Σύντομη Εισαγωγή στη Java

Φυλλάδιο σημειώσεων για το 2ο εργαστήριο του μαθήματος

1 Χρήσιμες πληροφορίες και συνηθισμένα λάθη

Όνομα κλάσης και αρχείου Τα περισσότερα συστήματα Java απαιτούν ο κώδικας μίας public κλάσης να βρίσκεται σε ένα `.java` αρχείο που έχει το ίδιο όνομα με την κλάση. Αν αγνοήσουμε αυτή τη σύμβαση μπορεί να προκύψουν αρκετά προβλήματα κατά τη μεταγλώττιση.

```
// Wrong: File Lab2.java
public class Airplane extends Vehicle {
  int num_seats;
  public Airplane() {
    num_seats = 42;
  }
}
```

```
> javac Lab2.java
Lab2.java:2: class Airplane is public, should be declared in a file named Airplane.java
```

Σχέσεις κλάσεων και αρχείων Τα αρχεία Java έχουν μία πολύ ειδική σχέση με τις κλάσεις που βρίσκονται σε αυτά. Σε γενικές γραμμές, οι κανόνες είναι οι εξής:

1. Κάθε κλάση ορίζεται σε ακριβώς ένα αρχείο.
2. Το πολύ μία public κλάση μπορεί να οριστεί σε ένα αρχείο.
3. Αν μία public κλάση βρίσκεται σε ένα αρχείο, τότε το όνομα της κλάσης και το όνομα του αρχείου πρέπει να είναι τα ίδια.

Σύγκριση δύο αντικειμένων Ο τελεστής ισότητας == χρησιμοποιείται για τη σύγκριση δύο αναφορών σε αντικείμενα. Ελέγχει δηλαδή αν δύο αναφορές δείχνουν στο ίδιο αντικείμενο. Για τη σύγκριση των ίδιων των αντικειμένων πρέπει να χρησιμοποιηθεί η μέθοδος `equals` η οποία κληρονομείται σε όλες τις κλάσεις από την `java.lang.Object`. Κατά συνέπεια κληρονομείται και στα strings καθότι και τα strings στη Java είναι αντικείμενα της κλάσης `java.lang.String`!

```
// Wrong way to find out if the first argument is "-a"
if (args[0] == "-a")
  optionsAll = true;

// Right way
if ("-a".equals(args[0]))
  optionsAll = true;
```

Αρχικοποίηση πινάκων αντικειμένων Στη Java ένας πίνακας που δηλώνεται ότι είναι πίνακας σε αντικείμενα κάποιου τύπου, στην πραγματικότητα είναι ένας πίνακας αναφορών σε αντικείμενα αυτού του τύπου. Έτσι, όλα τα στοιχεία του πίνακα αρχικοποιούνται αυτόματα σε null. Είναι, λοιπόν, απαραίτητο να θέσουμε κάθε στοιχείο του πίνακα σε μία πραγματική αναφορά σε κάποιο αντικείμενο του σωστού τύπου.

```
// Wrong way to create an array of data buffers
StringBuffer[] myTempBuffers;
myTempBuffers = new StringBuffer[3];
myTempBuffers[0].add(data);
```

```
// Right way to create an array of data buffers and initialize it
StringBuffer[] myTempBuffers;
myTempBuffers = new StringBuffer[3];
for (int ix = 0; ix < myTempBuffers.length; ix++)
  myTempBuffers[ix] = new StringBuffer();
myTempBuffers[0].add(data);
```

Επισκίαση πεδίου από τοπική μεταβλητή Η Java επιτρέπει στον προγραμματιστή να ορίσει μέσα σε μεθόδους τοπικές μεταβλητές με ονόματα ίδια με αυτά των πεδίων της κλάσης. Σε αυτές τις περιπτώσεις οι τοπικές μεταβλητές έχουν προτεραιότητα και, όπου αυτές εμφανίζονται, επισκιάζουν τα πεδία της κλάσης. Υπάρχουν δύο τρόποι για την επίλυση αυτού του προβλήματος, η χρήση της δομής this.<πεδίο> όταν αναφερόμαστε σε κάποιο πεδίο και η μετονομασία είτε των πεδίων είτε των τοπικών μεταβλητών.

```
// Wrong: it doesn't do what the programmer intended
public class ... {
  int i = 0; int j = 0;  // target coordinates
  public boolean hits(Point[] p2list) {
    for (int i = 0; i < p2list.length; i++) {
      Point p2 = p2list[i];
      if (p2.x == i && p2.y == j)
        return true;
    }
    return false;
  }
}
```

```
// One way to fix the problem
public class ... {
  int i = 0; int j = 0;  // target coordinates
  public boolean hits(Point[] p2list) {
    for (int i = 0; i < p2list.length; i++) {
      Point p2 = p2list[i];
      if (p2.x == this.i && p2.y == this.j)
        return true;
    }
    return false;
  }
}
```

```
// A (better?) way to fix the problem
public class ... {
  int x = 0; int y = 0;  // target coordinates
  public boolean hits(Point[] p2list) {
    for (int i = 0; i < p2list.length; i++) {
      Point p2 = p2list[i];
      if (p2.x == x && p2.y == y)
        return true;
    }
    return false;
  }
}
```

Κλήση του κατασκευαστή της υπερκλάσης Όταν μία κλάση επεκτείνει κάποια άλλη, τότε ο κατασκευαστής της υποκλάσης πρέπει με κάποιο τρόπο να καλεί τον κατασκευαστή της υπερκλάσης. Συνήθως αυτό επιτυγχάνεται βάζοντας μία κλήση στον κατασκευαστή της υπερκλάσης στην πρώτη γραμμή του κατασκευαστή της υποκλάσης, γράφοντας δηλαδή κάτι σαν και το `super(arg1, ..., argN)`. Αν η πρώτη γραμμή του κατασκευαστή της υποκλάσης δεν είναι κάτι σαν και το παραπάνω, ο compiler συνήθως εισάγει μία κλήση `super()` χωρίς ορίσματα. Αυτό όμως προκαλεί προβλήματα αν η υπερκλάση δεν διαθέτει έναν κατασκευαστή που να μην δέχεται παραμέτρους.

```
// Wrong
public class JavaClassFile extends File {
  String classname;
  public JavaClassFile(String cl) {
    classname = cl;
  }
}
```

```
// Right
public class JavaClassFile extends File {
  String classname;
  public JavaClassFile(String cl) {
    super(cl + ".class");
    classname = cl;
  }
}
```

Οι static και οι λοιπές Ορίζουμε μία μέθοδο ως static, όπως τη main, για να δηλώσουμε ότι δεν χρειάζεται να δημιουργήσουμε ένα στιγμιότυπο της κλάσης που την περιέχει για να την καλέσουμε. Όμως από τη στιγμή που δεν έχει δημιουργηθεί ένα στιγμιότυπο της κλάσης, η main δεν μπορεί να χρησιμοποιήσει πεδία της κλάσης ούτε και να καλέσει κάποιες από τις μεθόδους της, γιατί δεν υπάρχουν! Για την προσπέλαση ενός πεδίου της κλάσης, θα πρέπει πρώτα να δημιουργήσουμε ένα στιγμιότυπό της.

```
// Wrong
public class StaticDemo {
  public String my_member_variable = "somedata";
  public static void main(String[] args) {
    // Access a non-static member from static method
    System.out.println("This generates a compiler error" +
      my_member_variable);
  }
}
```

```
// Right
public class NonStaticDemo {
  public String my_member_variable = "somedata";
  public static void main(String[] args) {
    NonStaticDemo demo = new NonStaticDemo();
    // Access member variable of demo
    System.out.println("This WON'T generate an error" +
      demo.my_member_variable);
  }
}
```

Όσον αφορά στις μεθόδους μπορούμε είτε να κάνουμε ό,τι κάναμε και για τα πεδία είτε να ορίσουμε και τη μέθοδο που θέλουμε να καλέσουμε ως static.

```
// Wrong
public class DivTest {
  boolean divisible(int x, int y) {
    return (x % y == 0);
  }
  public static void main(String[] args) {
    int v1 = 14, v2 = 9;
    if (divisible(v1, v2))      // compiler error!
      System.out.println(v1 + " is a multiple of " + v2);
    else
      System.out.println(v2 + " does not divide " + v1);
  }
}
```

```
// One way to fix the problem
public class DivTest {
  int modulus;
  public DivTest(int m) { modulus = m; }
  boolean divisible(int x) {
    return (x % modulus == 0);
  }
  public static void main(String[] args) {
    int v1 = 14, v2 = 9;
    DivTest tester = new DivTest(v2);
    if (tester.divisible(v1)
      System.out.println(v1 + " is a multiple of " + v2);
    else
      System.out.println(v2 + " does not divide " + v1);
  }
}
```

```
// Another way to fix the problem
public class DivTest {
  static boolean divisible(int x, int y) {
    return (x % y == 0);
  }
  public static void main(String[] args) {
    int v1 = 14, v2 = 9;
    if (divisible(v1,v2))
      System.out.println(v1 + " is a multiple of " + v2);
    else
      System.out.println(v2 + " does not divide " + v1);
  }
}
```

Μέθοδος ή κατασκευστής Οι κατασκευαστές αντικειμένων στην Java μοιάζουν πολύ με μεθόδους. Οι μόνες εμφανείς συντακτικές διαφορές μεταξύ των δύο είναι ότι οι κατασκευαστές δεν έχουν τύπο επιστροφής στον ορισμό τους και το όνομά τους είναι ίδιο με το όνομα της κλάσης. Παρόλα αυτά, και οι μέθοδοι επιτρέπεται να έχουν το ίδιο όνομα με την κλάση.

```
// Wrong
public class IntList {
  Vector list;
  // This may look like a constructor, but it is a method
  public void IntList() {
    list = new Vector();
  }
  public append(int n) {
    list.addElement(new Integer(n));
  }
}
```

```
// Right
public class IntList {
  Vector list;
  // This is a real constructor
  public IntList() {
    list = new Vector();
  }
  public append(int n) {
    list.addElement(new Integer(n));
  }
}
```

Oh strings! Τα strings στην Java είναι αντικείμενα που δεν επιδέχονται μετατροπή (immutable). Αυτό σημαίνει ότι δεν τα μεταχειριζόμαστε ως πίνακες χαρακτήρων — που όπως όλοι οι πίνακες επιδέχονται μετατροπές (mutable).

```
// Wrong
public static void main(String[] args) {
  String test1 = "Today is ";
  appendTodaysDate(test1);
  System.out.println(test1);
}
public static void appendTodaysDate(String line) {
  line = line + (new Date()).toString();
}
```

```
// One way to fix the problem
public static void main(String[] args) {
  String test1 = "Today is ";
  test1 = appendTodaysDate(test1);
  System.out.println(test1);
}
public static String appendTodaysDate(String line) {
  return (line + (new Date()).toString());
}
```

```
// Another way to fix the problem
public static void main(String[] args) {
  StringBuffer test1 = new StringBuffer("Today is ");
  appendTodaysDate(test1);
  System.out.println(test1.toString());
}
public void appendTodaysDate(StringBuffer line) {
  line.append((new Date()).toString());
}
```

Μεγάλα strings που εκτείνονται σε περισσότερες γραμμές:

```
// Wrong
String s = "A very long string which just happens to go over the
  end of a line and causes a problem with the compiler";

//Right
String s = "A very long string which just happens to go over " +
  "the end of a line and causes a problem with the compiler";
```

Κατά τιμή και (όχι) κατ' αναφορά Η Java χρησιμοποιεί μόνο πέρασμα παραμέτρων κατά τιμή. Όταν περνάμε σαν παράμετρο έναν πρωτόγονο τύπο δεδομένων, όπως char, int, float, double, αυτό σημαίνει ότι στη μέθοδο που καλείται περνάει ένα αντίγραφο της τιμής αυτού του τύπου, το οποίο μπορεί μεν να μεταβληθεί αλλά έχει διάρκεια ζωής μέχρι το τέλος της μεθόδου. Όταν όμως περνάμε σαν παράμετρο ένα αντικείμενο, αυτό που αντιγράφεται είναι η *αναφορά* στο αντικείμενο (τα αντικείμενα προσπελάζονται πάντα μέσω αναφορών στη Java) και επομένως το πέρασμα μιας τέτοιας παραμέτρου μοιάζει με το πέρασμα κατ' αναφορά σε άλλες γλώσσες. Οι αλλαγές που η μέθοδος θα κάνει στο αντικείμενο θα είναι ορατές και μετά την κλήση της μεθόδου.

Casting Η Java επιτρέπει στον προγραμματιστή να μεταχειριστεί ένα αντικείμενο μίας υποκλάσης σαν αντικείμενο της υπερκλάσης της. Αυτό συμβαίνει γιατί το upcasting στην Java γίνεται αυτόματα. Αντίθετα, το downcasting πρέπει να δηλώνεται ρητά.

```
// Wrong
Object arr[] = new Object[10];
arr[0] = "m";
arr[1] = new Character('m');
String arg = args[0];
if (arr[0].compareTo(arg) > 0)
  System.out.println(arg + " comes before " + arr[0]);
```

```
// Right
Object arr[] = new Object[10];
arr[0] = "m";
arr[1] = new Character('m');
String arg = args[0];
if ( ((String)arr[0]).compareTo(arg) > 0)
  System.out.println(arg + " comes before " + arr[0]);
```

```java
// File: Input.java
import java.io.*;

public class Input {
  public static void main(String[] args) {
    BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));
    String line = null;
    int sum = 0;
    try {
      while (true) {
        line = reader.readLine();
        if (line == null || line.equals("stop")) {
          System.out.println("Sum: " + sum);
          return;
        }
        sum += Integer.parseInt(line);
      }
    } catch (Exception e) { // Bad practice!
      System.out.println("Something went wrong: " + e);
    }
  }
}
```

Σχήμα 1: Ανάγνωση από το standard input.

2 FAQ

E: Πώς διαβάζω από το standard input;

A: Με 3-4 *Reader κάτι μπορεί να γίνει, όπως φαίνεται και στο Σχήμα 1.

Η Java για να χειρίζεται και να διαβάζει αρχεία χρησιμοποιεί τα I/O Streams που ορίζονται στο `java.io` package. Ένας εύκολος τρόπος για διάβασμα από αρχείο είναι χρησιμοποιώντας την κλάση BufferedReader, όπως φαίνεται στο παραπάνω παράδειγμα. Αρχικά, φτιάχνουμε ένα InputStream (FileReader) με το αρχείο εισόδου και με αυτό δημιουργούμε ένα αντικείμενο κλάσης BufferedReader. Με το BufferedReader που έχουμε ορίσει μπορούμε να διαβάσουμε ένα-ένα χαρακτήρα (`read()`) ή ολόκληρη γραμμή από χαρακτήρες (`String readLine()`). Ένα χρήσιμο εργαλείο είναι το `String[] split(String)`, που παίρνει ως παράμετρο ένα String και επιστρέφει έναν πίνακα από Strings. Η συνάρτηση αυτή χωρίζει το αρχικό String σε επιμέρους με βάση κάποιο διαχωριστικό όπως το κενό, το κόμμα, την παύλα, κ.λπ.

Για παράδειγμα το παρακάτω κομμάτι κώδικα:

```
String a = "boo:and:foo";
String b = a.split(":");
String c = a.split("o");
```

δίνει το εξής αποτέλεσμα:

```
b = { "boo", "and", "foo" }
c = { "b", "", ":and:f" }
```

Ένα παράδειγμα χρήσης των παραπάνω δίνεται στο Σχήμα 2.

Εναλλακτικά, για την επεξεργασία της εισόδου του προγράμματος μπορεί να χρησιμοποιηθεί και η κλάση `java.util.Scanner`. Για παράδειγμα, για να διαβάσει κανείς

```java
// File: Input.java
import java.io.*;

public class Input {
  public static void main(String[] args) {
    try {
      BufferedReader in = new BufferedReader(new FileReader("input.txt"));
      String line = in.readLine();
      String[] a = line.split(" ");
      int n = Integer.parseInt(a[0]);
      int m = Integer.parseInt(a[1]);
      int [][] floor = new int[m][n];

      for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
          floor[i][j] = in.read();
      in.close();
    }
    catch(IOException e) {
      e.printStackTrace();
    }
  }
}
```

Σχήμα 2: Ανάγνωση από το standard input.

τα χαρακτηριστικά ενός γράφου (πλήθος κόμβων και ακμών και βάρη ακμών) από το `stdin` θα μπορούσε να χρησιμοποιήσει κώδικα παρόμοιο με αυτόν που περιέχεται στο Σχήμα 3.

E: Πώς διαβάζω command line arguments;

A: Το διάβασμα command line arguments γίνεται απλά, περίπου όπως στη C. Το μόνο που χρειάζεται προσοχή είναι ότι κάθε argument θεωρείται String και αν θέλουμε να το χειριστούμε ως integer, ή κάποιον άλλο τύπο, πρέπει να το μετατρέψουμε.

Παράδειγμα (από την άσκηση του 2008 για το TV zapping):

Η είσοδος δίνεται ως εξής: Πρώτο όρισμα είναι ο μέγιστος αριθμός καναλιών και τα υπόλοιπα ορίσματα είναι τα κανάλια που είναι προγραμματισμένες οι τηλεοράσεις μια δεδομένη χρονική στιγμή. Δηλαδή θέλουμε να διαβάσουμε το πρώτο όρισμα και να το αποθηκεύσουμε σε μια μεταβλητή και τα υπόλοιπα ορίσματα τα αποθηκεύουμε σε ένα πίνακα.

```java
public static void main(String[] args) {
  int i,max,temp;
  int [] channels;

  if (args.length > 1){
    max = Integer.parseInt(args[0]);
    channels = new int[args.length-1];
    for (i = 1; i < args.length; i++)
      channels[i-1] = Integer.parseInt(args[i]);
  }
  else
    System.out.println("Wrong input\n");
}
```

```java
// File: Graph.java
import java.util.Scanner;

public class Graph {
  public static void main(String[] args) {
    ...
    Scanner in = new Scanner(System.in);
    n = in.nextInt();
    m = in.nextInt();
    for (int i = 0 ; i < m ; i ++) {
       int source = in.nextInt();
       int target = in.nextInt();
       int weight = in.nextInt();
       edges.addEdge(source, target, weight);
    }
    in.close();
    ...
}
```

Σχήμα 3: Ανάγνωση από το standard input με χρήση `Scanner`.

E: Τι είναι οι iterators και πώς χρησιμοποιούνται;

A: Σε κάποια κλάση που υλοποιεί το interface Collection (ή κάποιο ειδικότερο interface, π.χ. List ή Set), όπως οι ArrayList, Vector, HashSet, κ.λπ., για να διατρέξουμε τα στοιχεία τους χρησιμοποιούμε το interface `iterator`. Ειδικά για λίστες υπάρχει και το subinterface `ListIterator`, που επιτρέπει να διατρέξουμε τη λίστα και προς την αντίθετη κατεύθυνση και να τροποποιήσουμε κάποιο στοιχείο της.

```java
class IteratorDemo {
  public static void main(String args[]) {
    // create an array list
    ArrayList al = new ArrayList();
    // add elements to the array list
    al.add("C"); al.add("A"); al.add("E");
    al.add("B"); al.add("D"); al.add("F");

    // use iterator to display contents of al
    System.out.print("Original contents of al: ");
    Iterator itr = al.iterator();
    while (itr.hasNext()) {
      String element = (String)itr.next();
      System.out.print(element + " ");
    }
    System.out.println();

    // modify objects being iterated
    ListIterator litr = al.listIterator();
    while (litr.hasNext()) {
      String element = (String)litr.next();
      litr.set(element + "+");
    }
    System.out.print("Modified contents of al: ");
    itr = al.iterator();
    while (itr.hasNext()) {
      String element = (String)itr.next();
      System.out.print(element + " ");
    }
    System.out.println();
```

```
    // now, display the list backwards
    System.out.print("Modified list backwards: ");
    while (litr.hasPrevious()) {
      String element = (String)litr.previous();
      System.out.print(element + " ");
    }
    System.out.println();
  }
}
```

Το output του παραπάνω κώδικα είναι:

```
Original contents of al: C A E B D F
Modified contents of al: C+ A+ E+ B+ D+ F+
Modified list backwards: F+ D+ B+ E+ A+ C+
```

Με τα generics, δε χρειάζεται να κάνουμε cast σε συγκεκριμένο τύπο αντικειμένου.

```
ArrayList<String> al = new ArrayList<String>();
System.out.print("Original contents of al: ");
Iterator itr = al.iterator();
while (itr.hasNext()) {
  String element = itr.next();
  System.out.print(element + " ");
}
```

Κάτι πιο εύκολο από τους iterators, είναι το “for each” loop. Ο παραπάνω κώδικας γράφεται ως εξής:

```
System.out.print("Original contents of al: ");
for(String element : a)
  System.out.print(element + " ");
```

E: Πώς μπορώ να διατάξω ή να συγκρίνω αντικείμενα μιας κλάσης με βάση κάποιο πεδίο;

A: Η java υποστηρίζει διάταξη αντικειμένων σε κάποιες βασικές κλάσεις όπως Integer, String. Αν θέλουμε να διατάξουμε κάποια αντικείμενα μια κλάσης που φτιάξαμε μόνοι μας πρέπει να δηλώσουμε ότι η κλάση υλοποιεί τη διαπροσωπεία `Comparable` και να ορίσουμε μια μέθοδο `compareTo()`, η οποία θα καθορίζει με βάση ποιο πεδίο θέλουμε να γίνεται η διάταξη των αντικειμένων μας. Η μέθοδος αυτή συγκρίνει δύο αντικείμενα της κλάσης: το τρέχον αντικείμενο και ένα άλλο το οποίο περνιέται σαν όρισμα στη μέθοδο. Η τιμή που επιστρέφει η μέθοδος ορίζει τη φυσική σειρά ταξινόμησης για τα αντικείμενα αυτής της κλάσης: Εάν το τρέχον αντικείμενο πρέπει να ταξινομηθεί πριν (πιο πάνω) από το άλλο αντικείμενο, επιστρέφει -1. Εάν το τρέχον αντικείμενο πρέπει να ταξινομηθεί μετά (πιο κάτω) από το άλλο αντικείμενο, επιστρέφει 1. Εάν τα δύο αντικείμενα είναι ίσα, επιστρέφει 0. Ο κώδικας του συγκεκριμένου παραδείγματος βρίσκεται στην Εικόνα 4.

```
import java.util.*;

public class Shop {
  public static class Item implements Comparable {
    int id;
    String name;
    float price;

    public Item(int idIn, String nameIn, float priceIn) {
      id = idIn;
      name = nameIn;
      price = priceIn;
    }

    public int compareTo(Object obj) {
      Item temp = (Item)obj;
      if (this.price < temp.price)
        return -1;
      else
        if (this.price > temp.price)
          return 1;
        else return 0;
    }
  }

  public static void main(String[] args) {
    Item item1 = new Item(0,"pen", 2);
    Item item2 = new Item(1,"pencil", 1);
    Item item3 = new Item(1,"post-it", 3);

    LinkedList catalog = new LinkedList();
    catalog.add(item1);
    catalog.add(item2);
    catalog.add(item3);
    Collections.sort(catalog);
    Item min_item = (Item)Collections.min(catalog);
    System.out.println("The cheapest item is " + min_item.name);
  }
}
```

Σχήμα 4: Μια κλάση οριζόμενη από το χρήστη η οποία επιτρέπει σύγκριση μεταξύ αντικειμένων κλάσης Item.

3 Λίγη θεωρία...

Χρήσιμες Δομές Δεδομένων

Vector Είναι δομή δεδομένων παρόμοια με έναν παραδοσιακό πίνακα, εκτός από το γεγονός ότι μπορεί να μεγαλώνει, όταν απαιτείται, για να δέχεται νέα στοιχεία, ενώ μπορεί επίσης και να μικραίνει. Όπως και στον πίνακα, τα στοιχεία ενός αντικειμένου `Vector` μπορούν να προσπελάζονται μέσω μιας τιμής δείκτη. Το πλεονέκτημα που παρέχει είναι ότι δε χρειάζεται να προβληματιστούμε σχετικά με το μέγεθος του αντικειμένου κατά τη στιγμή της δημιουργίας του καθώς μεγαλώνει και μικραίνει αυτόματα, όταν και όπως απαιτείται.

```
// Create Vector
Vector v = new Vector();
// Create vector with initial capacity
Vector v = new Vector(25);
// Create Vector with initial capacity and step
Vector v = new Vector(25,5);
// Add and remove elements
v.add("orange");
v.add("apple");
v.add(2,"lemon");
v.remove(2);
// Get the last element added
String s = (String)v.lastElement();
// Get elements using a pointer
String s1 = (String)v.get(0);
```

Hashtable Το Hashtable είναι μια δομή που αντιστοιχίζει κλειδιά (keys) σε τιμές (values). Αυτό είναι χρήσιμο όταν θέλουμε να προσπελάσουμε τα δεδομένα μέσω ενός συγκεκριμένου κλειδιού και όχι ενός ακέραιου δείκτη. Τα κλειδιά και οι τιμές μπορούν να είναι οποιουδήποτε τύπου αντικείμενα. Αντικείμενα εισάγονται ως τιμές με κάποιο αναγνωριστικό κλειδί, με το οποίο μετά μπορούν να καλεστούν ή να αφαιρεθούν από το Hashtable.

```
// Create a hashtable
Hashtable numbers = new Hashtable();

// Add elements
numbers.put("one", new Integer(1));
numbers.put("two", new Integer(2));
numbers.put("three", new Integer(3));

// Find an element
Integer n = (Integer)numbers.get("two");
if (n != null) {
  System.out.println("two = " + n);
}
```

Modifiers

Οι modifiers είναι δεσμευμένες λέξεις που προσθέτονται μπροστά σε ορισμούς για να αλλάξει η σημασία τους.

Έλεγχος Πρόσβασης

Σύμφωνα με το προκαθορισμένο επίπεδο πρόσβασης, μία μέθοδος ή μεταβλητή είναι ορατή σε οποιαδήποτε άλλη κλάση περιέχεται στο ίδιο πακέτο. Για να αλλάξουμε το επίπεδο πρόσβασης χρησιμοποιούμε τους παρακάτω modifiers.

private Η μέθοδος ή η μεταβλητή είναι ορατή μόνο μέσα στην κλάση στην οποία ορίστηκε. Μια μεταβλητή private μπορεί να χρησιμοποιείται από μεθόδους της ίδιας κλάσης, αλλά όχι από αντικείμενα οποιωνδήποτε άλλων κλάσεων. Μια μέθοδος private μπορεί να καλείται από άλλες μεθόδους της κλάσης, αλλά όχι από άλλες κλάσεις.

public Η μέθοδος ή η μεταβλητή είναι ορατή σε όλες τις κλάσεις. Η μέθοδος main() μιας εφαρμογής πρέπει να είναι public.

protected Η μέθοδος ή η μεταβλητή είναι ορατή μόνο στις υποκλάσεις της κλάσης στην οποία έχουν οριστεί και στις άλλες κλάσεις του ίδιου πακέτου.

Άλλοι

static Η μέθοδος ή η μεταβλητή είναι της κλάσης, και όχι κάθε στιγμιοτύπου ξεχωριστά. Μπορεί να προσπελαστεί χωρίς να είναι απαραίτητη η δημιουργία καινούριου αντικειμένου-στιγμιοτύπου και αν αλλάξει, θα επηρεάσει όλα τα αντικείμενα αυτής της κλάσης.

final Υποδεικνύει ότι η κλάση, μέθοδος ή μεταβλητή δε θα αλλάξει. Συγκεκριμένα: μια κλάση final δεν μπορεί να έχει υποκλάσεις, μια μέθοδος final δεν μπορεί να υπερκαλύπτεται απ' οποιεσδήποτε υποκλάσεις, και μια μεταβλητή final δεν μπορεί να αλλάξει τιμή.

abstract Αφορά μια κλάση και σημαίνει ότι δεν μπορούν να δημιουργηθούν αντικείμενα αυτής της κλάσης, αλλά μόνο των υποκλάσεών της. Ουσιαστικά είναι μια κλάση που περιέχει κάποια χαρακτηριστικά και μεθόδους για μια ομάδα από υποκλάσεις, και επιτρέπεται να δημιουργήσουμε αντικείμενα μόνο των υποκλάσεών της. Οι abstract κλάσεις μπορούν να περιέχουν και abstract μεθόδους, οι οποίες είναι υπογραφές μεθόδων χωρίς υλοποίηση.(Δεν μπορεί να δηλωθεί μια μέθοδος abstract σε μια κλάση η οποία δεν είναι επίσης abstract).

4 Eclipse: Ένα πλήρες IDE για τη Java

Εγκατάσταση

Για Windows, Linux, Mac OS X η έκδοση Eclipse Classic είναι διαθέσιμη για download από τη σελίδα:

`http://www.eclipse.org/downloads/`

Δεν απαιτεί installation, απλό τρέξιμο του εκτελέσιμου `eclipse`. Προυποθέτει όμως ήδη εγκατεστημένη Java και χρόνο ή καλή σύνδεση για το download γιατί είναι 151 MB.

Ξεκινώντας

- Στην εκκίνηση του Eclipse, επιλέγουμε Workspace, δηλαδή τον φάκελο όπου θα αποθηκεύονται τα Project που θα γράφουμε και πατάμε OK.
- Κλείνουμε την οθόνη υποδοχής και μπροστά μας βλέπουμε το interface που θα χρησιμοποιούμε.

Λίγα λόγια για το interface

- Αποτελείται απο πολλά παράθυρα. Με διπλό κλικ στον τίτλο ενός παραθύρου ή `Ctrl+M`, μεταβαίνει σε full screen mode. Με τον ίδιο τρόπο επανέρχεται πάλι στο αρχικό του μέγεθος.
- Για να επανέλθει στην αρχική του διάταξη παραθύρων το περιβάλλον επιλέγουμε Window → Reset Perspective.
- Αριστερά βρίσκεται ο Package Exlorer όπου εμφανίζονται με τα περιεχόμενα τους όλα τα Project που θα δημιουργούμε και στο κέντρο βρίσκεται ο Editor όπου γράφουμε τον κώδικα.
- Κάτω βρίσκονται τα παράθυρα Problems, στο οποία εμφανίζονται τα λάθη του προγράμματος μας, και Console που αποτελεί την κονσόλα όπου διαβάζεται η είσοδος και εκτυπώνεται η έξοδος.

Το πρώτο project

Ένα Project στο Eclipse είναι μια ομαδοποίηση των αρχείων που αποτελούν ένα πρόγραμμα. Μπορεί να περιέχει `.java` αρχεία, μεταγλωττισμένα `.class` αρχεία, `.jar` αρχεία και διάφορα άλλα μέρη.

- Επιλέγουμε File → New → Java Project.
- Δίνουμε όνομα στο project (πεδίο Project Name) και επιλέγουμε Finish.
- Με διπλό κλικ στο project που δημιουργήθηκε, στον Package explorer, βλέπουμε ότι περιέχει ένα φάκελο src. Με δεξί κλικ στο φάκελο αυτό, επιλέγουμε New → Class για να δημιουργήσουμε την πρώτη μας κλάση. Με την ίδια μέθοδο μπορούμε αργότερα να δημιουργήσουμε και άλλες κλάσεις μέσα στο ίδιο Project.
- Στο πεδίο Name δίνουμε όνομα στην κλάση μας (δε χρειάζεται η επέκταση `.java`) και προαιρετικά τσεκάρουμε το κουτί με όνομα

  ```
  public static void main(String[] args)
  ```

  για να δημιουργηθεί αυτόματα ένας σκελετός για τη main στην κλάση μας. Επιλέγουμε Finish.
- Στον editor έχει δημιουργηθεί η κλάση και περιέχει (αν το έχουμε επιλέξει) μια άδεια `main` μέθοδο. Συμπληρώνουμε τον κώδικα μας. Για παράδειγμα για ένα απλό `Hello World` :

```
public class HelloWorld {
  /**
   * @param args
   */
  public static void main(String[] args) {
    // TODO Auto-generated method stub

    System.out.println("Hello, World");

  }
}
```

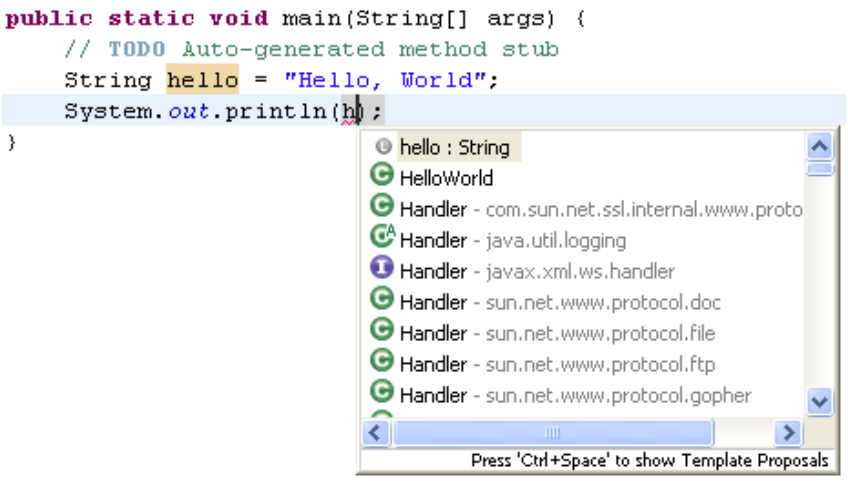
- Με File → Save. (ή `Ctrl+S`) το πρόγραμμα μεταγλωττίζεται. Είναι ισοδύναμο με την εντολή `javac HelloWorld.java` στη γραμμή εντολών. Αν υπάρχει κάποιο error στη μεταγλώττιση θα εμφανιστεί στο παράθυρο Problems στο κάτω μέρος της οθόνης, αμέσως μετά το Save.
- Αν δεν υπάρχουν errors τότε,στο `HelloWorld.java` στον Package Explorer με δεξί κλικ επιλέγουμε Run as → Java Application.
- Η έξοδος του προγράμματος τυπώνεται στο παράθυρο Console στο κάτω μέρος της οθόνης.

Για να τρέξουμε το πρόγραμμα που εκτελέσαμε την τελευταία φορά μπορούμε απλά να πατήσουμε `Ctrl+F11`.

Χρήσιμα Εργαλεία

Auto-complete Κάθε φορά που γράφουμε κάτι που πιστεύουμε ότι το Eclipse μπορεί να το "μαντέψει", αξίζει να δοκιμάζουμε το `Ctrl+Space` να δούμε αν θα το κάνει. Το Eclipse βρίσκει πράγματα όπως ονόματα μεθόδων, μεταβλητές σε εμβέλεια, ονόματα κλάσεων ή ακόμη και να κάνει προτάσεις για μια μεταβλητή που δηλώνουμε.

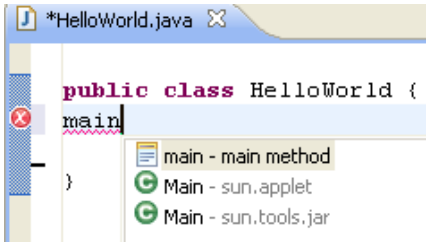
Για παράδειγμα, αν ορίσουμε ένα String hello και στη συνέχεια πάμε να το εκτυπώσουμε, πληκτρολογώντας απλά το πρώτο γράμμα του String, δηλαδή το h, και `Ctrl+Space` θα έχουμε το αποτέλεσμα που φαίνεται στην παρακάτω εικόνα :



Σχήμα 5: Auto-complete example.

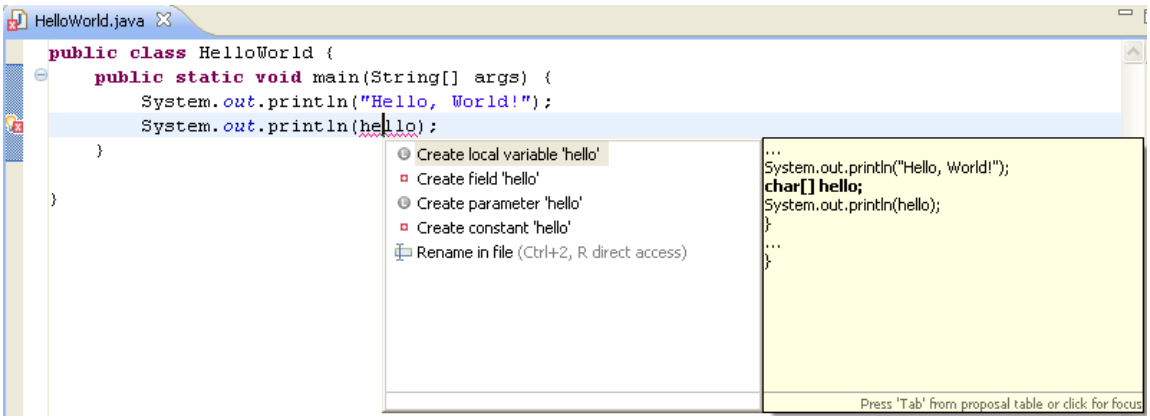
Code Templates Αν πληκτρολογήσουμε κάποιες λέξεις κλειδιά, το Eclipse μπορεί να γεννήσει μόνο του ένα πρότυπο για κάποια κομμάτια κώδικα που χρησιμοποιούμε συχνά. Για παράδειγμα με τη λέξη `main` και `Ctrl+Space`, επιλέγοντας το main-main method παράγεται ένας σκελετός της main (6). Ακόμα με `sysout` και `Ctrl+Space` παράγεται μια κλήση `System.out.println()` και με `foreach` και `Ctrl+Space` ένα for loop για τη διάσχιση ενός πίνακα ή άλλης δομής. Για να δείτε τα υπάρχοντα templates ή για να δημιουργήσετε νέα Window → Preferences → Java → Editor → Templates.

Formatter Αν ο κώδικας σας δεν είναι σωστά στοιχισμένος, είναι πολύ απλό να γίνει αυτόματα, πατώντας απλά `Ctrl+Shift+F` σε οποιοδήποτε σημείο του κώδικα σας για να στοιχίσετε όλο το αρχείο κώδικα, ή `Ctrl+I` για να στοιχίσετε μια συγκεκριμένη γραμμή.



Σχήμα 6: Main template example.

Quick Fix Όταν στον κώδικα μας υπάρχει κάποιο compile error, τότε στο εικονίδιο του αρχείου που έχει το error, αλλά και σε αυτό του project εμφανίζεται ένα κόκκινο X. Μπορούμε να εντοπίσουμε το λάθος στον κώδικα μας με διπλό κλικ στο συγκεκριμένο πρόβλημα στο παράθυρο Problems (η επιλέγοντας το κόκκινο σημάδι στην στήλη στα δεξιά του editor). Αριστερά από τη γραμμή όπου υπάρχει το error, υπάρχει είτε απλά ένα X, οπότε πρέπει να βρούμε μόνοι τη λύση στο πρόβλημα μας, είτε ένα X με μια μικρή λάμπα, που σημαίνει ότι το πολύ χρήσιμο Quick Fix του Eclipse έχει κάποια ιδέα για το πώς να μας βοηθήσει να το λύσουμε. Για να δούμε την πρόταση του, πατάμε `Ctrl+1` ή κάνουμε κλίκ στο είκονίδιο αυτό. Τότε, όπως φαίνεται και στην παρακάτω εικόνα, εμφανίζεται ένα μενού με προτάσεις για τη διόρθωση του λάθους απ' όπου επιλέγουμε κάποια που μας ικανοποιεί, εάν υπάρχει.



Σχήμα 7: Quick Fix menu.

Προσοχή : Στο Eclipse η μεταγλώττιση γίνεται (συνήθως) αυτόματα όταν κάνουμε Save, επομένως για να αντιληφθεί ότι διορθώσαμε κάποιο πρόβλημα, μετά τις αλλαγές μας επιλέγουμε πάντα `Ctrl+S` ή File → Save.

Setters-Getters Στη Java για να προσπελάζουμε και να αλλάζουμε τις τιμές private πεδίων κάποιας κλάσης, χρησιμοποιούμε setters και getters. Το eclipse μπορεί να δημιουργήσει αυτόματα τον κώδικα που χρειάζεται για αυτές τις μεθόδους. Πηγαίνουμε στο Source και επιλέγουμε Generate Getters and Setters. Στο παράθυρο που εμφανίζεται υπάρχει η λίστα των πεδίων της κλάσης στην οποία βρισκόμαστε. Μπορούμε να επιλέξουμε ποια πεδία θέλουμε να δημιουργηθεί αυτόματα κώδικας για μέθοδο Set ή/και Get. Επίσης μπορούμε να επιλέξουμε πού θα μπει ο κώδικας και το πεδίο πρόσβασης που επιθυμούμε να έχει (public, protected, default, private).