διαφέρουν από τις εφαρμογές όχι στον τρόπο με τον οποίο δημιουργούνται ούτε στα συστατικά τα οποία περιέχουν παρά μόνο στον τρόπο με τον οποίο εκτελούνται. Για να εκτελεστεί μία μικροεφαρμογή αρκεί να «τοποθετηθεί» (συγκεκριμένα, να αναφερθεί) σε μία html σελίδα χρησιμοποιώντας τα κατάλληλα tags.

Όταν ένας χρήστης με ένα πρόγραμμα περιήγησης φορτώσει μία ιστοσελίδα που περιέχει μία μικροεφαρμογή, το πρόγραμμα περιήγησης φορτώνει την μικροεφαρμογή από κάποιον Web Server και την εκτελεί στον υπολογιστή του χρήστη. Απαραίτητη προϋπόθεση είναι ο περιηγητής σελίδων να έχει ενσωματωμένο το διερμηνευτή της Java.

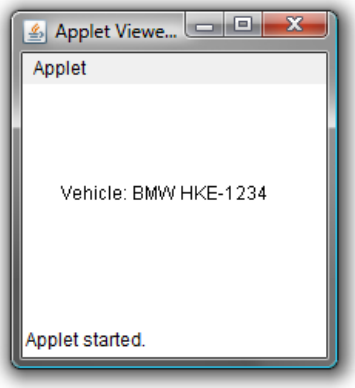
Εφόσον η διαφορά μεταξύ των εφαρμογών και των μικροεφαρμογών είναι στην εκτέλεση τους αυτό σημαίνει ότι θα πρέπει μία εφαρμογή να λειτουργεί ως μικροεφαρμογή και το αντίθετο κάνοντας ελάχιστες τροποποιήσεις. Πράγματι το σημείο εκκίνησης μιας εφαρμογής είναι η μέθοδος main ενώ αντίθετα στις μικροεφαρμογές εκτελούνται μία σειρά από άλλες μεθόδους όπου η κάθε μία έχει συγκεκριμένες αρμοδιότητες. Οι μέθοδοι αυτές είναι :

- init(): Χρησιμοποιείται για την αρχικοποίηση της μικροεφαρμογής και καλείται όταν φορτώνεται η μικροεφαρμογή. Εκεί για παράδειγμα μπορούν να δημιουργηθούν τα διάφορα αντικείμενα που θα χρειαστεί να τοποθετηθούν στη μικροεφαρμογή.
- paint(Graphics g): Ορίζει το πώς η μικροεφαρμογή θα εμφανίζει κάτι στην οθόνη. Η εκτύπωση μπορεί να γίνει πολλές φορές κατά την διάρκεια της ζωής της μικροεφαρμογής, σε αντίθεση με την αρχικοποίηση όπου γίνεται μόνο κατά την εκκίνηση της μικροεφαρμογής.
- start(): Μια εφαρμογή πρέπει να εκκινήσει αφού κατόπιν έχει αρχικοποιηθεί είτε αφού προηγουμένως έχει διακοπεί.
- stop(): Διακόπτει την μικροεφαρμογή. Αυτό συμβαίνει όταν για παράδειγμα εγκαταλείπουμε μία σελίδα που περιέχει μία μικροεφαρμογή.
- destroy(): επιτρέπει στη μικροεφαρμογή να κάνει ότι τροποποιήσεις σε αντικείμενα χρειάζεται και κατόπιν να ελευθερώσει την μνήμη που είχε καταλάβει.

Οι παραπάνω μέθοδοι καλούνται όταν συμβαίνουν οι αντίστοιχες καταστάσεις στη μικροεφαρμογή. Ωστόσο είναι εφικτό να υποσκελιστούν και να κάνουν συγκεκριμένα πράγματα για κάθε ενέργεια. Παρακάτω φαίνεται μία απλή μικροεφαρμογή η οποία χρησιμοποιεί ένα αντικείμενο τύπου Vehicle και εμφανίζει τα στοιχεία του σε μία μικροεφαρμογή.

```
1 import java.awt.Graphics;
2 import javax.swing.JApplet;
3 public class VehicleApplet extends JApplet{
4       Vehicle v;
5       public void init(){            //Υποσκελισμός μεθόδου init
6               v=new Vehicle("BMW","HKE-1234");      //Δημιουργία αντικειμένου Vehicle
7       }
8       public void paint(Graphics g){   //Υποσκελισμός μεθόδου paint
9               g.drawString(v.info() ,25 , 75); //εμφάνιση στοιχείων στη θέση 25, 75
10      }
11}
```

Στον παραπάνω κώδικα υποσκελίζουμε τις μεθόδους *init()* και *paint()*. Συγκεκριμένα χρησιμοποιούμε την *init()* για την δημιουργία ενός αντικειμένου τύπου Vehicle. Κατόπιν στην *paint()* τοποθετούμε το αποτέλεσμα της κλήσης της μεθόδου *info()* σε συγκεκριμένη θέση στη μικροεφαρμογή. Η μικροεφαρμογή θα έχει ως εξής.

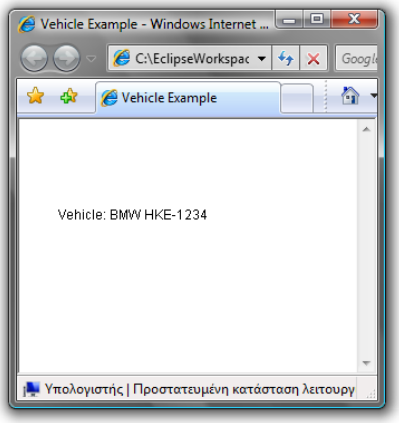


Για να δούμε το παραπάνω πρέπει να τρέξουμε την μικροεφαρμογή. Αυτό γίνεται τοποθετώντας την σε μία σελίδα html και είτε φορτώνοντας την σελίδα με κάποιο περιηγητή είτε δοκιμάζοντας το αρχείο με το appletviewer (παρέχεται από το JDK) πληκτρολογώντας την εντολή : '*appletviewer <ονομα html σελίδας>*'.

Για να τοποθετήσουμε σε μία σελίδα html μία μικροεφαρμογή αρκεί να τοποθετήσουμε την διαδρομή του .class αρχείου μέσα στα κατάλληλα tags.

```
<APPLET CODE= "ονομα_αρχείου.class" ></APPLET>
```

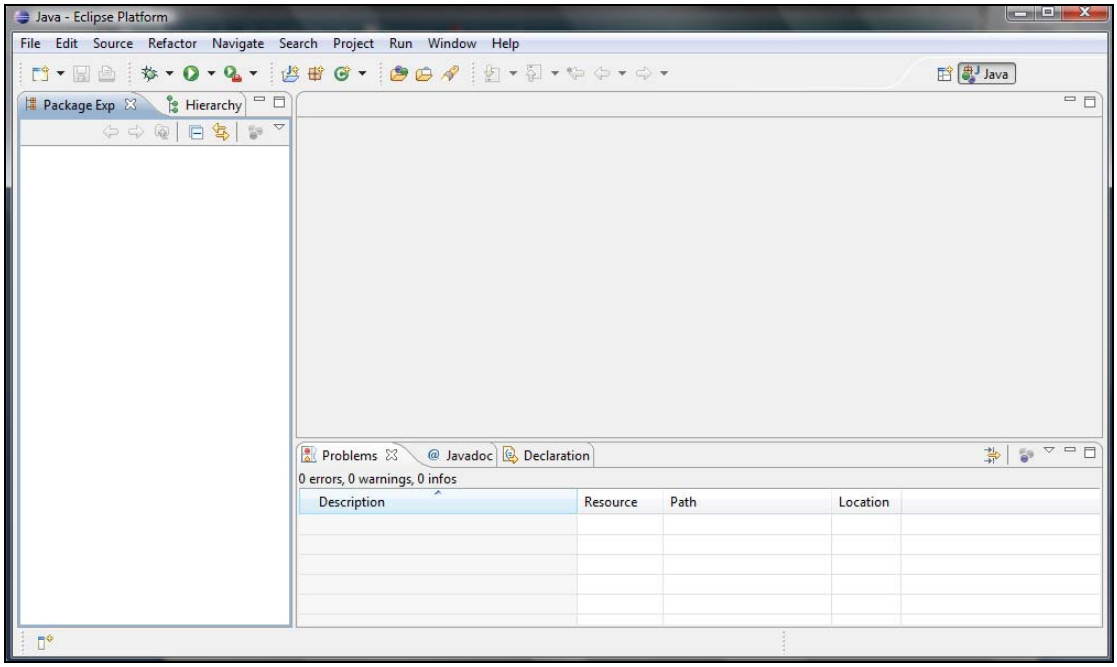
Η προβολή της παραπάνω μικροεφαρμογής μέσα από τον περιηγητή εμφανίζει το παρακάτω:



8. Eclipse

8.1 Περιβάλλον Eclipse

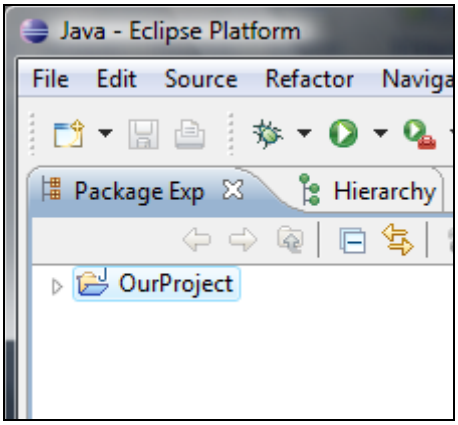
Το Eclipse είναι ένα ολοκληρωμένο περιβάλλον για την συγγραφή, μεταγλώττιση και εκτέλεση προγραμμάτων Java. Διανέμεται δωρεάν και είναι διαθέσιμο από την σελίδα http://www.eclipse.org/ η οποία περιέχει εκδόσεις για διάφορα λειτουργικά συστήματα. Ξεκινώντας στο Eclipse η πρώτη οθόνη που συναντάμε είναι η εξής:



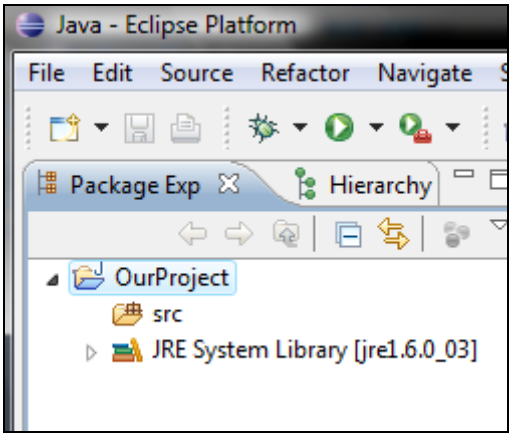
Στην περιοχή αριστερά τοποθετούνται τα προγράμματα μας. Η μεγάλη γκρίζα περιοχή στα δεξιά είναι η επιφάνεια εργασίας. Εκεί ουσιαστικά είναι η περιοχή στην οποία γράφουμε τον κώδικα των προγραμμάτων μας. Ακριβώς από κάτω υπάρχει μία κονσόλα πολλαπλών λειτουργιών.

8.2 Νέο έργο

Για να δημιουργήσουμε νέο έργο (project) κάνουμε κλικ διαδοχικά στα *File* → *New* → *Java Project* η πιο απλά κάνοντας κλικ στο εικονίδιο  που βρίσκεται πάνω αριστερά. Κατόπιν από το παράθυρο που ανοίγει επιλέγουμε ένα όνομα για το έργο και πατάμε το πλήκτρο *Finish*. Όπως θα γίνει και στη συνέχεια κατανοητό το όνομα του έργου που δημιουργούμε μπορεί να είναι τελείως διαφορετικό από το όνομα των κλάσεων του προγράμματος. Παρατηρούμε πλέον ότι στην περιοχή των εργασιών του **Eclipse** υπάρχει η εργασία που μόλις δημιουργήσαμε.



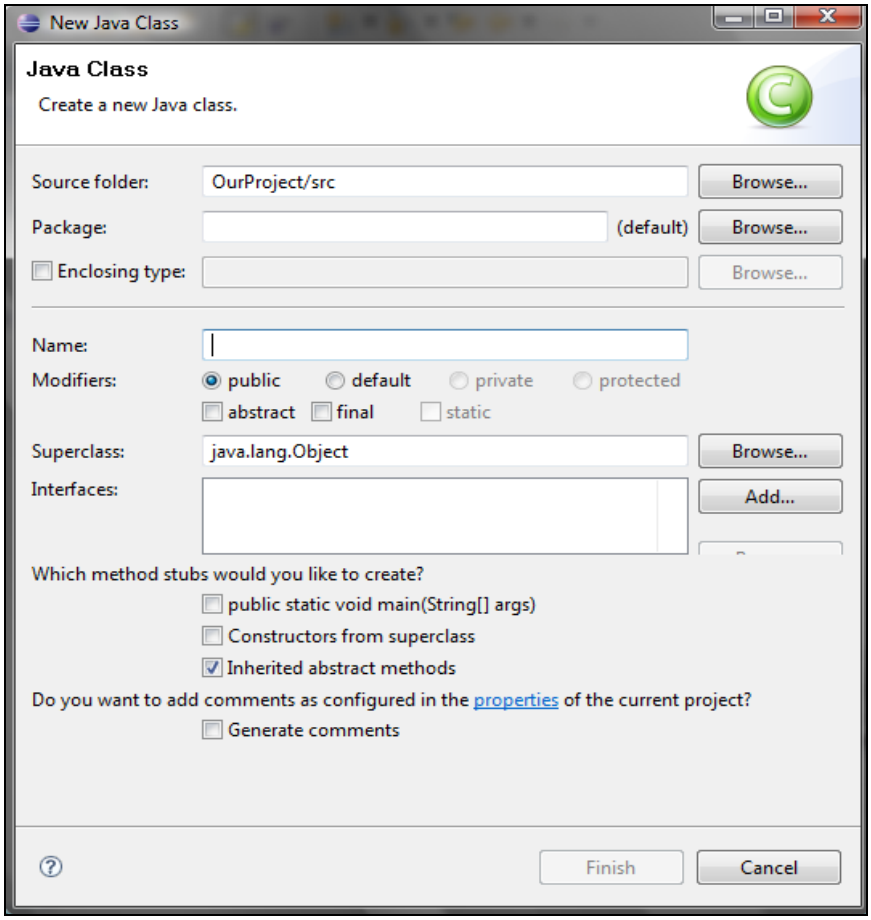
Κάνοντας κλικ στο βέλος δίπλα από το όνομα του έργου ανοίγει το μενού που μας δείχνει τα περιεχόμενα της εργασίας μας.



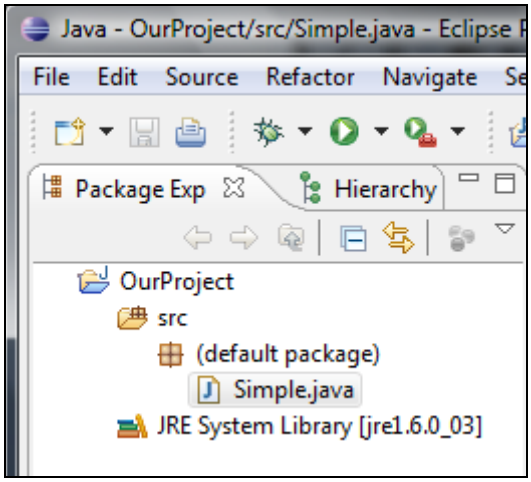
Το *src* περιέχει τον κώδικα του προγράμματος μας (το οποίο προς στιγμήν είναι άδειο) και από κάτω φαίνονται οι βιβλιοθήκες οι οποίες χρειάζονται για την μεταγλώττιση και εκτέλεση του προγράμματος.

8.3 Νέα κλάση

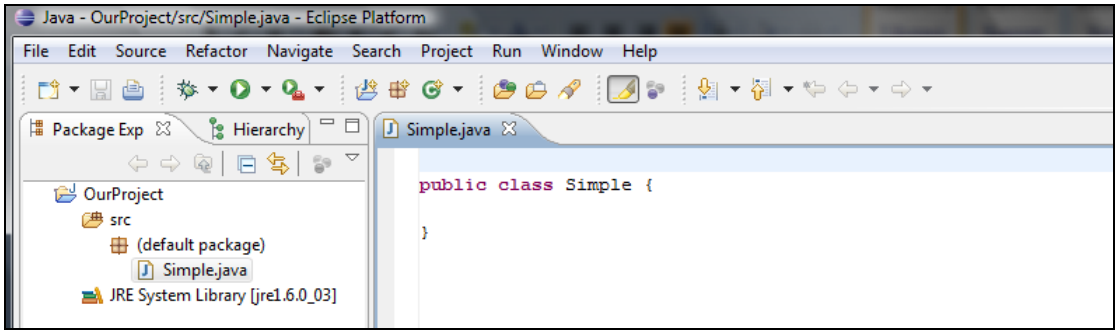
Για να δημιουργήσουμε μία νέα κλάση αρκεί να κάνουμε δεξί κλικ πάνω στην εργασία μας και να επιλέξουμε από το μενού *New* → *Class*.



Εκεί βάζουμε το όνομα που θέλουμε να έχει η κλάση και ένα πλήθος άλλων επιλογών, όπως εάν θέλουμε να την τοποθετήσουμε σε κάποιο πακέτο, εάν θέλουμε να υλοποιεί κάποια διεπαφή, κτλ. Αφού δημιουργήσουμε την κλάση πατάμε το *Finish*. Προσέξτε ότι η κλάση βρίσκεται στο παράθυρο των εργασιών μας.



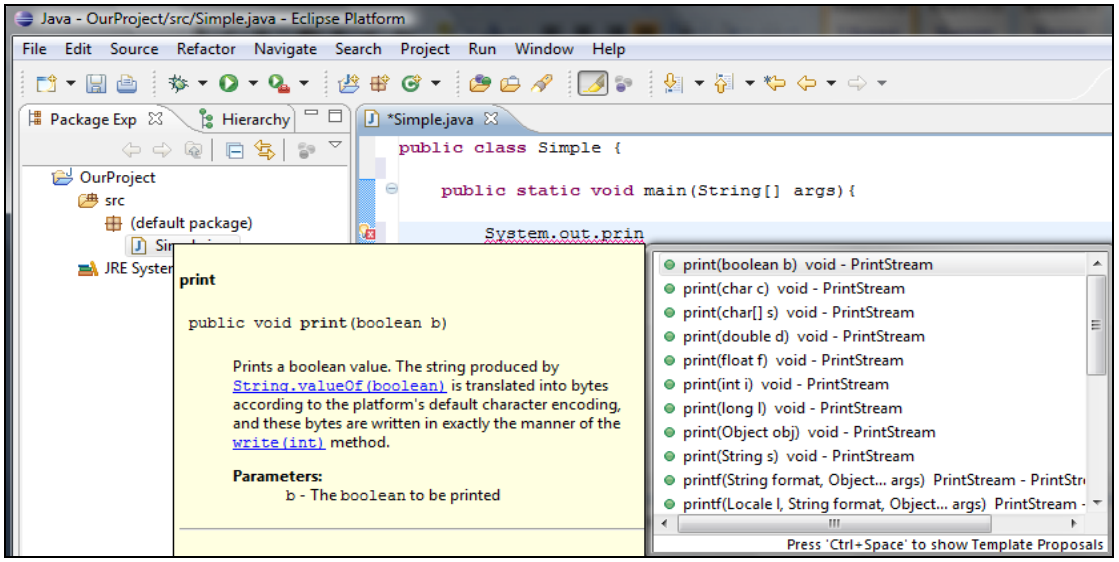
Παρατηρούμε ότι πλέον έχουμε μία κλάση Simple η οποία δεν βρίσκεται σε κάποιο πακέτο (η αλλιώς βρίσκεται στο προεπιλεγμένο πακέτο). Πλέον κάνοντας διπλό κλικ πάνω στην κλάση παρατηρούμε ότι στην περιοχή εργασίας μπορούμε πλέον να γράψουμε τον κώδικα για την κλάση.



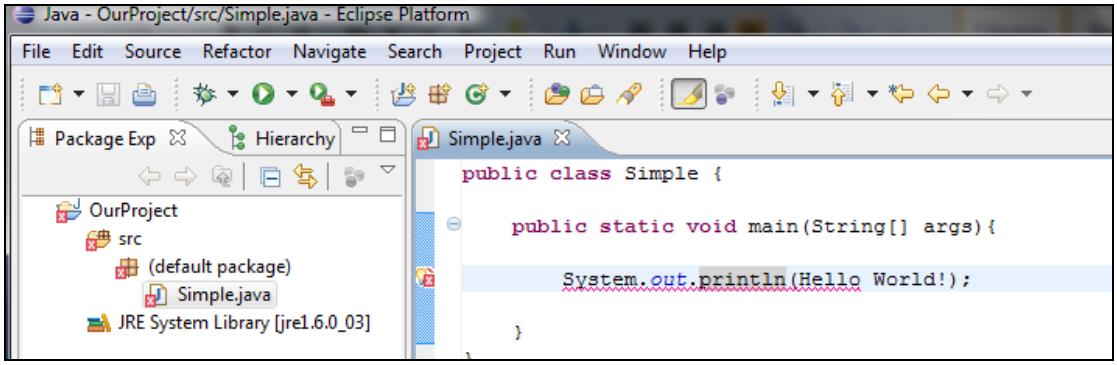
Είναι αξιοσημείωτο το γεγονός ότι το Eclipse χρησιμοποιεί διαφορετικά χρώματα για τις λέξεις κλειδιά (public, class, int, …) κάνοντας έτσι πιο ευανάγνωστο τον κώδικα.

8.4 Προγραμματίζοντας

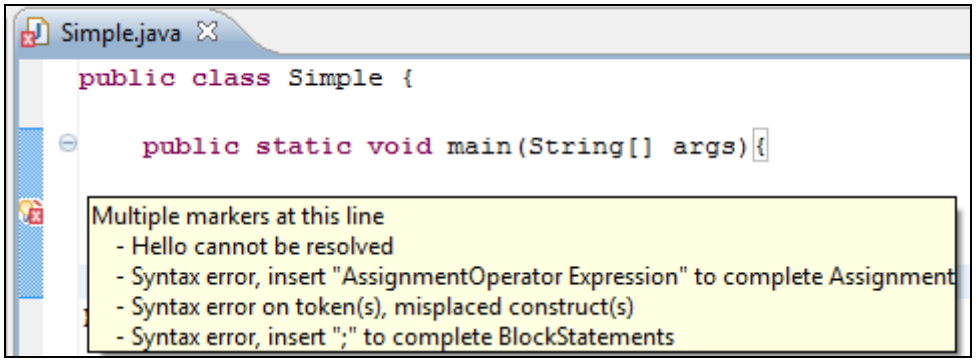
Ήρθε η ώρα που μπορούμε να αρχίσουμε να προγραμματίζουμε. Ας ξεκινήσουμε φτιάχνοντας μία main μέθοδο η οποία θα εκτυπώνει "Hello World". Καθώς γράφουμε θα παρατηρήσουμε κάποια παράθυρα (pop-ups) να εμφανίζονται. Αυτά δεν είναι τίποτε άλλο παρά παράθυρα που μας βοηθούν να γράψουμε κώδικα. Σε περίπτωση που θελήσουμε να εμφανίσουμε μόνοι μας, ένα τέτοιο βοηθητικό παράθυρο, το μόνο που έχουμε να κάνουμε είναι να πατήσουμε το συνδυασμό πλήκτρων ***ctrl+space***. Αν για παράδειγμα γράψουμε *System.out.prin* και πατήσουμε ***ctrl+space*** θα δούμε το εξής:



Το παράθυρο αυτό μας δείχνει όλες τις δυνατές επιλογές που ξεκινούν με "prin". Σε αυτό το σημείο μπορούμε είτε να πλοηγηθούμε ανάμεσα στις δυνατές επιλογές είτε να συνεχίσουμε να γράφουμε κώδικα. Με αυτό τον τρόπο το Eclipse μας βοηθάει να μειώσουμε τον αριθμό των συντακτικών λαθών. Ακόμα όμως και αν κάνουμε κάποιο λάθος, υπάρχει και πάλι ένα εύκολος τρόπος για να το βρούμε και να το διορθώσουμε. Έστω για παράδειγμα ότι γράφουμε System.out.println(Hello World!). Παρατηρούμε ότι η πρόταση που θέλουμε να εκτυπώσουμε δεν είναι μέσα σε " " οπότε έχουμε ένα συντακτικό λάθος. Αυτό που κάνει το Eclipse για τέτοιου είδους λάθη είναι το παρακάτω:



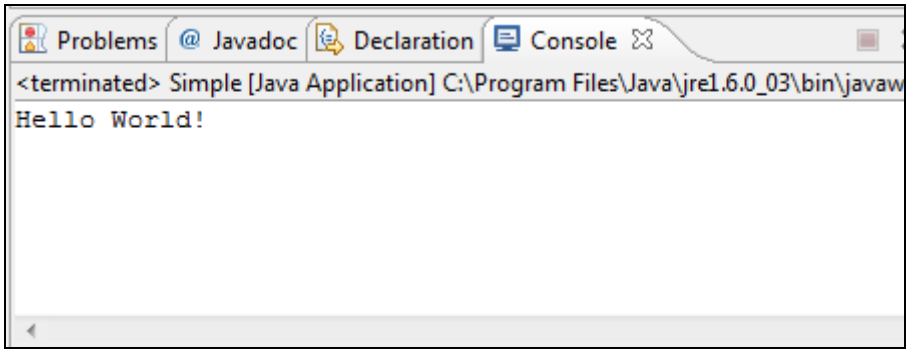
Παρατηρούμε τις κόκκινες ενδείξεις τόσο στην περιοχή των εργασιών όσο και στον κώδικα της κλάσης. Κοιτώντας στην περιοχή των εργασιών μπορούμε να καταλάβουμε σε ποια κλάση (εάν έχουμε πολλές) βρίσκεται το συντακτικό λάθος ενώ μετέπειτα κοιτώντας στον κώδικα μπορούμε να εντοπίσουμε το λάθος. Αν πάμε με τον δείκτη του ποντικιού πάνω στην ένδειξη στο σημείο που βρίσκεται το λάθος το Eclipse μας αναφέρει το συντακτικό λάθος που υπάρχει.



Όταν θελήσουμε να μεταγλωττίσουμε το πρόγραμμα δεν χρειάζεται να μπλεκόμαστε με πηγαία αρχεία '.java' και αρχεία μεταγλωττιστή. Πολύ απλά το Eclipse αυτό που κάνει είναι να μεταγλωττίζει την εργασία (δηλαδή όλες τις κλάσεις που αυτή μπορεί να περιέχει) αυτόματα. Το μόνο που χρειάζεται να κάνει ο χρήστης είναι να σώσει το έργο κάνοντας διαδοχικά *File* → *Save* ή πατώντας το κουμπί .

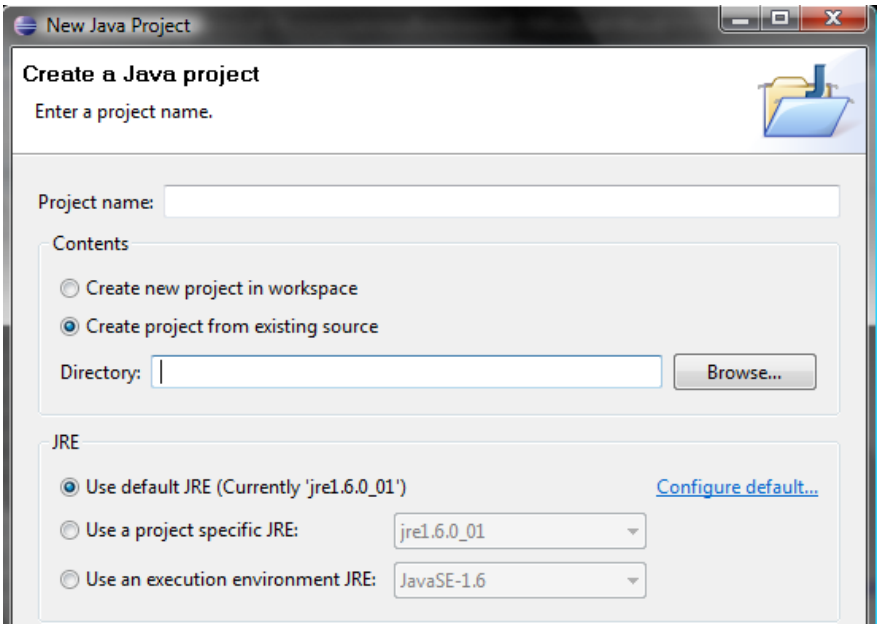
8.5 Εκτέλεση εφαρμογής

Για να εκτελέσουμε μία εφαρμογή, υπάρχουν 2 τρόποι. Είτε ανοίγοντας το αρχείο το οποίο περιέχει την *main* μέθοδο και πατώντας το κουμπί (*Run*), είτε επιλέγοντας το αρχείο που περιέχει την *main* μέθοδο και κάνοντας δεξί κλικ πάνω του επιλέγουμε από το μενού που προκύπτει *Run as* → *Java Application*. Τα αποτελέσματα της εκτέλεσης εμφανίζονται στο παράθυρο πολλαπλών λειτουργιών που βρίσκεται στο κάτω μέρος της οθόνης.



8.6 Εισαγωγή Έργου

Εάν έχουμε ήδη δημιουργήσει μία εργασία (όχι απαραίτητα με το **Eclipse**) και θέλουμε να την επεξεργαστούμε, τότε αρκεί να δημιουργήσουμε μία νέα εργασία στο **Eclipse** και κατόπιν να ψάξουμε για τα '.java' αρχεία της. Για να γίνει αυτό δημιουργούμε μία νέα εργασία όπως προηγουμένως και επιλέγουμε την διαδρομή που βρίσκονται τα αρχεία του προγράμματος μας. Πηγαίνοντας λοιπόν στο *File* → *New* → *Java Project* εμφανίζεται η παρακάτω οθόνη



Επιλέγουμε "*Create project from existing source*" και από κάτω εισάγουμε την διαδρομή που βρίσκονται τα αρχεία της εργασίας. Κατόπιν δίνουμε ένα όνομα στην εργασία που εισάγουμε και πατώντας *Finish* εισάγεται η εργασία στο **Eclipse.**

8.7 Πληροφορίες

Αναλυτικότερες πληροφορίες για το Eclipse μπορείτε να βρίτε επιλέγοντας *Help* → *Help Contents.* Αξίζει να δείτε τις δυνατότητες αποσφαλμάτωσης (debugging) που σας προσφέρονται, καθώς και τις ευκολίες για τη δημιουργία jars (Java Archives, ένα αρχείο με κατάληξη '.jar' που μπορεί να περιέχει πολλά Java αρχεία).

9. Αναφορές

- **Java API** http://java.sun.com/javase/6/docs/api/

- **Ελληνικά βιβλία**
  1. *JAVA with UML: Object oriented design and programming (in Greek)* by E. Lervik and V Havdal, Klidarithmos Publications, 2005
  2. *Programming in Java (in Greek)* (1st Edition) by I.Kavouras, Klidarithmos Publications, 2003

- **Αγγλικά βιβλία**
  1. *Program Development in Java: Abstraction, Specification and Object Oriented Design* by Barbara Liskov and John Guttag, Addison Wesley 2001
  2. *Java Collections: An Introduction to Abstract Data Types, Data Structures and Algorithms*, David A. Watt,Deryck F. Brown and Dave Watt, John Wiley & Sons 2001
  3. *Java 1.5 Program Design* by James P. Cohoon and Jack W. Davidson, McGraw-Hill, ISBN 007235447x, 2004
  4. *JAVA How to Program* (5th Edition) by Deitel & Deitel, Prentice Hall, 2003

- **Ελληνικές πηγές στο διαδίκτυο**
  1. *Διαλέξεις εισαγωγής στον αντικειμενοστρεφή προγραμματισμό χρησιμοποιώντας τη γλώσσα προγραμματισμού Java*, Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης (http://www.csd.uoc.gr/~hy252/)
  2. *Εισαγωγή στη Java*, Κωνσταντίνος Μαργαρίτης http://people.debian.org/~markos/guides/Course-Java-Notes.pdf
  3. *Τα βασικά της Java* Κακαρόντζας Γεώργιος (http://www.cs.teilar.gr/gkakaron/java/java.pdf)
  4. *Εισαγωγή στη γλώσσα Java,* Σημειώσεις για το μάθημα Αντικειμενοστρεφής Σχεδιασμός και Προγραμματισμός, Πανεπιστημίο Πειραιώς, Τμήμα Τεχνολογικής Εκπαίδευσης (http://users.softlab.ece.ntua.gr/~bxb/courses/unipi2001_te/00-CourseNotes/031-OO&Java/TE031-2.pdf)
  5. Σημειώσεις μαθήματος «Γλώσσες Προγραμματισμού», Πανεπιστήμιο Αθηνών, Τμήμα Μαθηματικών (http://www.math.uoa.gr/pps-epeaek2/Pliroforiki/112/gl_pr/Java-2005-SHMEIWSEIS.pdf)

- **Αγγλικά βιβλία στο διαδίκτυο**
  1. *The Java language specification 3rd Ed.* (http://java.sun.com/docs/books/jls/)
  2. *Thinking in Java 3rd Ed.* (http://www.mindview.net/Books/TIJ/)
  3. *Thinking in Patterns with Java* (Revision 0.9) by Bruce Eckel, published electronically by MindView Inc., May 20, 2003 (http://www.mindview.net/Books/TIPatterns/)