ΕΘΝΙΚΟ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΑΝΑΜΟΡΦΩΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ ΠΡΟΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΤΟΥ ΤΜΗΜΑΤΟΣ ΜΑΘΗΜΑΤΙΚΩΝ ΤΟΥ ΠΑΝΕΠΙΣΤΗΜΙΟΥ ΑΘΗΝΩΝ ΜΕ ΕΜΦΑΣΗ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ, ΤΗ ΔΙΔΑΚΤΙΚΗ ΚΑΙ ΤΙΣ ΕΦΑΡΜΟΓΕΣ ΤΩΝ ΜΑΘΗΜΑΤΙΚΩΝ

ΕΛΛΑΔΑ 2008

ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΥΡΩΠΑΪΚΟ ΤΑΜΕΙΟ ΠΕΡΙΦΕΡΕΙΑΚΗΣ ΑΝΑΠΤΥΞΗΣ

ΠΑΙΔΕΙΑ ΜΠΡΟΣΤΑ
2ο Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

# ΣΗΜΕΙΩΣΕΙΣ ΜΑΘΗΜΑΤΟΣ
# «ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ»

Κωνσταντίνος Π. Φερεντίνος
Διδάσκων ΠΔ 407/80

Οι σημειώσεις αυτές του μαθήματος «Γλώσσες Προγραμματισμού»
αναπτύχθηκαν στα πλαίσια του προγράμματος
**«ΕΠΕΑΕΚ 2 – Πρόγραμμα Αναβάθμισης Προπτυχιακών Σπουδών»**

ΙΟΥΝΙΟΣ 2005

# Σημειώσεις JAVA – 1η εβδομάδα

**Γιατί JAVA;**

1) Portable (φορητότητα): εκτέλεση του ίδιου κώδικα ανεξαρτήτως πλατφόρμας (αρκεί να είναι εγκατεστημένος ο αντίστοιχος μεταγλωττιστής Java) → Internet
2) Αντικειμενοστραφής αλλά απλούστερη της C++.
3) Μεγάλη βιβλιοθήκη κλάσεων (έτοιμων προγραμμάτων)
4) Χρησιμοποιεί στοιχεία της C.

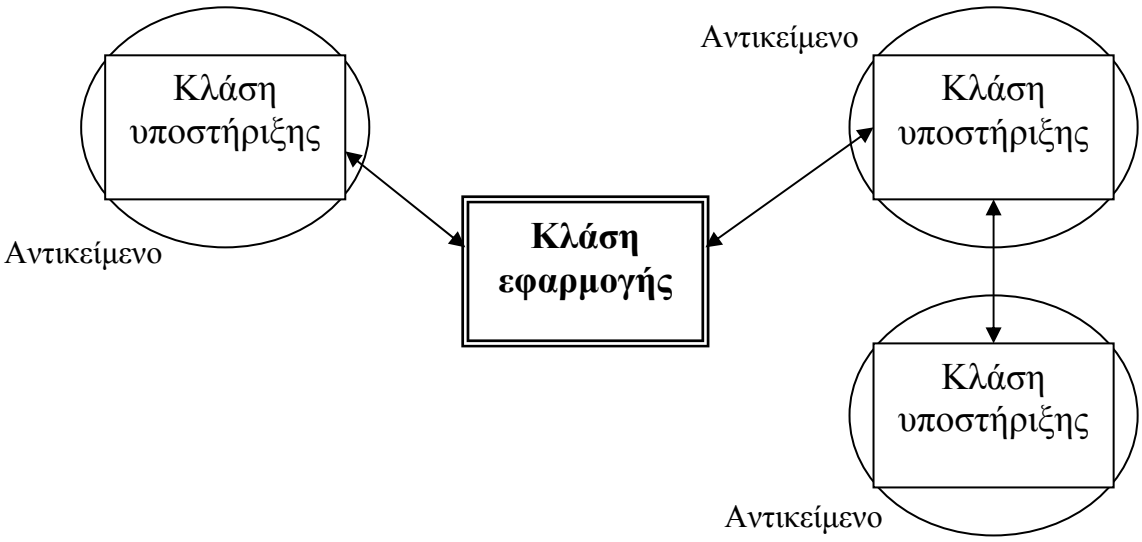
Άλλα χαρακτηριστικά της JAVA: Ασφαλής, κατανεμημένη, multithreaded.

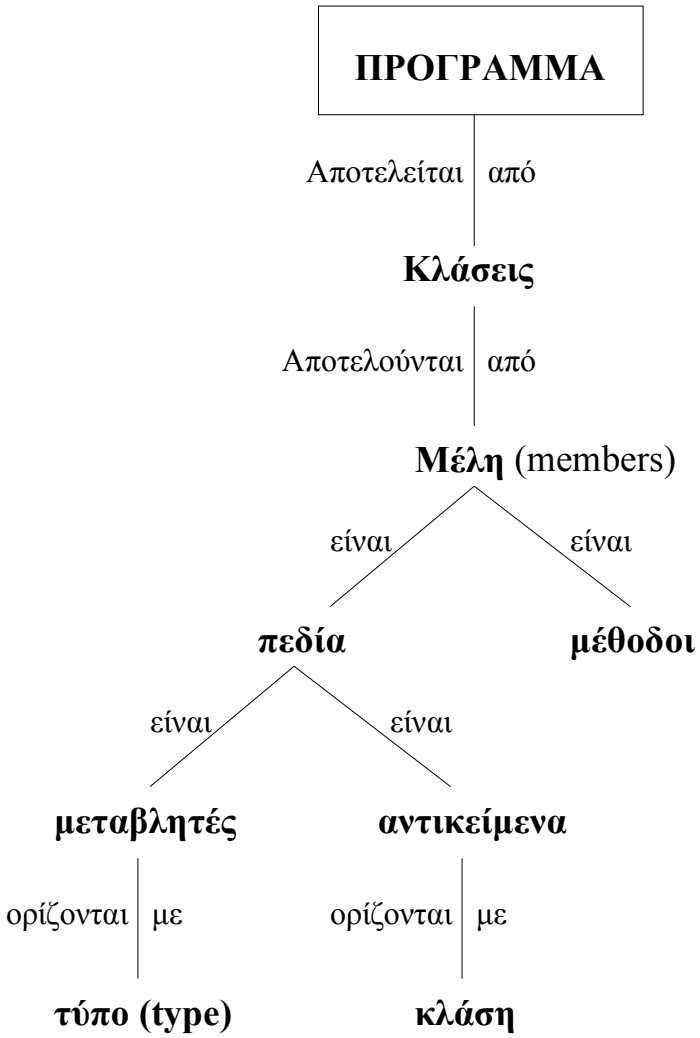
---

**Αλγόριθμος:** Βήμα προς βήμα διαδικασία για την επίλυση κάποιου προβλήματος. Το πλήθος των βημάτων πρέπει να είναι πεπερασμένο.

1) Καθορισμός στόχου
   - περιορισμοί;
   - παραδοχές; (υποθέσεις)
2) Καταμερισμός σε λογικά βήματα
   - απαραίτητα inputs
   - σταθερές
   - user inputs
   - data από αρχείο
   - data από άλλο μέρος του προγράμματος
3) Επιλογή των outputs
   - Ποια;
   - Που; (οθόνη; αρχείο; εκτυπωτή; αλλού;)
4) Διαδικασίες για τη λύση του προβλήματος
   - Καταμερισμός σε υπο-προβλήματα / ενέργειες / εξισώσεις
   - Εφαρμογή βημάτων σε λογική σειρά

---






**Ανάπτυξη εφαρμογής:**

1) Κώδικας Java → κλάσεις → αρχεία: Class1.java, Class2.java, …
2) Compiler (μεταγλωτιστής): javac Class1.java → Class1.class, …
3) Interpreter (διερμηνευτής): εκτελεί αρχεία .class: java Class1, …

Αν είναι applet: εκτέλεση μέσω browser ή appletviewer.

**Απλή εφαρμογή:** (Μία μόνο κλάση → κλάση εφαρμογής)

```
class Hello
{
  public static void main (String [ ] args)
  {
     System.out.println("Hello!");
  }
}
```

→ save: Hello.java

C:\> javac Hello.java

C:\> java Hello

---

## **ΚΛΑΣΗ:** σύνθετος τύπος δεδομένων

**Types:**

- primitive types (πρωτογενείς) (Boolean, int, double, κτλ.)
- String
- **Κλάσεις** → σύνθετη δομή σχεδιασμένη απ' τον προγραμματιστή

Όπως μια μεταβλητή "a" μπορεί να είναι ακέραια:

```
public int a;
```

έτσι και ένα αντικείμενο "fiat" μπορεί να είναι της κλάσης Car:

```
Car fiat = new Car();
```

(ένα αντικείμενο λέγεται και *στιγμιότυπο (instance)* μιας κλάσης)

**ΚΛΑΣΗ:**

| Επικεφαλίδα |
| --- |
| Πεδία (μεταβλητές, αντικείμενα) |
| Μέθοδοι |

- Constructors (κατασκευαστές ή δημιουργοί) (περισσότερα σε επόμενη παράδοση)

## Κάποιοι κανόνες σύνταξης της Java:

Δεσμευμένες λέξεις (reserved): class, public, private, int, double, char, κ.α.

Identifiers → ονόματα πραγμάτων (μεθόδων, κλάσεων, μεταβλητών...)

- ξεκινάνε με: γράμμα, _, #
- περιέχουν: γράμματα, νούμερα, _, #
- μέχρι 256 χαρακτήρες (3-15 συνήθως)
- χωρίς κενά

Συνήθως:

- ονόματα Κλάσεων ξεκινάνε με Κεφαλαίο
- υπόλοιπα ονόματα: με μικρό
- σταθερές: ΟΛΑ κεφαλαία

---

**Παράδειγμα Κλάσης:**

*Επικεφαλίδα:*

```
public class ToKelvin
{
```

*Πεδία:*

```
   private double C2K = 273.15;
```

*Μέθοδος:*

```
   public double returnKelvin (double tempC)
   {
      return (tempC + C2K);
   }

} // end class
```

Ορισμός μεταβλητής:

```
[ορατότητα] <τύπος> <όνομα> [= <τιμή>];
     ↓          ↓
  public      int
  private   double
     .          .
     .          .
     .          .
```

```
π.χ.: public double var1;
      private int var2 = 5;
```

Ορισμός μεθόδου:

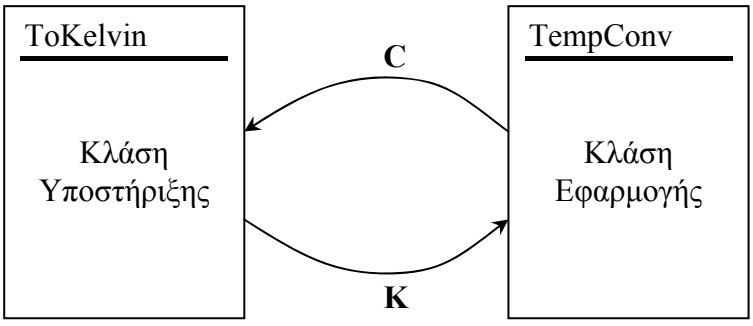
```
[ορατότητα] <τύπος> <όνομα> ([παράμετροι])
     ↓          ↓
  public      void
  private     int
     .       double
     .          .
     .          .
```

`void`: η μέθοδος ***δεν*** επιστρέφει τίποτα
`int, double, char, ...` : η μέθοδος επιστρέφει μια μεταβλητή του αντίστοιχου τύπου, άρα περιέχει τη λέξη `return`.

Οι παράμετροι, εάν υπάρχουν, είναι της μορφής:
`<όνομα_μεθόδου> (<τύπος> <παράμ.1>, <τύπος> <παράμ.2>, ...)`

---

Η κλάση `ToKelvin` είναι μια κλάση υποστήριξης. Για να «τρέξει» πρέπει να φτιάξουμε μια κλάση εφαρμογής (`TempConv`), έτσι ώστε:



```
public class TempConv
{
   public static void main (String [ ] args)
   {

      // dimiourgia metavlitwn:

      double C = 15.5;
      double K;

      // dimiourgia antikeimenou tis klasis ToKelvin:

      ToKelvin ToK = new ToKelvin();


      // klisi tis methodou returnKelvin me apostoli
      // tis thermokrasias se vathmous Celsiou
      // kai epistrofi tis se vathmous Kelvin:

      K = ToK.returnKelvin(C);

      // ektypwsi stin othoni tis therm. se Kelvin:

      System.out.println(K);

   } // end main

} // end class
```

---

## ΠΟΛΥ ΣΗΜΑΝΤΙΚΟ:

**Εκτέλεση μεθόδου του στιγμιότυπου (αντικειμένου) μιας κλάσης:**

1) Δημιουργία στιγμιότυπου (αντικειμένου) της κλάσης:

**<όνομα_Κλάσης> <όνομα_αντικειμ> = new <όνομα_Κλάσης> ( );**

π.χ.: `Class1 obj1 = new Class1();`

2) Κλήση μεθόδου (invoke method):

**<όνομα_αντικειμ>.<όνομα_μεθόδου> ( );**

π.χ.: `obj1.method1();`

**Παράδειγμα:**

Κλάση υποστήριξης:

```
public class Class1
{
   private int a = 5;
   private int b = 3;
   public int c;

   public void method1()
   {
      c = 2 * a + b;
   }
}
```

Κλάση εφαρμογής:

```
public class Class2
{
   public static void main (String [ ] args)
   {

      // dimiourgia 2 antikeimenwn tis Class1:

      Class1 obj1 = new Class1();
      Class1 obj2 = new Class1();

      // klisi tis methodou method1 gia to antikeimeno obj1

      obj1.method1();


      // ektypwsi stin othoni:

      System.out.println(obj1.c); // tha typwsei: 13
      System.out.println(obj2.c); // tha typwsei: 0

   } // end method

} // end class
```

Το `obj1.c` είναι 13 γιατί έχουμε καλέσει πριν τη μέθοδο `method1` για το συγκεκριμένο αντικείμενο (`obj1`). Το `obj2.c` είναι 0 γιατί δεν έχουμε καλέσει τη `method1` για το αντικείμενο αυτό (`obj2`).

Καλύτερη προσέγγιση → *Χρήση accessor methods:*

Κάνουμε τις εξής αλλαγές στον κώδικα:

→ Κάνουμε `private` τη μεταβλητή `c` της κλάσης `Class1`

→ Προσθέτουμε μια accessor method στην `Class1`:

```
public int method2()
 {
   return c;
 }
```

→ Στην `Class2` αλλάζουμε τις "System.out.println" διαταγές σε:

```
System.out.println(obj1.method2());
System.out.println(obj2.method2());
```

Το αποτέλεσμα είναι ακριβώς το ίδιο. **Όμως,** με τον τρόπο αυτό **δεν** επιτρέπεται στην `Class2` να αλλάξει τη μεταβλητή `c` όπως θέλει, άρα η `c` αλλάζει μόνο όπως ο προγραμματιστής έχει επιλέξει στην `Class1` (μέσω της `method1()` ).
Αυτός θεωρείται ασφαλής τρόπος προγραμματισμού, και είναι ο ενδεδειγμένος.

Άρα, διακρίνουμε 3 είδη μεθόδων:

**1) Operator methods (λειτουργικές):**

```
public void opMethod()
{
     //δηλώσεις;
     //υπολογισμοί;
}
```

(είναι πάντα `void`, δεν επιστρέφουν τίποτα, απλά υπολογίζουν κάτι)

**2) Modifier methods (τροποποιητικές)**

```
private double var1;

public void modMethod(double var2)
{
   var1 = var2;
}
```

(δέχονται πάντα κάποια παράμετρο (π.χ. `var2`), την τιμή της οποίας «περνάνε» σε κάποια `private` μεταβλητή της κλάσης τους (π.χ. `var1`) )

**3) Accessor methods (πρόσβασης)**

```
private int var3;

public int accessMethod()
{
   return var3;
}
```

(είναι πάντα `public`, επιστρέφουν (`return`) μια `private` μεταβλητή και είναι του ίδιου τύπου με τη μεταβλητή που επιστρέφουν – στο παραπάνω παράδειγμα `int`)

*Συχνά υπάρχουν μέθοδοι που είναι συνδυασμοί των παραπάνω τύπων, κυρίως των τύπων (2) και (3).*

# Σημειώσεις JAVA – 2η εβδομάδα

**Οι 5 βασικές λειτουργικές έννοιες της αντικειμενοστραφούς φιλοσοφίας της JAVA:**

(υπάρχουν και άλλες βασικές έννοιες, για τις οποίες θα μιλήσουμε αργότερα...)

1) Ορισμός Κλάσης:

```
public class Name
{
   // μεταβλητές
   // μέθοδοι
}
```

2) Ορισμός μεταβλητής:

```
[ορατότητα] <τύπος> <όνομα> [=τιμή];
  public     int     var1    = 5;
 private   double   var2 ;
  public   String   var3 = "Hello.";
```

3) Ορισμός μεθόδου:

```
[ορατότητα] <τύπος> <όνομα> ( [παράμετροι] )
{
  ...
}
```

**π.χ.**

```
 private     void   method1()
{
   ...
}

 public    double  method2()
{
   return doubleVar;
}

private int a;
private double b;

public void method3(int x, double b)
{
   a = x;
   this.b = b;  // το this. «δείχνει» τη μεταβλητή της κλάσης
}
```

4) Δημιουργία αντικειμένου (στιγμιότυπου) κλάσης:

```
<Κλάση> <αντικείμενο> = new <Κλάση> ( );
```

```
 Circle     c1        = new  Circle();

  Dog      ivan       = new  Dog();
```

5) Κλήση μεθόδου αντικειμένου

ΠΡΟΣΟΧΗ: *Όχι* «κλήση μεθόδου *κλάσης*». Η κλάση είναι «καλούπι» για αντικείμενα. Οι μέθοδοι μιας κλάσης ανήκουν στην ουσία *στα αντικείμενά της*.

**<αντικείμενο>.<μέθοδος>( );**

π.χ.
Αν έχουμε μια κλάση υποστήριξης ως εξής:

```
public class Dog
{
  private String color;

  public void setColor(String a)
  {
    color = a;
  }

  public String accessColor()
  {
    return color;
  }
}
```

τότε από κάποια άλλη κλάση υποστήριξης ή από την κλάση εφαρμογής, μπορούμε να φτιάξουμε αντικείμενα της κλάσης Dog, να στείλουμε τα χρώματα των αντικειμένων αυτών, ή από κάπου αλλού να «διαβάσουμε» τα χρώματά τους:

```
// δημιουργία αντικειμένων της κλάσης Dog
Dog ivan = new Dog();
Dog voula = new Dog();


// αποστολή του χρώματος του κάθε αντικειμένου
ivan.setColor(black);
voula.setColor(white);

// Όταν κάπου αλλού θέλουμε να ανακτήσουμε τα χρώματα
// των αντικειμένων που έχουμε φτιάξει:

String myDogColor;     // το χρώμα του σκύλου μου
String nikosDogColor;  // το χρώμα του σκύλου του Νίκου
```

```
// αν ο σκύλος μου είναι ο Ivan και ο σκύλος του Νίκου η
Βούλα:

myDogColor = ivan.accessColor();        // επιστρέφει black
nikosDogColor = voula.accessColor();    // επιστρέφει white
```

Δηλαδή, η μεταβλητή `color` της κλάσης `Dog`, στην ουσία δεν έχει κάποια συγκεκριμένη τιμή. Συγκεκριμενοποιείται μόνο όταν φτιάξουμε κάποιο αντικείμενο της κλάσης Dog και έχει τόσα «αντίγραφα» όσα είναι τα αντικείμενα. Αφού συγκεκριμενοποιηθεί από τη δημιουργία αντικειμένου, παίρνει συγκεκριμένη τιμή όταν καλέσουμε τη μέθοδο `setColor`.

---

Primitive Types (πρωτογενείς τύποι μεταβλητών)

byte -128 ≤ ακέραιος ≤ 127
short -32768 ≤ ακέραιος ≤ 32767
int $-2^{31} \leq$ ακέραιος $\leq 2^{31}-1$
long $-2^{63} \leq$ ακέραιος $\leq 2^{63}-1$
float δεκαδικός – ακρίβεια: $\pm 10^{-46}$ – εύρος: $\pm 10^{38}$
double δεκαδικός – ακρίβεια: $\pm 10^{-324}$ – εύρος: $\leq \pm 10^{308}$
char χαρακτήρας unicode
boolean false, true

π.χ.
```
int a = 18743;
double b = 14.3217;
char c = 'A';
Boolean d = true;
```

Το "=" δεν είναι το μαθηματικό "=". Είναι μεταβίβαση της τιμής στα δεξιά του στη μεταβλητή στα αριστερά του.

Άρα, τα ακόλουθα είναι σωστά στη JAVA:

```
a = a + 5;
```

```
x = 1; y = 2;
x = y;
```

Τύπος String (κλάση)

```
public String x = "Αύριο είναι ";
public String y = "Σαββάτο";

System.out.println(x+y)  // --> Αύριο είναι Σαββάτο


public String a = "10";
```

```
public String b = "20";

System.out.println(x+y)   // --> 1020
```

Μετατροπή τύπων:

```
doubleVar = (double) intVar;
intVar = (int) doubleVar;
```

Τελεστές πράξεων:

+, -, *, /. %

3*4 -->  12,  3/4 --> 0 (ακέραιοι!),  3%4 --> 3,  11%3 --> 2

Μαθηματικές συναρτήσεις:

Math.sin(x) --> ημ(x),  Math.cos(x) --> συν(x),  Math.exp(x) --> $e^x$,  Math.pow(x,y) --> $x^y$

Math.PI --> π

---

```
a+=x; --> a=a+x;
a-=x; --> a=a-x;
a*=x; --> a=a*x;
a/=x; --> a=a/x;

a++; }
     } --> a=a+1;
++a; }

a--; }
     } --> a=a-1;
--a; }
```

π.χ.

```
int a, b, c;
a=2;

a+=6;     --> a=8
a-=3;     --> a=5
a*=2;     --> a=10
b = 2 * a++;   --> a=11, b=20
c = 2 * ++a;   --> a=12, b=24
```

Παράδειγμα μετατροπής τύπων:

```
int a = 73, b = 10;
double c, d;

c = a / b;    --> c=7.0
d = (double) a / b;   --> d=7.3
```

Σύγκριση μεταβλητών:

```
a==b  a<b  a>b   |
                 |--> boolean
a!=b  a<=b a>=b  |
```

```
(5==5) --> true
(5==3) --> false

boolean isRefrigLightOn;
boolean isRefrigDoorOpen;
isRefrigLightOn = (isRefrigDoorOpen==true);
```

Λογικές σχέσεις:

```
A&&B, A&B  AND  (το && σταματάει αν η πρώτη έκφραση είναι false)
A||B, A|B  OR   (το || σταματάει αν η πρώτη έκφραση είναι true)
!A         NOT

(3==7) && (2==(3/0)) --> false
(3==7) & (2==(3/0))  --> σφάλμα στο compilation (division by zero)
```

Επικοινωνία με το χρήστη

Μέσω διαλόγων (dialog boxes)

--> `import javax.swing.*;`

Χρήση της class: JOptionPane

Μέθοδοι: 1) showInputDialog --> ζητάει εισαγωγή δεδομένων (input)
2) showMessageDialog --> εξαγωγή αποτελεσμάτων (output)
3) showConfirmDialog --> επιλογή περιπτώσεων (Yes/No/Cancel)

1)
```
String s;
  s = JOptionPane.showInputDialog(<μήνυμα>)
```

π.χ. `String s = JOptionPane.showInpurDialog("Δώσε την ηλικία σου");`

(επιστρέφει την είσοδο σαν `String`. Άρα, αν η είσοδος είναι νούμερο, πρέπει να το μετατρέψουμε:

--> Πώς μετατρέπουμε String σε int και double?

- Σε int:
```
int age;
age = Integer.parseInt(s);     // s είναι το string που πήρε ο showInputDialog
```

- Σε double:
```
double age;
age = Double.parseDouble(s);
```

2) `JOptionPane.showMessageDialog(null,"……");`

3)
```
int i;
i = JOptionPane.showConfirmDialog(null,"……")
      YES --> 0, NO --> 1, CANCEL --> 2
```

ή

`i=JOptionPane.showConfirmDialog(null,"question","box title", x)`

όπου το `x` επιλέγει τον τύπο του παραθύρου και μπορεί να είναι 0, 1 ή 2, ως εξής:

```
0 --> YES/NO
1 --> YES/NO/CANCEL
2 --> OK/CANCEL
```

# Σημειώσεις JAVA – 3η εβδομάδα

## Constructors (κατασκευαστές ή δημιουργοί)

Ειδικός τύπος μεθόδων, οι οποίες:

- Είναι `public` και έχουν το ίδιο όνομα με αυτό της κλάσης
- χρησιμοποιούνται για να αρχικοποιήσουν κάποιες μεταβλητές των αντικειμένων που δημιουργούν
- καλούνται αυτόματα όταν αρχικοποιούμε ένα αντικείμενο κάποιας κλάσης
- δεν επιστρέφουν κάποια τιμή

Σε μία κλάση με το όνομα `ClassName`:

```
public class ClassName
{
   private int a;
   private String b;

     // default constructor:
   public ClassName()
   {

   }

     // another constructor:
   public ClassName(int x, String s)
   {
      a = x;
      b = s;
   }

   // other statements, methods, etc.

}
```

Παράδειγμα constructors στην κλάση `Circle`:

```
public class Circle
{

   private double x, y;  // συντεταγμένες κέντρου
   private double r;     // ακτίνα


      // μέθοδος για υπολογισμό εμβαδού κύκλου
   public double area()
   {
      return Math.PI*Math.pow(r,2);
   }

      // μέθοδος για υπολογισμό περιφέρειας κύκλου
   public double circumf()
   {
      return 2*Math.PI*r;
   }

}
```

Οι μεταβλητές `x`, `y` και `r` είναι `private`. Ένας τρόπος για να τους δώσουμε τιμές θα ήταν η ύπαρξη αντίστοιχων modifier μεθόδων στην κλάση `Circle`. Επειδή όμως, κάθε αντικείμενο («κύκλος») που θα δημιουργούμε, αναγκαστικά θα πρέπει να έχει τουλάχιστον κάποια δεδομένη ακτίνα, μπορούμε να αυτοματοποιήσουμε την απόδοση τιμών σε (κάποιες από) αυτές τις μεταβλητές, με την δημιουργία κάποιων constructors:

```
public Circle(double a, double b, double c)
{
   x = a;
   y = b;
   r = c;
}

public Circle(double r)
{
   x = 0; y = 0; this.r = r;
}

public Circle()
{
   x = 0; y = 0; r = 1;
}
```

Ο πρώτος constructor αρχικοποιεί όλες τις μεταβλητές, ο δεύτερος αρχικοποιεί μόνο την ακτίνα, ενώ ο τρίτος είναι παραλλαγή του default constructor και δημιουργεί «μοναδιαίους κύκλους» (αντικείμενα με r = 1).

Έτσι, μπορούμε να έχουμε τα εξής statements για τη δημιουργία αντικειμένων της κλάσης Class:

```
Circle circle1 = new Circle(2.5, 3.5, 6);

Circle circle2 = new Circle(4.3);

Circle circle3 = new Circle();

~~Circle circle4 = new Circle(1.3, 4.5);~~
```

→ ΛΑΘΟΣ! Δεν έχουμε φτιάξει constructor που να δέχεται δύο παραμέτρους...

# ΕΛΕΓΧΟΣ ΡΟΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

**I. Ελεγκτές συνθηκών ή περιπτώσεων:**

**i) `if/else`**

**ii) `switch`**

**II. Επαναληπτικές διαδικασίες:**

**i) `for`**

**ii) `while`**

**iii) `do/while`**

## - if/else:

```
if (συνθήκη)
{
   statements; // if true
}
else
{
   statements; // if false
}
```

```
if (συνθήκη)
{
   statements;
}
else if(συνθήκη)
{
   statements;
}
else if(συνθήκη)
{
   statements;
}
else
{
   statements;
}
```

→ Η συνθήκη είναι πάντα `boolean` έκφραση.

Σε περίπτωση απόδοσης τιμής (`x=y`), το if/else μπορεί να γραφτεί σε συνοπτική μορφή:

```
if (x > y)
  max = x;
else                  →   max = (x > y) ? x : y;
  max = y;
```

π.χ.

```
if (grade>90)          // πάνω από 90 -> A
  letterGrade = 'A';
else if (grade>80)     // από 81 εως 90 -> B
  letterGrade = 'B';
else if (grade>70)     // από 71 εως 80 -> C
  letterGrade = 'C';
else if (grade>60)     // από 61 εως 70 -> D
  letterGrade = 'D';
else                   // από 60 και κάτω -> F
  letterGrade = 'F';
```

**- switch:**

```
switch (varName)
{
  case value1:
  {
     statements;
     break;  // ή return
  }

  case value2:
  {
     statements;
     break;
  }

  case value3:
  {
     statements;
     break;
  }

  default:
  {
     statements;
  }

}
```

→ Η μεταβλητή `VarName` είναι είτε ακέραια, είτε τύπου `boolean`, είτε τύπου `char`.

Το `switch` «στέλνει» τη ροή του κώδικα στο ισχύον case. Αν δεν υπάρχει κάποιο `break` (ή `return`) στο τέλος ενός case, τότε η ροή του κώδικα συνεχίζει στο επόμενο case!

π.χ.

```
switch (letterGrade)
{
  case 'A':
  {
     System.out.println("O vathmos einai megalyteros tou 90.");
  break;
  }
  case 'B':
  {
     System.out.println("O vathmos einai metaksi 81 kai 90.");
  break;
  }
  case 'C':
  {
     System.out.println("O vathmos einai metaksi 71 kai 80.");
  break;
  }
  case 'D':
  {
     System.out.println("O vathmos einai metaksi 61 kai 70.");
  break;
  }
  default:
  {
     System.out.println("O vathmos einai apo 60 kai katw...");
  }
}
```

Παράδειγμα με παράλειψη κάποιου break:

```
switch (letterGrade)
{
  case 'A':
  {
     System.out.println("Poly kala!!!");
  }
  case 'B':
  {
     System.out.println("Bravo!");
  break;
  }
  case 'C':
  {
     System.out.println("Oxi ki asxima.");
```

```
  break;
  }
  case 'D':
  {
     System.out.println("Etsi ki etsi...");
  break;
  }
  default:
  {
     System.out.println("Wx.........");
  }
}
```

Το `break` στο "`case 'A'`" λείπει. Άρα, σε περίπτωση που ο βαθμός είναι **A**, θα εμφανιστεί:

```
Poly kala!!!
Bravo!
```

δηλαδή θα εκτελεστεί ***και*** το "`case 'B'`", ενώ αν ο βαθμός είναι **B**, θα εμφανιστεί μόνο το:

```
Bravo!
```

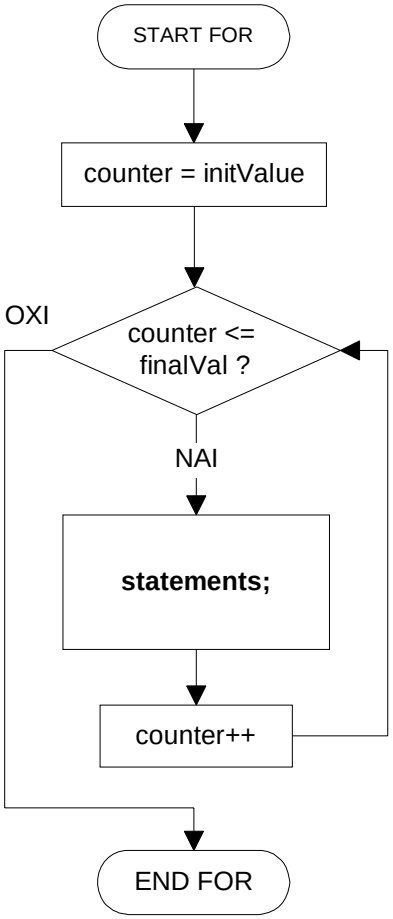
## - for:

```
for (counter=initVal; counter<=finalVal; counter++)
{
  statements;
}
```

Σειρά εκτέλεσης εντολών:

```
i)   counter = initVal;
ii)  έλεγχος counter<=finalVal ?
iii) statements;
iv)  counter++
v)   έλεγχος κτλ.
```

**<u>Διάγραμμα Ροής του for-loop</u>**




→ Συνήθως ο μετρητής (counter) ορίζεται τοπικά, στο πρώτο μέρος της παρένθεσης του for-loop, δηλαδή είναι μια τοπική μεταβλητή (με εμβέλεια (scope) μόνο το for-loop). Παράδειγμα:

```
for (int i=0; i<5; i++)
{
   System.out.println(i);
}
```

Τυπώνει:

```
0
1
2
3
4
```

Άλλο ένα παράδειγμα: Πρόγραμμα υπολογισμού του ν!

(είμαστε μέσα σε μια κλάση και έχει γίνει import το javax.swing.*)

```
private int n;
private int factorial = 1;


// methodos pou ypologizei to n!, opou to n eisagetai apo ton
xristi

public int fact()
{
   String s = JOptionPane.showInputDialog("Dwse enan akeraio");
   n = Integer.ParseInt(s);

   for (int i=1; i<=n; i++)
   {
      factorial *= i;
   }

   JOptionPane.showMessageDialog("To " + n + " paragontiko" +
      "einai iso me " + factorial);
}
```

**Nested for-loops:**

π.χ.

```
for (int i=1; i<=5; i++)
{
   //statements_1;

   for (int j=1; j=<3; j++)
   {
     //statements_2;
   }

   // statements_3;
}
```

Σειρά εκτέλεσης:
Μία φορά τα "statements_1"
Τρεις φορές τα "statements_2"
Μία φορά τα "statements_3"
Δεύτερη φορά τα "statements_1"
Άλλες τρεις φορές τα "statements_2"
*κτλ.*

**- while:**

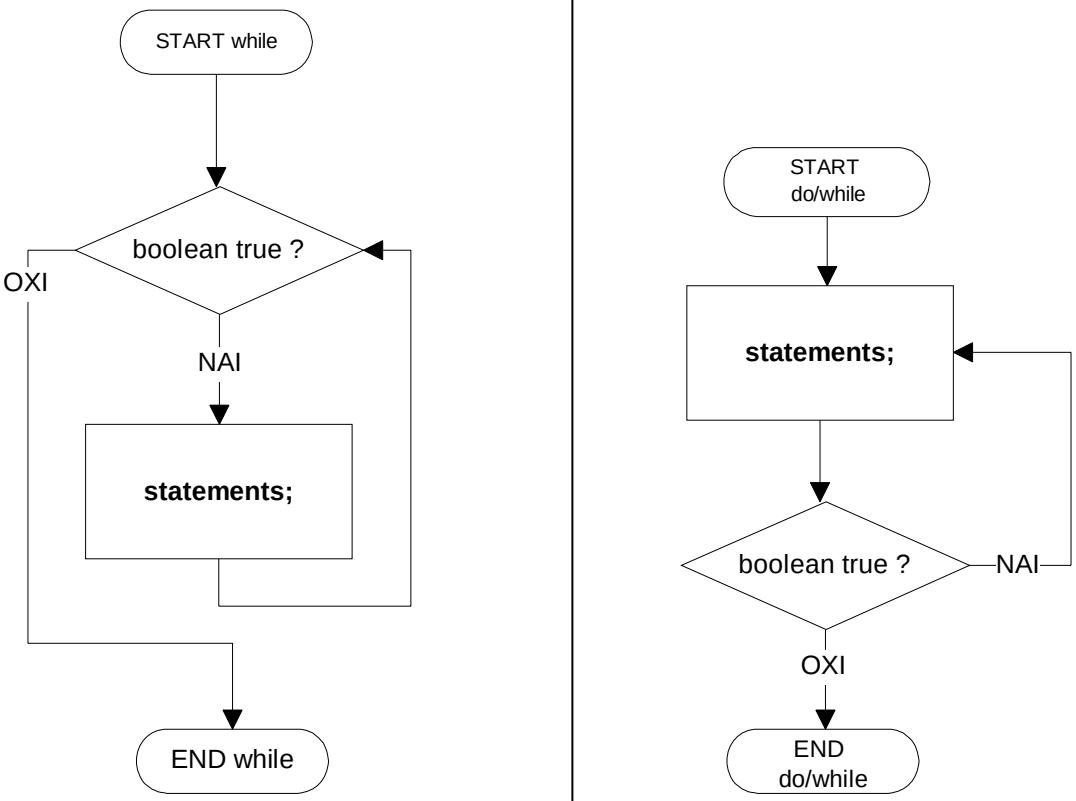
```
while(συνθήκη)
{
  statements;
}
```

**- do/while**

```
do
{
   statements;
}
 while(συνθήκη);
```

Η συνθήκη είναι ένα `boolean` expression (άρα: `true` ή `false`)
Κάνουν επανάληψη των statements καθ' όσον η συνθήκη είναι αληθής.
Η διαφορά μεταξύ τους είναι ότι το `do/while` θα εκτελέσει τα statements *τουλάχιστον μία φορά*, ανεξάρτητα με το αν είναι `true` η συνθήκη, αφού την ελέγχει στο τέλος του loop, ενώ η `while` δεν μπαίνει καθόλου στο loop αν αρχικά η συνθήκη είναι `false`.

## Διαγράμματα Ροής του `while` και του `do/while`




Παράδειγμα επιλογής while ή do/while:

Θέλουμε να ζητήσουμε από τον χρήστη μια τιμή για τη μεταβλητή `x` η οποία να είναι μεγαλύτερη ή ίση του μηδενός. Αν ο χρήστης δίνει αρνητική τιμή, τότε η ερώτηση θα επαναλαμβάνεται μέχρι να δοθεί τιμή μεγαλύτερη ή ίση του μηδενός:

```
private double x;

public void askUser()
{
  while(x<0) // do
```

```
  {
    String s = JOptionPane.showInputDialog("Dwse mi arnitiki timi
gia to x");
    x = Double.parseDouble(s);
  }
   // while(x<0);

}
```

Όπως είναι γραμμένος ο κώδικας, θα χρησιμοποιήσουμε είτε τον κόκκινο κώδικα (`while`) είτε τον μπλε (`do/while`) ο οποίος είναι σαν comments εδώ. Ποιο από τα δύο θα επιλέξουμε;
Η επιλογή του `while` είναι ΛΑΘΟΣ, γιατί το `x` είναι ήδη 0, άρα η ροή του κώδικα δεν θα μπει ποτέ μέσα στις αγκύλες του `while`, αφού το (`x<0`) είναι αρχικά `false`! Άρα δεν θα ζητηθεί ποτέ από τον χρήστη να εισάγει κάποια τιμή για το `x`... Το σωστό είναι να χρησιμοποιήσουμε το `do/while`.

# Σημειώσεις JAVA – 4η εβδομάδα

**Ανακεφαλαίωση:**

- Σύνταξη → δεσμευμένες λέξεις, identifiers (ονόματα)

- Ορισμός Κλάσης
  `public class ClassName` → save σε αρχείο ClassName.java

- Η Κλάση έχει:
  - μεταβλητές →
    ```
    private int varName;
    public String varName2;
    ```
  - constructors → `public ClassName(...)`
  - μεθόδους →
    ```
    public void methodName(...)
    private int methodName2()
    ```

- Είδη μεθόδων:

  i) operator →
  ```
  public void methodName1()
  {
     statements;
  }
  ```

  ii) modifier →
  ```
  private double var1;
  public void methodName2(double v1)
  {
     var1 = v1;
  }
  ```

  iii) accessor →
  ```
  private boolean var2;
  public boolean methodName3()
  {
     return var2;
  }
  ```

- Application Class → main method

  Support Classes → «καλούπια» για δημιουργία αντικειμένων

  ```
  Circle c1 = new Circle();
  Circle c2 = new Circle(4,3,2);
  ```

- Κλήση μεθόδων των αντικειμένων:

  ```
  double a;
  ```

```
a = c2.area();
```

- Πρωτογενείς (primitive) τύποι μεταβλητών
- Μαθηματικές πράξεις, λογικές σχέσεις, συγκρίσεις μεταβλητών, μετατροπές τύπων, εμβέλεια (scope) μεταβλητών
- If / else
- Switch
- For-loop
- while – do / while

---

## ΑΣΚΗΣΕΙΣ – ΠΑΡΑΔΕΙΓΜΑΤΑ

Ποια είναι η τιμή της μεταβλητής `var4`;

```
double var1 = 4.3;
double var2 = 2.7;
int var3 = 4;
int var4;
var4 = (((int)var1)*(int)var2)%var3;
```

```
        (    4     *     2    ) % 4
                   8           % 4   → 0
```

---

Τι θα τυπώσουν τα ακόλουθα προγράμματα;

```
double w=75;
double p;
if (w>=35)
   p=20.0;
   if (w>=50)
      p=50;
      if (w>=75)
        p=100.0;
System.out.println(p);
```

Απάντηση: `100.0`

```
double w=75;
double p;
if (w>=35)
   p=20.0;
else if (w>=50)
   p=50;
else if (w>=75)
   p=100.0;
System.out.println(p);
```

Απάντηση: `20.0`

---

Ο παρακάτω κώδικας να γραφεί χρησιμοποιώντας το switch:

```
if (prefix == 'm') System.out.println("milli");
if (prefix == 'c') System.out.println("centi");
if (prefix == 'k') System.out.println("kilo");
```

Απάντηση:

```
switch(prefix)
{
  case 'm':
    {
       System.out.println("milli");
       break;
    }
  case 'c':
    {
       System.out.println("centi");
       break;
    }
  case 'k':
    {
       System.out.println("kilo");
       break;
    }
}
```

---

Τι τιμές έχουν τα s και n μετά την εκτέλεση των παρακάτω;

```
int s = 1;
int n = 1;
while (s<10)
{
  s+=n;
  n++;
}
```

Απάντηση: `s = 11`
`n = 5`

```
int s = 1;
int n = 1;
for (n=1; n<5; ++n)
{
  s-=n;
}
```

Απάντηση: `s = -9`
`n = 5`

```
int s = 1;
int n = 1;
do
{
  s+=n;
  n--;
} while(n>-2);
```

Απάντηση: `s = 1`
`n = -2`

---

Μια διευκρίνηση για την `System.out.println`:

```
double x = 10;
double y = 15;
```

```
System.out.println("Το άθροισμα είναι " + x + y);
System.out.println(x + y);
System.out.println("Το άθροισμα είναι " + (x + y));
```

Αυτό θα τυπώσει στην οθόνη:

```
Το άθροισμα είναι 10.015.0
25.0
Το άθροισμα είναι 25.0
```

---

Τι θα τυπώσουν τα ακόλουθα τμήματα κώδικα (το S.O.P. χρησιμοποιείται αντί του System.out.println για συντομία – δεν είναι εντολή της JAVA);

```
int x = 9;
int y = 11;
if (x<10)
if (y>10)
S.O.P.(x+y);
else
S.O.P.(x-y);
S.O.P.(x%y);
```

Απάντηση: 20
9

```
int x = 11;
int y = 9;
if (x<10)
if (y>10)
S.O.P.(x+y);
else
S.O.P.(x-y);
S.O.P.(x%y);
```

Απάντηση: 2

```
int x = 9;
int y = 11;
if (x<10)
{
if (y>10)
S.O.P.(x+y);
}
else
{
S.O.P.(x-y);
S.O.P.(x%y);
}
```

Απάντηση: 20

```
int x = 11;
int y = 9;
if (x<10)
{
if (y>10)
S.O.P.(x+y);
}
else
{
S.O.P.(x-y);
S.O.P.(x%y);
}
```

Απάντηση: 2
2

---

Να γίνει System.out.println των περιττών μεταξύ 10 και 100 με δύο τρόπους: α) με for-loop και β) με while.

Απάντηση:

α)

```
for (int i=11; i<100; i+=2)
   System.out.println(i);
```

β)

```
int i=11;
while (i<100)
{
  System.out.println(i);
  i+=2;
```

```
}
```

---

Τι θα τυπωθεί στην οθόνη όταν γίνει κλήση της μεθόδου `methodName();`

```
private int n=4;
.
.
.
public void methodName()
{
   for (int n=0; n<4; ++n)
   {
      do
      {
         while (n<3)
         {
            System.out.println("n = " + n++);
         }
         n+=1;
      }
      while (n<10);
   }
   System.out.println("Final value of n = " + n);
}
```

Απάντηση:
```
n = 0
n = 1
n = 2
Final value of n = 4
```

---

Nested for-loops:
Τι θα τυπώσει στην οθόνη ο παρακάτω κώδικας;

```
int k;
for (int i=0; i<3; i++)
{
   k=0;
   for (int j=0; j<5; j++)
   {
      k = k + j + i;
   }
   System.out.println(k);
}
```

Απάντηση:
```
10
15
20
```

---

# Σημειώσεις JAVA – 5η εβδομάδα

## Η Κλάση `String`

Κάθε αλφαριθμητικό (string) είναι αντικείμενο της κλάσης `String`.

Βασική ιδιότητα: Τα strings, δηλ. τα αντικείμενα της κλάσης `String`, δεν μπορούν να τροποποιηθούν.

Τρόποι δημιουργίας strings:

i) Με τη χρήση εισαγωγικών: `"Hello "`

ii) Με τη χρήση των + ή += πάνω σε υπάρχοντα strings
`"Hello " + "World"` --> νέο string --> `Hello World`

iii) Με κανονική δημιουργία αντικειμένου μέσω της `new`.

Η string έχει δύο constructors:

`public String()` και `public String(String value)`

π.χ.
```
String str1 = new String();
String str2 = new String(a);  // a είναι ένα υπάρχον String
```

Βασικές μέθοδοι της κλάσης `String`

- `public int length()` --> επιστρέφει το πλήθος των χαρακτήρων του αλφ/κού

π.χ.
```
String a = "blah blah";
int b = a.length();   // --> b = 9
```

- `public char charAt(int index)` --> επιστρέφει τον χαρακτήρα στο σημείο index

```
char d = a.charAt(3);  // --> d = 'h'
```
*(μετράμε από το 0!)*

---

- `public int indexOf(char ch)` --> επιστρέφει την πρώτη θέση του `ch`

```
a.indexOf('a');  --> 2
```

- `public int indexOf(char ch, int start)` -->επιστρέφει την πρώτη θέση του `ch`
από τη θέση `start` και μετά.

```
a.indexOf('a',3);  --> 7
```

- `public int indexOf(String str)` --> επιστρέφει την πρώτη θέση του `str`

- `public int indexOf(String str, int start)`-->επιστρέφει την πρώτη θέση του `str`
από τη θέση `start` και μετά.

- `public int lastIndexOf(char ch)` --> επιστρέφει την τελευταία θέση του `ch`

- `public int lastIndexOf(char ch, int start)` --> επιστρέφει την τελευταία
μέχρι και το `start` θέση του `ch`

- `...`[αντίστοιχα και για παράμετρο `String str`].

Όλες αυτές οι μέθοδοι αν δεν βρουν αυτό που ψάχνουν στο string (το `ch` ή το `str`), επιστρέφουν **`-1`**.

---

Μέθοδοι σύγκρισης

- `public boolean equals(String str)`
  - `--> true,` αν βρει ίδιο μήκος και ακριβώς ίδιους χαρακτήρες
  - `--> false,` σε διαφορετική περίπτωση

π.χ.
```
String e = "blah blah";
boolean f = a.equals(e);  // --> f = true
```

***ΠΡΟΣΟΧΗ***!
```
boolean g = (a==e); // --> g = false
```

Η `equals` ελέγχει για **ισότητα**.
Η == ελέγχει για **ταύτιση**!

Τα `a` και `e` είναι *ίσα* (ίδιο μήκος και ίδιοι χαρακτήρες) αλλά είναι *διαφορετικά αντικείμενα*, άρα δεν ταυτίζονται!

- `public boolean equalsIgnoreCase(String str)`
  Αγνοεί κεφαλαία ή μικρά γράμματα

- `public int compareTo(String str)`
  - `--> < 0` αν κάποιο string έχει < μήκος του `str`
  - `--> = 0` αν κάποιο string έχει = μήκος με το `str`
  - `--> > 0` αν κάποιο string έχει > μήκος του `str`

```
String h = "blah";
int i = h.compareTo(a);  // --> i<0
```

```
- public boolean regionMatches(int start, String str,
                               int strStart, int len)

          h.regionMatches(1,a,6,3);  // --> true

               h --> b l a h
                     0 1 2 3

               a --> b l a h   b l a h
                     0 1 2 3 4 5 6 7 8

     a.regionMatches(2,"Ah",0,2);  // --> false  (ah != Ah)

- public boolean regionMatches(Boolean ignoreCase, int start,
                               String str, int strStart, int len)

          a.regionMatches(true,2,"Ah",0,2);  // --> true
```

---

Μέθοδοι ελέγχου αρχής/τέλους:

```
- public boolean startsWith(String prefix)

- public boolean startsWith(String prefix, int offset)

- public boolean endsWith(String suffix)

          a.endsWith("ah")  --> true
```

---

Μέθοδοι δημιουργίας νέων αντικειμένων String

```
- public String replace(char oldChar, char newChar)

          String j = a.replace('a','i');  // --> blih blih
```

**Δεν** αλλάζει το string a. Φτιάχνει **<u>νέο</u>** string (αντικείμενο)

```
- public String toLowerCase()
```

π.χ.
Στην περίπτωση απλών αντικειμένων π.χ. μιας κλάσης υποστήριξης SuppClass, θα μπορούσαμε να έχουμε τον εξής κώδικα σε μια άλλη κλάση:

```
// Dimiourgia antikeimenou tis SuppClass
SuppClass obj1 = new SuppClass();

// Apodosi timis sti metavliti x tou obj1
obj1.x = 10;
```

```
// Dimiourgia antigrafou tou antikeimenou obj1
SuppClass obj2 = obj1;

// Apodosi timis sti metavliti x tou obj2
obj2.x = 5;

// Ektypwsi stin othoni tou x tou obj1
System.out.println(obj1.x);
```

Ο κώδικας αυτός θα εκτυπώσει την τιμή **5**.

Στην ουσία έχουμε **ένα** αντικείμενο της `SuppClass`, το οποίο έχει δύο ονόματα, `obj1` και `obj2`. Οπότε, αλλάζοντας την τιμή του `x` στο `obj2`, στην ουσία αλλάζει η τιμή και στο `obj1`.

Αν τώρα έχουμε έναν αντίστοιχο κώδικα, αλλά με αντικείμενα της κλάσης `String` αντί κάποιας άλλης κλάσης:

```
// Dimiourgia antikeimenwn "typou" String
String k, m;

// Apodosi timis sto antikeimeno k
k = "Hello";

// Dimiourgia antigrafou tou antikeimenou k
m = k;

// "Tropopoiisi" tou antikeimenou m
m.toLowerCase();

// Ektypwsi stin othoni tou antikeimenou k
System.out.println(k);
```

Ο κώδικας αυτός θα εκτυπώσει **`Hello`** και όχι `hello` όπως θα περίμενε κάποιος, σε αντιστοιχία με το προηγούμενο παράδειγμα με την `SuppClass`.
Αυτό συμβαίνει γιατί κατά την εκτέλεση της εντολής `m.toLowerCase()` δεν αλλάζει η τιμή του `m` και άρα και του `k`, αλλά δημιουργείται ένα **νέο** αντικείμενο, στο οποίο «βλέπει» πλέον το αντικείμενο `m`. Άρα, το αρχικό αντικείμενο `k` μένει αμετάβλητο (`Hello`), το `m` παύει να υπάρχει ως αντίγραφο του `k` και δημιουργείται ένα νέο αντικείμενο (`hello`) με το όνομα `m`.
Τα αντικείμενα τύπου `String` λοιπόν, δεν μπορεί να μεταβληθούν. Κάθε «τροποποίηση» επιστρέφει ένα <u>νέο αντικείμενο</u> τύπου `String`.

- `public String toUpperCase()`

- `public String trim()` --> κόβει τα κενά σε αρχή και τέλος

- public String concat(String str) --> το ίδιο με το +

```
// Apo prin: a=blah blah kai h=blah
a.concat(h); // -->  blah blahblah (ίδιο με το a+h;)
```

---

Μετατροπές απο/σε String

Από μεταβλητή τύπου `boolean`, `int`, `long`, `float`, `double` σε `String`:

```
String().valueOf(<μεταβλητή>);
```

```
int n = 10;
String p = String().valueOf(n);  // --> p = "10"
```

Από `String` σε `boolean` --> `new Boolean(str).booleanValue()`
`int` --> `Integer.ParseInt(str)`
`long` --> `Long.ParseLong(str)`
`float` --> `new Float(str).floatValue()`
`double` --> `new Double(str).doubleValue()`

---

Ο χαρακτήρας \ χρησιμοποιείται σε ειδικές περιπτώσεις, όπως για την εκτύπωση των εισαγωγικών:

```
System.out.println("Say \"Hi\"!"); --> τυπώνει: Say "Hi"!
```

(το παραπάνω string έχει 9 χαρακτήρες και όχι 11. Το \ δεν μετράει σαν χαρακτήρας)

Για εκτύπωση του \: \\

```
System.out.println("abc\\def");  --> τυπώνει: abc\def
```

`\n` --> new line
`\t` --> tab

---

## Η Κλάση `StringBuffer`

Είναι η κλάση που τροποποιεί αλφαριθμητικά.

Constructors:
`public StringBuffer()` και `public StringBuffer(String str)`

Για τροποποίηση:

- public void setCharAt(int index, char newChar)

Το πρόβλημα με την `String`:

```
String name = title + " " + firstName + " " + lastName;
```

Η εντολή αυτή θα δημιουργήσει διαδοχικά 5 νέα αντικείμενα της κλάσης `String`, από τα οποία μας ενδιαφέρει μόνο το τελευταίο. Αντίθετα, η `StringBuffer` φτιάχνει μόνο ένα αντικείμενο:

```
StringBuffer name = new StringBuffer().append(title).
                    append(" ").append(firstName).
                    append(" ").append(lastName);
```

Μετατροπές `String` / `StringBuffer`

```
public String changeString(String str)
{
   // metatropi tou String str se StringBuffer
   StringBuffer buffer = new StringBuffer(str);

   // tropopoiisi tou buffer
   // ...

   // metatropi tou StringBuffer se String kai epistrofi tou
   return buffer.toString();
}
```

---

Ένα αντικείμενο `StringBuffer` είναι διάνυσμα από `char` **μεταβλητού** μεγέθους.

- `public StringBuffer(int capacity)` --> constructor
- `public void ensureCapacity(int minimum)` --> διασφαλίζει μια ελάχιστη χωρητικότητα
- `public int capacity()` --> επιστρέφει την τρέχουσα χωρητικότητα

# Σημειώσεις JAVA – 6η εβδομάδα

## Πακέτα (Packages)

Τα πακέτα αποτελούν τρόπο ομαδοποίησης κλάσεων συναφούς λειτουργικότητας.

Ένα πακέτο λειτουργεί σαν βιβλιοθήκη κλάσεων που μπορούν να χρησιμοποιηθούν από ένα πρόγραμμα χωρίς να βρίσκονται στο directory του.

Κάθε πακέτο έχει ένα όνομα και όλες του οι κλάσεις βρίσκονται στο ίδιο directory, που έχει το ίδιο όνομα με το πακέτο.

Οι κλάσεις ενός πακέτου είναι κανονικές κλάσεις, άρα ορίζονται:

```
public class <ClassName>   // save σε αρχείο <ClassName>.java
```

αλλά **πριν** τον ορισμό τους (την επικεφαλίδα τους ή την ταυτότητά τους) έχουν τη δήλωση της μορφής:

```
package <packageName>;
```

---

π.χ.
Έστω ότι είμαστε μέσα στο directory με όνομα pack1, το οποίο έχει μέσα δύο αρχεία κλάσεων, ClassOne.java και ClassTwo.java:

Αρχείο ClassOne.java:

```
package pack1;

public class ClassOne
{
   ...
   ...
}
```

Αρχείο ClassTwo.java:

```
package pack1;

public class ClassTwo
{
   ...
   ...
}
```

Επίσης, έστω ότι στο directory pack1 υπάρχει sub-directory με το όνομα smallPack το οποίο περιέχει τις ακόλουθες κλάσεις:

Αρχείο Class1.java:

```
package pack1.smallPack;

public class Class1
{
   ...
   ...
}
```

Αρχείο Class2.java:

```
package pack1.smallPack;

public class Class2
{
   ...
   ...
}
```

Για να χρησιμοποιήσουμε στο πρόγραμμά μας τις κλάσεις του πακέτου pack1 χρησιμοποιούμε την εντολή **`import`**:

Αρχείο MyClass1.java:

```
import pack1.*;

public class MyClass1
{
   ...
   ...
}
```

ενώ ειδικά για τις κλάσεις του πακέτου smallPack:

Αρχείο MyClass2.java:

```
import pack1.smallPack.*;

public class MyClass1
{
   ...
   ...
}
```

Άρα, με την `import <packageName>.*;` μπορούμε να χρησιμοποιήσουμε τις κλάσεις πακέτων. Είναι κλάσεις που βρίσκονται εκτός του directory του προγράμματός μας.

Ορατότητα:

| Ορατότητα: | `public` | `private` | χωρίς δήλωση ορατότητας |
|---|---|---|---|
| Από την ίδια την κλάση | ναι | ναι | ναι |
| Από άλλη κλάση στο ίδιο πακέτο | ναι | όχι | ναι |
| Από άλλη κλάση έξω από το πακέτο | ναι | όχι | **όχι** |

δηλαδή,

- οι `public` μεταβλητές (ή μέθοδοι κτλ.) είναι ορατές από παντού
- οι `private` είναι ορατές μόνο μέσα στην κλάση όπου ορίζονται
- οι χωρίς δήλωση είναι ορατές από τις κλάσεις του πακέτου μόνο, δηλαδή αποτελούν κάτι ενδιάμεσο, μεταξύ `public` και `private`.

(Υπάρχει και το `protected` στο οποίο θα αναφερθούμε όταν μιλήσουμε για κληρονομικότητα)

---

Παράδειγμα:

Έχουμε ένα πακέτο (pack1) με τις ακόλουθες δύο κλάσεις:

```
package pack1;

public class Class1
{
  private void method1()
   { ... }

  public void method2()
   { ... }

  void method3()
   { ... }

  public void method4()
   { ... }
}
```

```
package pack1;

public class Class2
{

   Class1 obj = new Class1();

   private void method5()
   {
    obj.method1(); // ΛΑΘΟΣ!
    obj.method2();
    obj.method3();
   }

}
```

Στο πρόγραμμά μας κάνουμε χρήση του πακέτου `pack1`:

```
import pack1.*;

public class MyClass
{

  Class1 obj1 = new Class1();
```

```
  private void myMethod()
   {
    obj1.method1();  // ΛΑΘΟΣ!
    obj1.method2();
    obj1.method3();  // ΛΑΘΟΣ!!
    obj1.method4();
   }
}
```

---

--> Στο προηγούμενο παράδειγμα, θα μπορούσαμε αντί για

```
import pack1.*;
```

να είχαμε:

```
import pack1.Class1;
```

το οποίο θα έκανε `import` μόνο την `Class1` του πακέτου `pack1`.

--> Σε μια κλάση μπορεί να έχουμε και package declaration και import statement.

π.χ.

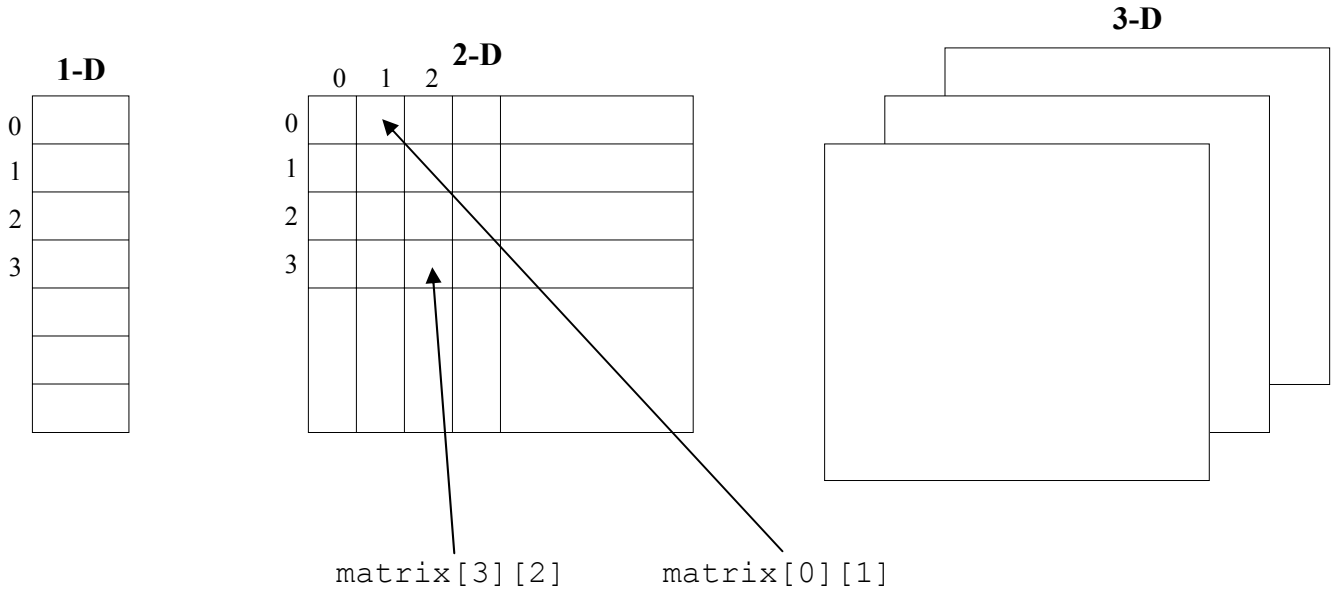
```
package pack2;
import javax.swing.*;

public class askUser
{
  // methods using JOptionPane
  ...
}
```

--> Αν υπάρχει κλάση με το ίδιο όνομα σε δύο διαφορετικά πακέτα, τότε αναφορές στο όνομα της κλάσης αυτής θα πρέπει να περιέχουν και το όνομα του πακέτου:

π.χ.
```
pack1.Class1 obj1 = new pack1.Class1();
pack2.Class1 obj2 = new pack2.Class1();
```

---

# Πίνακες (Arrays)




Γενική δήλωση πίνακα:

```
elementType [ ] arrayName = new elementType [size];  // --> 1-D
```

π.χ. `double [ ] temperature = new double [100];`

2-D: `elementType [ ][ ] arrayName = new elementType [size1][size2];`

Αρχικοποίηση σε συγκεκριμένες τιμές:

```
elementType [ ] name = {value1, value2, ...};
```

π.χ., `int [ ] daysInMonth = {31, 28, 31, 30, 31, ...};`

ή

```
int [ ] daysInMonth = new int [12];
daysInMonth[0] = 31; daysInMonth[1] = 28;
daysInMonth[2] = 31; daysInMonth[3] = 30; ...
```

Για 2-D:
π.χ.:

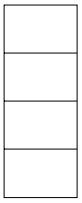
```
int [ ][ ] a = { {2, 3, 5}, {4, 2, 0}, {0, 2, 1} };
```

**ΠΡΟΣΟΧΗ:** Το `daysInMonth[12]` **δεν** υπάρχει!
12 είναι το μέγεθος του πίνακα. Το index πάει από 0 έως 11.

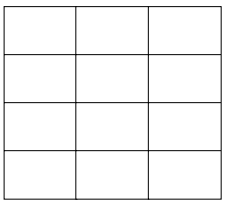
**<u>Λεπτομέρεια:</u>** Στην JAVA τυπικά υπάρχουν μόνο 1-D πίνακες! Όταν φτιάχνουμε έναν πίνακα 2-D, στην ουσία φτιάχνουμε έναν 1-D πίνακα για κάθε στοιχείο ενός αρχικού 1-D πίνακα. Δηλαδή, όταν φτιάχνουμε έναν 2-D πίνακα μεγέθους [4] x [3] με την εντολή:

```
int [ ][ ] array1 = new int [4][3];
```

στην αρχή δημιουργείται ένας 1-D πίνακας μεγέθους [4]:



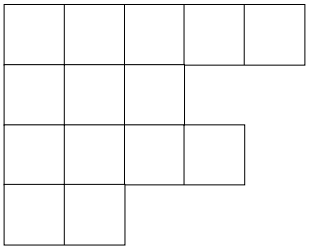
και στη συνέχεια κάθε στοιχείο του γίνεται ένας 1-D πίνακας μεγέθους [3], άρα φαινομενικά δημιουργείται ο 2-D πίνακας διαστάσεων [4] x [3]:



*Αυτό δίνει τη δυνατότητα στη JAVA να δημιουργήσει πίνακες μη-ορθογώνιους, δηλαδή πίνακες με διαφορετικό αριθμό στηλών ανά γραμμή:*

```
int [ ][ ] array2 = new int [4][ ];
array2[0] = new int [5];
array2[1] = new int [3];
array2[2] = new int [4];
array2[3] = new int [2];
```

Οι εντολές αυτές δημιουργούν τον ακόλουθο πίνακα:



Η length επιστρέφει το μέγεθος ενός πίνακα:

```
double [ ] a = new double [10];
int x = a.length;   // x=10
```

Οπότε, μια καλή μέθοδος αυτοματοποιημένης αρχικοποίησης ενός πίνακα: (παράδειγμα):

```
int [ ] b = new int [100];
for (int i=0; i<b.length; i++)
{
   b[i] = 1;
}
```

Στην περίπτωση 2-D πινάκων:

```
double [ ][ ] c = new double [20][10];
int rows = c.length;  // rows=20
int columns = c[0].length;  // columns=10
```

Το 0 στο `c[0].length` θα μπορούσε να είναι οποιοσδήποτε ακέραιος μεταξύ 0 και 9 εφόσον ο `c` είναι ορθογώνιος πίνακας (όλες η γραμμές έχουν τον ίδιο αριθμό στοιχείων).

---

Αντιγραφή πινάκων

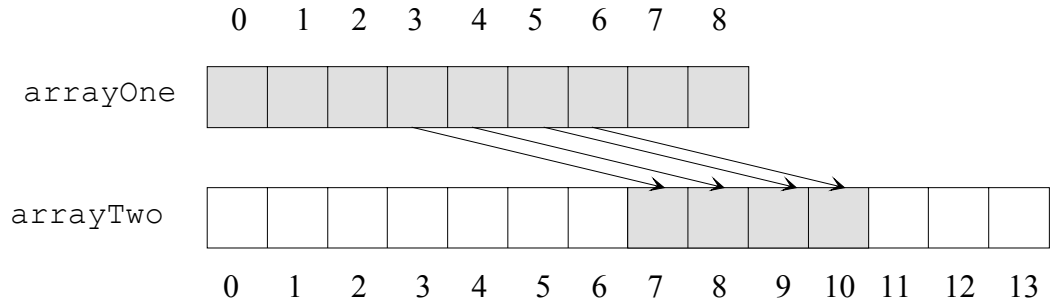
```
// dimiourgia pinakwn
int [ ] array1 = new int [5];
int [ ] array2 = new int [5];

// arxikopoiisi pinaka array1
for (int i=0; i<array1.length; i++)
{
   array1[i] = 2*I;   // array1 = {0,2,4,6,8}
}

// prospatheia antigrafis tou pinaka array1
// ston pinaka array2
array2 = array1;  // --> ίδιος χώρος στη μνήμη
                  // (ένας πίνακας με δύο ονόματα!)

// stin ousia auto den ekane antigrafi!
array2[0]=100;
System.out.println(array1[0]); // typwnei "100" kai oxi 0
!
// allazontas ton array2 allazei kai o array1.
// Logiko, afou prokeitai gia ENAN pinaka me dyo
onomata..
// O swstos tropos antigrafis tou array1 ston array2:

System.arrayCopy(array1, 0, array2, 0, array1.length);
```

Δηλαδή, για αντιγραφή πινάκων χρησιμοποιούμε την `System.arrayCopy`:

```
System.arrayCopy(sourceArray,
sourceArrayStartingPosition,
       destinationArray,
destinationArrayStartingPosition,
       howManyElementsToCopy);
```

όπου:
`sourceArray:` ο αρχικός πίνακας
`sourceArrayStartingPosition:` από ποιο σημείο να αρχίσει να παίρνει στοιχεία για αντιγραφή
`destinationArray:` ο τελικός πίνακας
`destinationArrayStartingPosition:` από ποιο σημείο να αρχίσει να αντιγράφει στον νέο πίνακα
`howManyElementsToCopy:` πόσα στοιχεία να αντιγραφούν

Άρα μπορούμε να αντιγράψουμε και τμήματα πινάκων:




```
System.arrayCopy(arrayOne,3,arrayTwo,7,4);
```

---

Αντί για στοιχεία πίνακα πρωτογενών τύπων (int, double, char, κτλ.), μπορεί να έχουμε και **στοιχεία αντικείμενα**.

Αν έχουμε μια κλάση υποστήριξης SuppClass, υπάρχουν δύο βήματα για τη δημιουργία αντικειμένων της κλάσης αυτής που θα αποτελούν στοιχεία πίνακα:

*i) δημιουργία πίνακα:*

```
SuppClass [ ] objects = new SuppClass [5];
```

Έτσι δημιουργείται ο πίνακας `objects` με μέγεθος 5, του οποίου τα στοιχεία είναι τύπου `SuppClass`, δηλαδή αντικείμενα της κλάσης `SuppClass`. Άρα, ο πίνακας objects θα περιέχει 5 αντικείμενα της κλάσης `SuppClass`.

*ii) δημιουργία αντικειμένων:*

```
for (int i=0; i<objects.length; i++)
{
   objects[i] = new objects();
}
```

Έτσι, δημιουργήθηκαν τα 5 αντικείμενα.

Οπότε, για κλήση π.χ. της μεθόδου `method1()` της `SuppClass` μέσω του αντικειμένου `objects[2]`:

```
objects[2].method1();
```

ή της `method3()` από κάποιο άλλο αντικείμενο, π.χ. το `objects[0]`:

```
objects[0].method3();
```

---

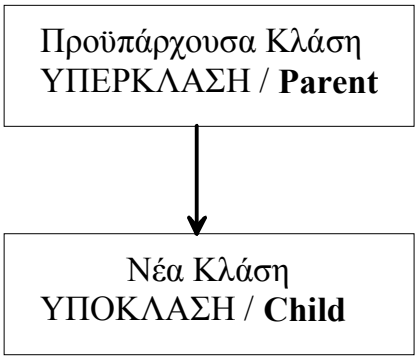
Accessor methods που επιστρέφουν πίνακες:

```
...

// dimiourgia enos private pinaka:

private double [ ] a = new double [10];

...

// accessor method pou epistrefei ton private pinaka

public double [ ] accessArray()
{
   return a;
}
```

# Σημειώσεις JAVA – 7η εβδομάδα

## Κληρονομικότητα (Inheritance)

Υπάρχουν κλάσεις που εμπεριέχουν δεδομένα και μεθόδους που έχουν οριστεί σε προϋπάρχουσες κλάσεις:



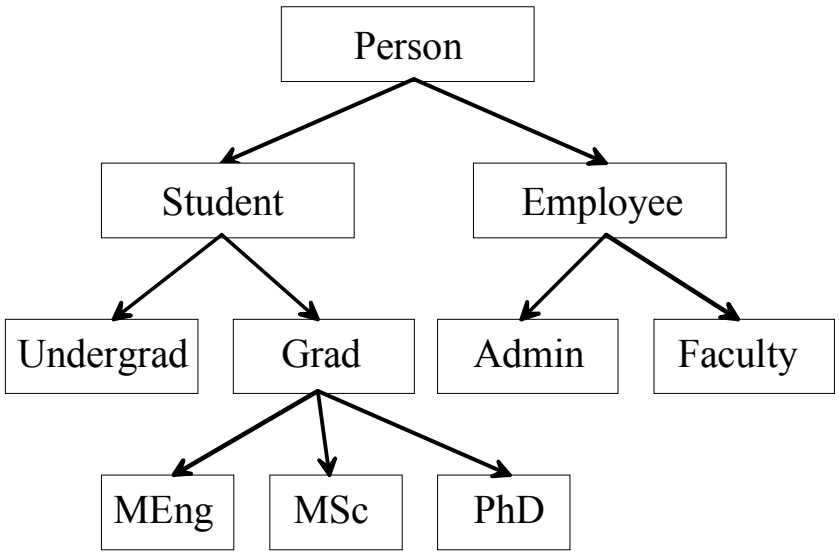

```
public class Child extends Parent
 {
    ...
 }
```

Η δήλωση λοιπόν της σχέσης κληρονομικότητας μεταξύ δύο κλάσεων είναι της μορφής:

```
public class <υποκλάση> extends <υπερκλάση>
```

Έτσι δημιουργείται μία ιεραρχία, όπου από τη γενικότερη κλάση μπορούμε να περάσουμε σε ειδικότερες, όπου κάθε μία έχει και τα χαρακτηριστικά της γενικότερης προγόνου της, αλλά και αυτά των ακόμα γενικότερων προγόνων της προγόνου της!

π.χ.) Θέλουμε να φτιάξουμε ένα πρόγραμμα που να περιέχει στοιχεία για τα μέλη ενός πανεπιστημίου. Μια ομαδοποίηση των μελών μπορεί να είναι η εξής:




Στο παράδειγμα αυτό, θα είχαμε δηλώσεις κλάσεων της μορφής:

```
public class Person
```

```
{
  ...
}

public class Student extends Person
{
  ...
}

public class Undergrad extends Student
{
  ...
}

public class Grad extends Student
{
  ...
}
```

κτλ.

*Και* τα αντικείμενα της `Student` *και* τα αντικείμενα της `Employee` αντιπροσωπεύουν ανθρώπους, άρα έχουν κάποιες κοινές ιδιότητες. Επομένως, οι μέθοδοι που θα αρχικοποιούν, θα τροποποιούν ή θα κάνουν output π.χ. το όνομα κάποιου μέλους του πανεπιστημίου, είτε είναι φοιτητής (Student) είτε είναι εργαζόμενος (Employee), θα είναι ίδιες! Όλες αυτές τις μεθόδους τις βάζουμε σε μία υπερκλάση `Person`.

Μια απλή μορφή της κλάσης `Person` είναι η εξής:

```
public class Person
{
        // class variables
  // variable to hold the name of the person
  private String name;

  // constructors

  public Person()
  {
    name = "No name yet..";
  }

  public Person(String initName)
  {
    name = initName;
  }
  // methods

  // modifier method to change the name
  public void setName(String newName)
  {
    name = newName;
  }
```

```
  // accessor method to return the name
  public String accessName()
  {
    return name;
  }

  // method to print the output
  public void writeOutput()
  {
    System.out.println("Name: " + name);
  }

}
```

Η κλάση `Student` κληρονομεί την κλάση `Person` (τις μεταβλητές της (το `name` δηλαδή) και τις μεθόδους της) και προσθέτει μια δικιά της μεταβλητή για τον αριθμό μητρώου του κάθε φοιτητή:

```
public class Student extends Person
{
         // class variables
  // variable to hold student's ID number
  private int studentNumber;

  // constructors

  public Student()
  {
    super();
    studentNumber = 0;
  }

  public Student(String initName, int initStudNo)
  {
    super(initName);  // -> klisi tou constructor tis Person
    studentNumber = initStudNo;
  }

  // methods

  //  modifier  method  to  change  values  of  name  and
studentNumber
  public void reset(String newName, int newStudNo)
  {
    setName(newName);  // -> klisi methodou tis Person!
    studentNumber = initStudNo;
  }

  // modifier method to change value of studentNumber
  public void setStudNumber(int newStudNo)
  {
    studentNumber = newStudNo;
  }
```

```
  // accessor method to return the studentNumber
  public int accessStudentNumber()
  {
    return studentNumber;
  }

  // method to print the output
  public void writeOutput()
  {
    System.out.println("Name: " + accessName());
    System.out.println("Student no.: " + studentNumber);
  }
}
```

Αν λοιπόν είχαμε τις δύο αυτές κλάσεις υποστήριξης, στην κλάση εφαρμογής θα μπορούσαμε να είχαμε τα εξής:

```
public class InheritanceExample
{
  public static void main (String [ ] args)
  {
    Student s = new Student();
    s.setName("Nikos Papadopoulos"); // -> setName tis Person
    s.setStudentNumber(14934);  //  ->  setStudentNumber  tis
Student
    s.writeOutput();      //   ->   writeOutput   tis   Student
(OVERRIDING)
  }
}
```

Η writeoutput() που «τρέχει» είναι της Student και όχι της Person. Αυτό λέγεται overriding (υπερκάλυψη). Αν στην υποκλάση υπάρχει μέθοδος με ίδιο όνομα, πλήθος και είδος παραμέτρων με κάποια μέθοδο της υπερκλάσης, τότε η μέθοδος της υποκλάσης κάνει override (παρακάμπτει) τη μέθοδο της υπερκλάσης.

Αν η μέθοδος που παρακάμπτεται (της υπερκλάσης) επιστρέφει (`return`) κάποια μεταβλητή, η νέα μέθοδος που την κάνει override (στην υποκλάση) πρέπει να επιστρέφει μεταβλητή του ίδιου τύπου.

--> **Overriding (υπερκάλυψη)** / **Overloading (υπερφόρτωση)**

- Ίδιο όνομα μεθόδων
- και ίδιο πλήθος παραμέτρων
- και ίδιο είδος παραμέτρων
- και ίδιο είδος επιστροφής (return)

- Ίδιο όνομα μεθόδων
- και διαφορετικό πλήθος παραμ.
- ή διαφορετικό είδος παραμ.
- ή διαφορετικό είδος επιστροφής

---

Η λέξη `final` δηλώνει μέθοδο που δεν μπορεί να γίνει overridden.

π.χ.
```
public final void specialMethod()
{
  ...
}
```

---

Η λέξη `super`:

i) `super.writeOutput(); //` -> κλήση μεθόδου που έχει γίνει overridden

ii) Η `super()` τρέχει τον constructor της υπερκλάσης.

ΠΡΟΣΟΧΗ! Η λέξη super όταν χρησιμοποιείται για την κλήση ενός constructor της υπερκλάσης μέσα σε κάποιον constructor της υποκλάσης, πρέπει να είναι η πρώτη εντολή αυτού του constructor. Δεν μπορεί να χρησιμοποιηθεί αργότερα μέσα στον constructor.

Στην πραγματικότητα, αν δεν υπάρχει κλήση του constructor της υπερκλάσης (κάποια μορφή της `super()` δηλαδή) μέσα στον constructor της υποκλάσης, τότε η JAVA καλεί *αυτόματα* τον default constructor της υπερκλάσης!
Άρα, στον constructor της `Student`:

```
public Student()
{
  super();
  studentNumber = 0;
}
```

θα ήταν το ίδιο αν γράφαμε:

```
public Student()
{
  studentNumber = 0;
}
```

Συμβουλή: Καλύτερα να γράφουμε την κλήση της `super()` κι ας μην είναι απαραίτητο, για να είναι πιο σαφής ο κώδικάς μας.

---

Η λέξη `this` αναφέρεται στον constructor της υποκλάσης.

π.χ., ένας ακόμη constructor της `Student` θα μπορούσε να είναι ο εξής:

```
public Student(String initName)
{
  this(initName, 0); -> klisi tou contructor: Student(String, int)
}
```

Σημείωση: Πρέπει και η κλήση της `this` να είναι η πρώτη εντολή του constructor.

---

Από αυτά που είπαμε μέχρι τώρα για την super, προκύπτει ότι θα μπορούσαμε να γράψουμε πιο σωστά τη μέθοδο `writeOutput()` της `Student`, χωρίς επανάληψη κώδικα:

Η αρχική μέθοδος:

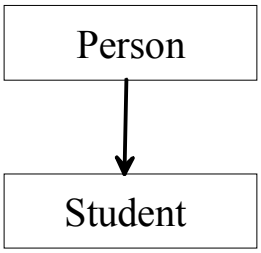
```
public void writeOutput()
{
  System.out.println("Name: " + accessName());
  System.out.println(Student no.: " + studentNumber);
}
```

μπορεί να γίνει:

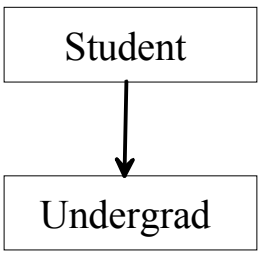
```
public void writeOutput()
{
  super.writeOutput();
  System.out.println(Student no.: " + studentNumber);
}
```

---

Όπως είχαμε το:




έτσι μπορούμε να έχουμε και το:




Στη δεύτερη αυτή σύνδεση (κληρονομικότητα), ο κλάση Student γίνεται πλέον υπερκλάση και το ρόλο της υποκλάσης παίζει η κλάση Undergrad. Έτσι δημιουργείται μια αλυσίδα κληρονομικότητας που αποτελείται από απλά ζευγάρια κληρονομικότητας. Η αλυσίδα αυτή ξεκινάει από το «γενικό» και προχωράει προς το «ειδικό». Η «εξειδίκευση» όμως

αυτή κάνει τις κλάσεις «ευρύτερες», αφού κληρονομούν τα στοιχεία των γενικότερων κλάσεων και μπορούν να τα χρησιμοποιούν σαν δικά τους.

Μέχρι τώρα είχαμε δει τις εξής δηλώσεις ορατότητας μεταβλητών ή μεθόδων:

`public`, `private` και χωρίς δήλωση.

Υπάρχει και η δήλωση ορατότητας **`protected`** η οποία κάνει τις μεταβλητές ή μεθόδους ορατές μόνο από κλάσεις που σχετίζονται με κληρονομικότητα. Δηλαδή μια μεταβλητή ή μέθοδος που δηλώνεται σαν `protected`, λειτουργεί σαν `public` για τις κλάσεις απογόνους της κλάσης ορισμού της και σαν `private` για όλες τις υπόλοιπες κλάσεις του προγράμματος.

Ο παρακάτω πίνακας δείχνει την αύξηση του εύρους των διαθέσιμων στοιχείων των κλάσεων, καθώς αυτές εξειδικεύονται και συγχρόνως κληρονομούν στοιχεία των προγόνων τους, για το παράδειγμά μας:

| Αντικείμενα της κλάσης ---> | Person | Student | Undergrad |
|---|---|---|---|
| Χρησιμοποιούν μεταβλ. & μεθ. της κλάσης: | | | |
| Person | ναι | ναι | **ναι** |
| Student | όχι | ναι | ναι |
| Undergrad | όχι | όχι | ναι |

Προφανώς κάθε κλάση χρησιμοποιεί μεταβλητές και μεθόδους του εαυτού της, αλλά βλέπουμε ότι η `Person` χρησιμοποιεί απλά τα δικά της στοιχεία, ενώ η `Student` χρησιμοποιεί τα δικά της και της `Person`, ενώ η `Undergrad` χρησιμοποιεί και τα δικά της και της `Student`, αλλά και της `Person`. Όλα αυτά βέβαια αφορούν μεταβλητές και μεθόδους ορατότητας `public` ή `protected`.

---

Η κλάση Undergrad του παραδείγματός μας:

```
public class Undergrad extends Student
{
         // class variables
  // variable to hold student's year of study
  private int year;

  // constructors

  public Undergrad()
  {
    super();  // kalei ton default constructor tis Student
    year = 1;
  }

  public  Undergrad(String  initName,  int  initStudNo,  int
initYear)
  {
    super(initName, initYear);  // -> constructor tis Student
    year = initYear;
```

```
  }

  // methods

  // modifier method to change values of name, studentNumber and year
  public void reset(String newName, int newStudNo, int
newYear)
  {
    reset(newName, newStudNo); //-> klisi tis overloaded reset tis
Student
    year = newYear;
  }

  // modifier method to change value of year
  public void setYear(int newYear)
  {
    year = newYear;
  }

  // accessor method to return the year
  public int accessYear()
  {
    return year;
  }

  // method to print the output
  public void writeOutput()
  {
    super.writeOutput();
    System.out.println("Student year: " + year);
  }

}
```

# Σημειώσεις JAVA – 8η εβδομάδα

## Κληρονομικότητα (συνέχεια)

Η κληρονομικότητα ορίζεται μεταξύ δύο και μόνο κλάσεων,

από υπερκλάση --> σε υποκλάση
ή από parent --> σε child

και ο ορισμός γίνεται στην υποκλάση (child) ως εξής:

```
public class Child extends Parent
```

Στην κληρονομικότητα πάμε από το «γενικό» στο «ειδικό», αλλά το «ειδικό» κληρονομεί και τις ιδιότητες και τα στοιχεία του «γενικού».

Στην JAVA τα αντικείμενα τύπου `Parent` είναι μόνο τύπου `Parent`, ενώ τα αντικείμενα τύπου `Child` είναι τύπου `Child` και τύπου `Parent`,

Άρα, στο παράδειγμα της προηγούμενης βδομάδας, το:

```
Person p1 = new Student();
```

είναι **σωστό** γιατί ένα αντικείμενο τύπου `Student` είναι και τύπου `Person`, ενώ το:

```
Student s1 = new Person();  // ΛΑΘΟΣ!
```

είναι **λάθος**. Το:

```
Student s2 = new Undergrad();
```

είναι **σωστό**, όπως και το:

```
Person p2 = new Undergrad();
```

Αφού τα αντικείμενα της `Undergrad` είναι και τύπου `Student` αλλά και τύπου `Person`.

---

Ορατότητα:

| Ορατότητα: | `public` | `protected` | χωρίς δήλωση ορατότητας | `private` |
|---|---|---|---|---|
| Από την ίδια την κλάση | ναι | ναι | ναι | ναι |
| Από άλλη κλάση στο ίδιο πακέτο | ναι | **ναι** | ναι | όχι |
| Από άλλη κλάση έξω από το πακέτο | ναι | όχι | όχι | όχι |
| Από υποκλάση στο ίδιο πακέτο | ναι | **ναι** | ναι | όχι |
| Από υποκλάση έξω από το πακέτο | ναι | **ναι** | όχι | όχι |

---

**Η λέξη `final`:**

α) Σε μεταβλητές: Δεν αλλάζει η τιμή τους μετά την αρχικοποίησή τους.

π.χ.
```
private final int x = 5;
x = 6; //--> ΛΑΘΟΣ!
```

β) Σε μεθόδους: Δεν γίνονται override.

π.χ.
```
public final void specialMethod()
{
  ...
}
```

γ) Σε κλάσεις: Δεν κληρονομούνται.

π.χ.
```
public final class MyClass
{
  ...
}
------------------
public class ClassName extends MyClass   // -->
ΛΑΘΟΣ!
```

---

# Βασικά στοιχεία αντικειμενοστραφούς προγραμματισμού / Ανακεφαλαίωση

- Βασικά στοιχεία προγραμμάτων JAVA: **Κλάσεις**

- Κλάση --> Καλούπι για δημιουργία αντικειμένων

- Η κλάσει έχει: constructors (για αρχικοποίηση), μεταβλητές και μεθόδους (operating, modifier, accessor)

- Όλα τα στοιχεία μιας κλάσης ανήκουν στα αντικείμενά της. Δηλαδή, μια μεταβλητή `x` μιας κλάσης `MyClass`, έχει τόσα αντίγραφα (τόσες υποστάσεις) όσα (-ες) και τα αντικείμενα της κλάσης `MyClass`. Το ίδιο ισχύει και για τις μεθόδους. Δηλαδή, κάθε αντικείμενο έχει τις δικές του μεταβλητές και μεθόδους, όπως αυτές ορίζονται στην κλάση του (στο «καλούπι» του).

- Εξαίρεση αποτελούν οι μεταβλητές και μέθοδοι που δηλώνονται ως **`static`**. Αυτές είναι κοινές για όλα τα αντικείμενα μιας κλάσης και δεν χρειάζεται η δημιουργία αντικειμένου για τη χρήση τους. Καλούνται με τη μορφή π.χ. `MyClass.methodName();` αντί για `obj1.methodName();` (όπου `obj1` είναι ένα αντικείμενο της `MyClass`).

  Σημείωση: Μέθοδοι που έχουν δηλωθεί `static`, βλέπουν και τροποποιούν μόνο `static` μεταβλητές.

  Άρα, η `main` (που είναι `static`) διαχειρίζεται μόνο static μεταβλητές, ή μεταβλητές που έχουν δηλωθεί μέσα στην ίδια τη `main`, με τη μορφή:

  ```
  <τύπος> <μεταβλητή> [= αρχική_τιμή];
  ```

  (χωρίς δήλωση ορατότητα ή `static`)

- Στο πρόγραμμά μας, κάθε κλάση είναι ξεχωριστό αρχείο.

- Το πρόγραμμα «χτίζεται» στις κλάσεις υποστήριξης.

- Η κλάση εφαρμογής έχει τη μέθοδο main, η οποία καλείται αρχικά όταν «τρέχουμε» το πρόγραμμα.

- Πριν αρχίσουμε τον προγραμματισμό, φτιάχνουμε τον **αλγόριθμο** του προγράμματος. Εκεί σκεφτόμαστε τι κλάσεις χρειαζόμαστε και ποια ιεραρχία μπορεί να υπάρξει (γενικά στοιχεία / ειδικά στοιχεία) ώστε να δημιουργήσουμε σχέσεις κληρονομικότητας (εάν κάτι τέτοιο είναι εφικτό και χρήσιμο).

- Μπορούμε να χρησιμοποιήσουμε έτοιμες κλάσεις που ομαδοποιούνται σε πακέτα, κάνοντάς τα `import`. Επίσης, μπορούμε να φτιάξουμε δικά μας πακέτα κλάσεων.

- Υπάρχει δυνατότητα κάποιες μέθοδοι να γίνουν overload (*συνήθως μέσα στην ίδια κλάση*) --> οι μέθοδοι αυτές έχουν διαφορετικό signature

- Υπάρχει δυνατότητα κάποιες μέθοδοι να γίνουν `override` (*μεταξύ κλάσεων με σχέση κληρονομικότητας*) --> οι μέθοδοι αυτές έχουν ίδιο signature

- Μια κλάση μπορεί να κληρονομήσει μόνο μία κλάση

  ```
  public class Child extends Parent
  ```

  αλλά κληρονομεί και τα στοιχεία <u>όλων</u> των προγόνων της κλάσης που κληρονομεί (`Parent`).

- Με τη λέξη `super` αναφερόμαστε στον constructor της υπερκλάσης της κλάσης στην οποία τη χρησιμοποιούμε (τη λέξη `super`).

- Με τη λέξη `this` αναφερόμαστε στην υποκλάση

- Περιορισμοί ορατότητας γίνονται με τις δηλώσεις:
  - `private`
  - `protected`
  - (χωρίς δήλωση) (default ορατότητα)

  μπροστά από μια μεταβλητή ή μια μέθοδο, αντί για `public`.

- Περιορισμοί κληρονομικότητας γίνονται με τη λέξη `final`.

- Περιορισμοί «ποικιλομορφίας» γίνονται με τη λέξη `static`.

# Σημειώσεις JAVA – 9η εβδομάδα

## Vectors (διανύσματα)

Τα vectors είναι δυναμικές δομές δεδομένων.
Κυριότερο χαρακτηριστικό τους: έχουν δυναμικό (μεταβαλλόμενο) μέγεθος.

Το μέγεθος ενός πίνακα δεν μπορεί να αλλάξει από τη στιγμή που ο πίνακας οριστεί, ακόμα και αν το μέγεθος δεν ορίζεται επ'ακριβώς μέσα στο πρόγραμμα αλλά το δίνει ο χρήστης σαν input στο πρόγραμμα.

π.χ.
```
double [ ] a = new double [sizeOfa];
```
όπου το `int sizeOfa` το δίνει ο χρήστης.

Αν κατόπιν θέλουμε να έχουμε περισσότερα στοιχεία (elements) στον `a`, τότε υπάρχει πρόβλημα! Γι αυτό υπάρχουν τα vectors, που έχουν μεταβλητό μέγεθος.

-> Τότε γιατί να μην χρησιμοποιούμε πάντα vectors αντί για πίνακες;
Τα vectors έχουν δύο βασικά προβλήματα:

i) Είναι λιγότερο αποτελεσματικά από τους πίνακες
ii) Τα στοιχεία τους πρέπει να είναι αντικείμενα (objects). Δεν μπορούν να είναι απλές τιμές πρωτογενών μεταβλητών (int, double, char, κτλ.) όπως στους πίνακες.

---

### Χρήση των vectors:

Ο ορισμός της κλάσης `Vector` δεν παρέχεται αυτόματα. Βρίσκεται στο πακέτο `java.util` , το οποίο πρέπει να γίνει import από οποιοδήποτε πρόγραμμα χρησιμοποιεί vectors:

```
import java.util.*;
```

### Δημιουργία vector:

Η `Vector` είναι κλάση, άρα τα vectors δημιουργούνται όπως δημιουργούνται τα αντικείμενα:

```
Vector vectorName = new Vector();
```

Κάθε vector έχει μια χωρητικότητα (capacity) και μια «αύξηση χωρητικότητας» (capacityIncrement) και ένα μέγεθος (size). Η χωρητικότητά του είναι πάντα μεγαλύτερη ή ίση με το μέγεθός του (capacity >= size). Σαν μέγεθος θεωρούμε το πλήθος των στοιχείων που έχει ανά πάσα στιγμή το vector.

**Constructors της `Vector`:**

- `public Vector()` – δημιουργεί ένα κενό vector με αρχική χωρητικότητα = 10. Κάθε φορά που χρειάζεται αύξηση της χωρητικότητας, η χωρητικότητα διπλασιάζεται.

- `public Vector(int initialCapacity)` – όπως προηγουμένως, αλλά η αρχική χωρητικότητα είναι = `initialCapacity`.

  π.χ., `Vector v1 = new Vector(35);`

- `public Vector(int initialCapacity, int capacityIncrement)` – όπως προηγουμένως, αλλά η χωρητικότητα μεγαλώνει κατά "`capacityIncrement`" στοιχεία κάθε φορά που χρειάζεται.

## Μέθοδοι της `Vector`:

### - Προσθήκη στοιχείου:

*- στο τέλος του vector (μετά το τελευταίο στοιχείο)*

```
public void addElement(Object newElement)
```

π.χ., αν έχουμε το vector `v2`:

| 4 | 2 | 1 | 0 | 15 | 9 | |
|---|---|---|---|---|---|---|

και θέλουμε να προσθέσουμε μια `int` τιμή στο τέλος του vector:

```
private int x = 2;
v2.addElement(new Integer(x));
```

->

| 4 | 2 | 1 | 0 | 15 | 9 | 2 | |
|---|---|---|---|---|---|---|---|

*- ενδιάμεσα:*

```
public void insertElementAt(Object newElement, int index)
```

π.χ., `v2.insertElementAt(new Integer(7),4);` ->

| 4 | 2 | 1 | 0 | 7 | 15 | 9 | 2 | |
|---|---|---|---|---|---|---|---|---|

### - Αντικατάσταση (υπάρχοντος) στοιχείου:

```
public void setElementAt(Object newElement, int index)
```

π.χ., θέλουμε να αλλάξουμε το *τρίτο* στοιχείο του `v2` σε "5":

```
v2.setElementAt(new Integer(5), 2);
```

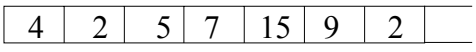
->

| 4 | 2 | 5 | 0 | 7 | 15 | 9 | 2 | |
|---|---|---|---|---|---|---|---|---|

- **Διαγραφή στοιχείου:**

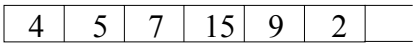
- `public void removeElementAt(int index)`

π.χ., `v2.removeElementAt(3); ->`




- `public boolean removeElement(Object theElement)`

π.χ., `v2.removeElement(2); -> true ->`




`v2.removeElement(3); -> false`

- `public void removeAllElements()`

- **Πρόσβαση σε στοιχείο:**

- `public Object elementAt(int index)`

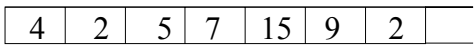
π.χ., θέλουμε να προσθέσουμε τις τιμές του 3ου και 5ου στοιχείου του v2:

```
int sum = ((Integer)v2.elementAt(2)).intValue() +
          ((Integer)v2.elementAt(4)).intValue();
          | |         |_____object_____| |           |
          | |_____Integer object_________|           |
          |_____________________int_____________________|
```

- **Μέθοδοι αναζήτησης:**

- `public boolean contains(Object target)  -> true/false`

π.χ., αν έχουμε το vector v2 στη μορφή:




`v2.contains(new Integer(15));  -> true`

- `public int indexOf(Object target) -> 1ο index του target`
  `ή -1 αν δεν υπάρχει`

π.χ., πάντα για το ίδιο v2:

```
v2.indexOf(new Integer(2));  -> 1
v2.indexOf(new Integer(3));  -> -1
```

- `public int indexOf(Object target, int StartIndex)`

π.χ., `v2.indexOf(new Integer(2), 3);  -> 6`

- `public int lastIndexOf(Object target)`

π.χ., `v2.lastIndexOf(new Integer(2));  -> 6`

- public Object firstElement()

π.χ., v2.firstElement(); -> 4 (το οποίο όμως είναι αντικείμενο!)

- public Object lastElement()

π.χ., v3.lastElement(); -> 2 (αντικείμενο)

**- <u>Άλλες μέθοδοι:</u>**

- public int size()

- public int capacity()

- public void trimToSize() -> κάνει το capacity = με το size

- public void setSize(int newSize) -> κάνει το <u>μέγεθος</u> (και όχι το capacity) = με το newSize.

Αν το newSize > προηγούμενο size -> τα νέα στοιχεία = null.
Αν το newSize < προηγούμενο size -> τα επιπλέον στοιχεία χάνονται.

π.χ., Vector v3 = new Vector();

v3.addElement(new Double(2.3)); ->

| 0 | 1 | 2 | 3 | ... | 9 |
|---|---|---|---|---|---|
| 2.3 | | | | ... | |

v3.addElement(new Double(1.5)); ->

| 2.3 | 1.5 | | | ... | |
|---|---|---|---|---|---|

v3.addElement(new Double(4.2)); ->

| 2.3 | 1.5 | 4.2 | | ... | |
|---|---|---|---|---|---|

v3.size(); // -> 3

v3.capacity(); // -> 10

v3.trimToSize(); ->

| 2.3 | 1.5 | 4.2 |
|---|---|---|

v3.setSize(5); ->

| 2.3 | 1.5 | 4.2 | null | null |
|---|---|---|---|---|

v3.setSize(2); ->

| 2.3 | 1.5 |
|---|---|

v3.setSize(3); ->

| 2.3 | 1.5 | null |
|---|---|---|

**Παράδειγμα χρήσης vector:**

*- Να γραφεί μέθοδος που να ζητάει από τον χρήστη εισαγωγή double τιμών διαφορετικών του μηδενός μέχρι να δοθεί η τιμή 0 (για τερματισμό της διαδικασίας), και να αποθηκεύει τις τιμές αυτές.*

Μπορούμε να χρησιμοποιήσουμε έναν double πίνακα (array) ;
Όχι, γιατί δεν ξέρουμε εξ αρχής πόσες τιμές θα εισάγει ο χρήστης! Άρα, χρησιμοποιούμε vector για την αποθήκευση των τιμών. Ένας τρόπος είναι ο ακόλουθος (υπάρχουν φυσικά διάφοροι τρόποι για να έχουμε το ίδιο αποτέλεσμα....):

```
import javax.swing.*;  // gia xrisi tou JOptionPane
import java.util.*;    // gia xrisi tou Vector

public class VectorExample
{
   Vector data = new Vector();   // dimiourgia tou Vector
"data"
   .
   .
   .
   // H methodos pou ziteitai:

   public void askAndStoreData()  // Epikefalida 'h ypografi (signature)
   {
      do
      {
         String s = JOptionPane.showInputDialog("Δώσε έναν
         πραγματικό αριθμό. Δώσε 0 για τερματισμό.");

         double input = Double.parseDouble(s);

         data.addElement(new Double(input));
      }
      while (input != 0);  // epanelave oso den dinetai i timi
0

      // prepei na svisoume to 0 pou apothikeutike sto telos!
      data.removeElement(new Double(0));
      // ή: data.removeElementAt(data.size()-1);

   } // end method

} // end class
```

---

Είπαμε ότι για να προσθέσουμε μεταβλητές πρωτογενών τύπων (primitive types) σε vectors, τις μετατρέπουμε πρώτα σε αντικείμενα (Objects) με τη χρήση κάποιων ειδικών κλάσεων. Π.χ.,

```
vectorName.addElement(new Double(x));  // το x είναι double

vectorName.addElement(new Integer(y));  // το y είναι int

vectorName.addElement(new Float(z));  // το z είναι float
```

Πώς προσθέτουμε ένα String σαν στοιχείο στο vector;

```
vectorName.addElement(s);  // το s είναι String
```

Γιατί έτσι απλά; Γιατί τα strings είναι αντικείμενα!

Σημείωση: Αναφέρθηκε ότι ο μόνος «περιορισμός» είναι ότι τα στοιχεία των vectors πρέπει να είναι αντικείμενα. Άρα, ένα vector μπορεί να περιέχει (φαινομενικά τουλάχιστον) στοιχεία διαφορετικών τύπων (φαινομενικά γιατί στην ουσία όλα τα στοιχεία είναι αντικείμενα). Π.χ., ένα vector μπορεί να είναι το εξής:

| index | **vector** | |
|---|---|---|
| 0 | 3.14 | --> double |
| 1 | 10 | --> int |
| 2 | Hello | --> String |
| 3 | a | --> char |
| 4 | 5 7 2 | --> πίνακας |

# Γενικές Ασκήσεις / Παραδείγματα

**1)** Τι εκτυπώνει το ακόλουθο πρόγραμμα;

```
public class What
{
   public static void main (String [] args)
   {
      int [] a = {0, 1, 0, 1, 0, 1};
      int n = 3;
      a[n] = n++;
      for (int j=0; j<a.length; j++)
         System.out.println(a[j]);
   }
}
```

Απάντηση: 010301

**2)** Τι εκτυπώνει το ακόλουθο πρόγραμμα;

```
public class Guess
{
   public static void main (String [] args)
   {
      String s1 = new String().valueOf(10);
      String s2 = "10";
      if (s1==s2)
        System.out.println(s1+s2);
      else

System.out.println(Integer.parseInt(s1)+Integer.parseInt(s2));
   }
}
```

Απάντηση: 20

**3) α)** Να ορισθεί `public` μέθοδος `methodA` που να δέχεται ως παραμέτρους έναν μονοδιάστατο πίνακα ακέραιων τιμών, μία ακέραια μεταβλητή και μία πραγματική μεταβλητή και να επιστρέφει είτε την τιμή `true` είτε την τιμή `false`.
Η μέθοδος να βρίσκεται μέσα στην κλάση `ClassA`.

Απάντηση:

```
public class ClassA
{
  public boolean methodA(int [] arr, int a, double b)
  {
      ...
  }
}
```

**β)** Μέσα σε μια άλλη κλάση `ClassB` να ορισθεί μονοδιάστατος πίνακας `myArray` ακέραιων τιμών, μεγέθους 10, μια ακέραια μεταβλητή `x` και μια πραγματική μεταβλητή `y`, και να δημιουργηθεί ένα αντικείμενο `objA` της `ClassA`.

Απάντηση:

```
public class ClassB
{
   private int [] myArray = new int [10];
   private int x;
   private double y;

   ClassA objA = new ClassA();
```

**γ)** Σαν συνέχεια του (β) ερωτήματος, να ορισθεί operator μέθοδος `methodB` χωρίς παραμέτρους που να καλεί την `methodA` με τα στοιχεία που ορίσθηκαν στο (β) και να βάζει την τιμή που επιστρέφει σε ανάλογη μεταβλητή `z`.

Απάντηση:

```
public void methodB()
{
   boolean z = objA.methodA(myArray, x, y);
}
```

**4) α)** Να γραφεί μέθοδος change που να δέχεται vector με στοιχεία 0 και 1 και να επιστρέφει νέο vector που να έχει 0 αντί για 1 και 1 αντί για 0.

Απάντηση:

```
public Vector change(Vector v1)
{
   Vector v2 = new Vector();

   for (int i=0; i<v1.size(); i++)
   {
      if (((Integer)v1.elementAt(i)).intValue() == 0)
         v2.addElement(new Integer(1));
      else
         v2.addElement(new Integer(0));
   }
   return v2;
}
```

**β)** Να γίνει το αντίστοιχο για πίνακα δύο διαστάσεων.

Απάντηση:

```
public int [][] changeMatrix(int [][] a1)
{
   int [][] a2 = new int [a1.length][a1[0].length];

   for (int i=0; i<a1.lengthl i++)
   {
      for (int j=0; j<a1[0].length; j++)
      {
         if (a1[i][j] == 0)
            a2[i][j] = 1;
         else
            a2[i][j] = 0;
      }
   }
   return a2;
}
```

# Σημειώσεις JAVA – 11η εβδομάδα

**Ανακεφαλαίωση – Παραδείγματα**

**ΥΛΗ:** Από το βιβλίο τα **κεφάλαια 2 – 9** (εκτός των παραγράφων 8.4.2 και 8.4.7) + **Vectors** + **JOptionPane** (γενική χρήση, όχι η ακριβής σύνταξη)

Πιο συγκεκριμένα, ό,τι υπάρχει στις σημειώσεις από τις παραδόσεις εκτός από τα περί αρχείων (ανάγνωση από και εγγραφή σε αρχείο).

## Επανάληψη

- Σύνταξη, κλάσεις (εφαρμογής και υποστήριξης), είδη μεθόδων, κλήση μεθόδων, πρωτογενείς τύποι μεταβλητών, μαθηματικές πράξεις, λογικές σχέσεις, συγκρίσεις, μετατροπές τύπων, εμβέλεια. −−> *βλέπε Σημειώσεις 4ης εβδομάδας*
- Constructors --> τι κάνουν, πως ορίζονται (σύνταξη), overloading
- Έλεγχος ροής --> if, if/else, switch, for, while, do/while και μετατροπές από μια δομή σε μια άλλη
- Οι τελεστές ++ και --
- Κλάση String --> τρόποι δημιουργίας, τι κάνουν οι βασικές μέθοδοι, πως συγκρίνονται τα strings, μετατροπές από/σε String
- Πακέτα --> τι είναι, πως χρησιμοποιούνται, ορατότητες
- Κληρονομικότητα (Σημειώσεις 7ης και 8ης εβδομάδας) --> τι είναι και γιατί υπάρχει, πως ορίζεται, overriding, super/this, final, παράδειγμα Person/Student/Undergrad, ορατότητες
- Ανακεφαλαίωση σημειώσεων 8ης εβδομάδας --> static κτλ.
- Πίνακες (arrays)
  - πως δηλώνονται 1-D και 2-D πίνακες
  - πως αρχικοποιούνται
  - πως χρησιμοποιούνται τα στοιχεία τους
  - πως βρίσκουμε το μήκος 1-D και 2-D πινάκων
- Vectors
  - πως δηλώνονται
  - πλεονεκτήματα / μειονεκτήματα
  - τι κάνουν οι βασικές μέθοδοι που αναφέραμε, πως συντάσσονται
  - πως μετατρέπονται στοιχεία αντικείμενα σε τιμές πρωτογενών τύπων
- Χειρισμός εξαιρέσεων
  - σκοπός
  - τρόπος χειρισμού --> try-catch-finally
  - πως «πετάμε» δικές μας εξαιρέσεις --> throw

1) Έχουμε τον εξής κώδικα:

```
public class ClassB
{
   ClassA objA = new ClassA();

   public void methodB()
   {
      Vector v1 = new Vector();
      double x1 = 10.5;
      double [] arr1 = new double [50];
      arr1 = objA.methodA(x1, v1);
   }
}
```

Που βρίσκεται η μέθοδος methodA και ποια είναι η υπογραφή της (πως ορίζεται);

Απάντηση:

Βρίσκεται στην κλάση ClassA και η υπογραφή της είναι η εξής:

```
public double [] methodA(double a, Vector v)
{

}
```

---

2) Μέσα σε μια μέθοδο υπάρχει ο εξής κώδικας:

```
int x1;
char x2;
String x3;
Vector vec = new Vector();
vec = methodName(x1, x2, x3);
```

Ποια είναι η υπογραφή της μεθόδου methodName;

Απάντηση:

Η methodName μπορεί να είναι είτε `public` είτε `private` και ο ορισμός της είναι ο εξής:

```
public Vector methodName(int x, char y, String z)
{

}
```

---

3) Έχουμε τον ακόλουθο ορισμό της μεθόδου methodA

```
public double [] [] methodA(String a, char b, double c)
{
```

```
    . . .
}
```

και τους ακόλουθους ορισμούς:

```
double [] [] arr1 = new double [10][10];
double [] arr2 = new double [15];
int [] [] arr3 = new int [10][10];
private x1, x2 = 5;
private double x3 = 10;
char x4 = 'a', x5 = 'b';
String x6 = new String ("Hello");
```

Πως θα ήταν μια πιθανή κλήση της μεθόδου methodA;

Απάντηση:

```
arr1 = methodA(x6, x4, x3);

ή  arr1 = methodA(x6, x5, x3);
```

---

4) Αντικατάσταση στοιχείων πίνακα με την τετραγωνική τους ρίζα.

```
private int arraySize = 100;
private double [] myData = new double [arraySize];
...
// ο πίνακα myData παίρνει τιμές
...

for (int i=0; i<myData.length; i++)
{
   if (myData[i]>0)
   {
      myData[i] = Math.sqrt(myData[i]);
   }
   else
   {
      myData[i] = 0; // για να μη χτυπήσει σφάλμα στους αρνητικούς
   }
}
```

(Σημείωση: Θα μπορούσαν να χρησιμοποιηθούν και άλλοι τρόποι αποφυγής του σφάλματος σε περίπτωση αρνητικών τιμών του πίνακα, όπως ο κατάλληλος χειρισμός της πιθανής εξαίρεσης με try-catch)

---

5) Να γραφεί μέθοδος myMethod που να δέχεται 3 ακέραιες μεταβλητές σαν παραμέτρους εισόδου. Μέσα στη μέθοδο υπάρχει for-loop που:

i) αρχικοποιεί τον μετρητή στην τιμή της πρώτης παραμέτρου εισόδου
ii) επαναλαμβάνει το loop όσο ο μετρητής είναι <= της $2^{ης}$ παραμέτρου

iii) αυξάνει τον μετρητή κατά την τιμή της 3ης παραμέτρου σε κάθε επανάληψη.

Το for-loop πρέπει να υπολογίζει το άθροισμα των τιμών που παίρνει ο μετρητής, χωρίς αυτό να αλλάζει την τιμή του. Το άθροισμα αυτό θα επιστρέφεται από τη μέθοδο.

Απάντηση:

```
public int myMethod(int x, int y, int z)
{
   int sum = 0;

   for (int i=x, i<=y; i+=z)
   {
      sum = sum + i;  // ή sum+=i;
   }

   return sum;
}
```

[ Σημείωση: Οπότε, μια κλήση της μεθόδου θα μπορούσε να είναι η:
```
   int mySum = myMethod(2,12,2);
```
η οποία θα επέστρεφε την τιμή 42 (2+4+6+8+10+12) ].

---

6) Έστω ότι έχει δημιουργηθεί ο πίνακας:

```
   private int row = 6, col = 6;
   private double [] [] myArray = new double [row][col];
```

Τι από τις ακόλουθες 5 περιπτώσεις κάνουν οι παρακάτω κώδικες;

i) πρόσθεση των στοιχείων μιας στήλης του myArray
ii) πρόσθεση όλων των διαγώνιων στοιχείων του myArray
iii) πρόσθεση όλων των στοιχείων του myArray
iv) πρόσθεση των στοιχείων μιας σειράς του myArray
v) πρόσθεση των στοιχείων μιας στήλης του myArray

a)
```
int k = 6;
double s = 0.0;
for (int i=0; i<6; i++)
  s += myArray[i][k];
```

Απάντηση: (v)

b)
```
int k = 3;
double s = 0.0;
for (int i=0; i<6; i++)
  s += myArray[i][k];
```

Απάντηση: (i)

c)
```
int k = 3;
double s = 0.0;
for (int i=0; i<k; ++i)
  s += myArray[j][j];
```

Απάντηση: (v)

d)
```
int k = 6;
double s = 0.0;
for (int i=0; i<6; i++)
  s += myArray[i][k];
```

Απάντηση: (ii)

e)
```
int k = 3;
double s = 0.0;
for (int i=0; i<6; i++)
  s += myArray[i,k];
```

Απάντηση: (v)

---

7) Σε κάποια κλάση υπάρχει η ακόλουθη κλήση μεθόδου:

```
int days = objName.getDays(monthNumber);
```

όπου η μεταβλητή int monthNumber έχει οριστεί νωρίτερα και αντιστοιχεί στο νούμερο κάποιου μήνα (Ιαν. = 1, Φεβρ. = 2, ..., 12). Η μέθοδος επιστρέφει το πλήθος των ημερών του αντίστοιχου μήνα που δέχεται σαν παράμετρο εισόδου.

Να γίνει χρήση του switch για να συμπληρωθεί η μέθοδος. Οι μέρες των μηνών από τον 1$^{ο}$ στο 12$^{ο}$ είναι: 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31.

Απάντηση:

```
public int getDays(int moNo)
{
   switch (moNo)
   {
      case 1:
      case 3:
      case 5:
      case 7:
      case 8:
      case 10:
      case 12: return 31;
      case 2: return 28;
      case 4:
      case 6:
      case 9:
      case 11: return 30;
```

```
      default: return 0;
   }
}
```

---

8) Τι τυπώνουν στην οθόνη οι ακόλουθοι κώδικες;

a)
```
double var1 = 4.4;
int var2 = 3;
double var3 = 2;
int var4 = 2;
var3 = var4*(int)(var1/var3);
System.out.println(var3);
```

Απάντηση: `4.0`

a)
```
double var1 = 4.0;
int var2 = 3;
double var3 = 1;
int var4 = 2;
var3 = var1*var2*var3+var4%var2;
System.out.println(var3);
```

Απάντηση: `14.0`