ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

# ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

## Σημειώσεις
## για τη Γλώσσα Προγραμματισμού
## JAVA

ΙΖΑΜΠΩ ΚΑΡΑΛΗ

ΑΘΗΝΑ – 2014

## Περιεχόμενο

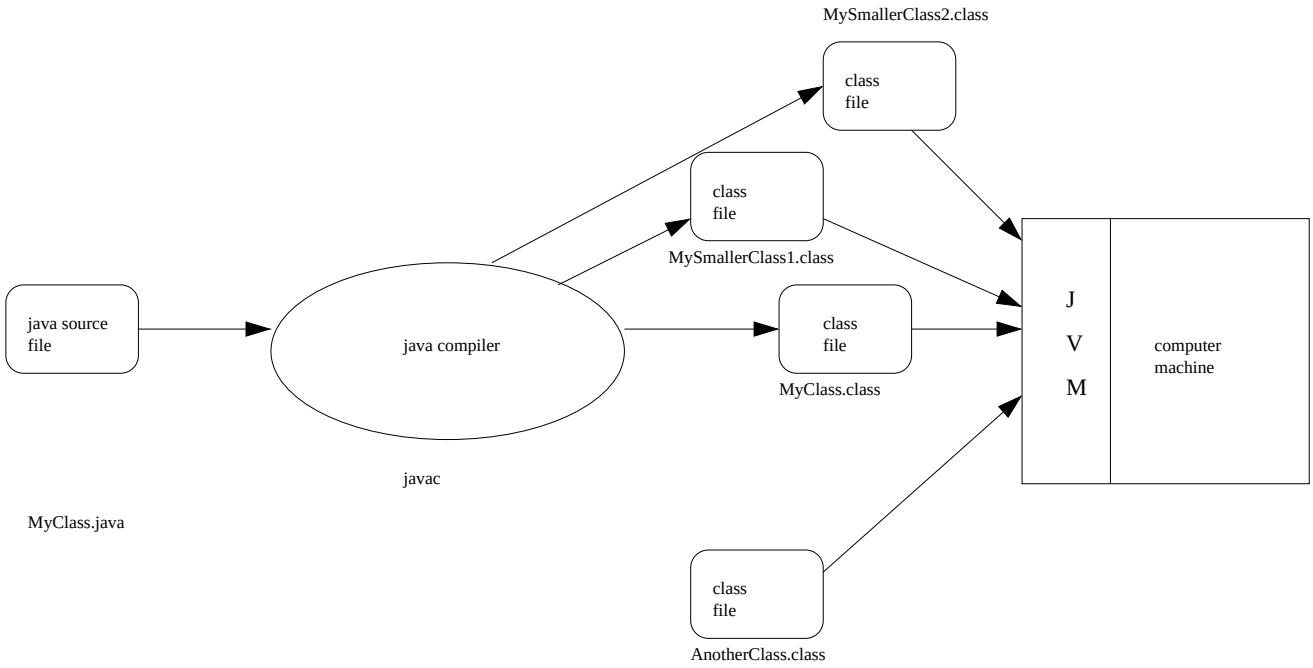
- Η ιστορία και ο στόχος της Java
- Το περιβάλλον εκτέλεσης προγραμμάτων Java
- Δομή και μεταγλώττιση προγραμμάτων Java
- Διαδικαστικός προγραμματισμός με/σε Java
- Αφαίρεση στα δεδομένα σε Java - Δημιουργία / καταστροφή αντικειμένων - Πολυμορφισμός
- Κληρονομικότητα σε Java
- Οργάνωση κώδικα
- Επιπλέον παρατηρήσεις και σχόλια

## Ιστορία και Στόχος της Java

- Αρχή στις αρχές της δεκαετίας ′90
- James Gosling, SUN (για ένα έργο για ευφυείς συσκευές)
- καλωδιακή τηλεόραση → WWW (εκτέλεση από browsers)
- Αρχικά ονομάστηκε Oak
- 1995: η πρώτη έκδοση της Java
- Τώρα διατίθεται από την Oracle
- Νέα τεχνολογία προγραμματισμού για έξυπνες συσκευές (κατανεμημένα περιβάλλοντα - ετερογενείς πλατφόρμες - επικοινωνία μεταξύ προγραμμάτων - επικοινωνία με χρήστη)
- Επιρροές από
  - Σύνταξη: C και C++
  - Όμως λιγότερες δυνατότητες για διαχείριση χαμηλού επιπέδου
  - Σχεδιασμός γλώσσας και αρχιτεκτονικής επηρεασμένος από γλώσσες όπως Eiffel και SmallTalk
  - Περιβάλλον εκτέλεσης: virtual machine - James Gosling για Pascal στα νεανικά του χρόνια.
- *write once, run anywhere*
- Αντικειμενοστραφής προγραμματισμός - τα πάντα οργανωμένα σε κλάσεις

## Το περιβάλλον εκτέλεσης προγραμμάτων Java

- Αρχιτεκτονική




- Αρχεία Java πηγαίου κώδικα (`.java`)
- Τα αρχεία αυτά μεταγλωττίζονται από το μεταγλωττιστή (`javac`) σε αρχεία κλάσεων (class file) (`.class`).
- Κάθε κλάση του πηγαίου κώδικα μεταγλωττίζεται σε ένα αρχείο κλάσης το οποίο περιέχει την αναπαράσταση/πληροφορία για την κλάση αυτή σε *bytecodes* (one-byte opcodes) που είναι εντολές για την *Java Virtual Machine (JVM)*. Το περιεχόμενο ενός αρχείου κλάσης είναι μια ακολουθία από 8-bit bytes

- Η JVM είναι μια αφηρημένη υπολογιστική μηχανή (abstract computing machine).
- Η JVM δε ξέρει τίποτα για Java απλά αναγνωρίζει το συγκεκριμένο δυαδικό format, το class file format.
- Υπάρχουν κι άλλες γλώσσες που μεταγλωττίζονται σε bytecode για να μπορούν να εκτελεστούν στη JVM, π.χ. η Scala.
- Η πλατφόρμα της Java έχει δύο συστατικά: την JVM και το Application Programming Interface (API)
- Το API είναι μια μεγάλη συλλογή έτοιμου λογισμικού που παρέχει πλήθος λειτουργιών. Αποτελεί την core library της γλώσσας. Λογισμικό συναφούς λειτουργικότητας είναι ομαδοποιημένο σε *πακέτα (packages).*
- Integrated Development Environments (IDEs) για Java: π.χ. ECLIPSE, Net Beans.

## Δομή και μεταγλώττιση προγραμμάτων Java

- Κάθε εφαρμογή σε Java πρέπει να έχει μια *κύρια* κλάση.
- Κύρια κλάση είναι μια οποιαδήποτε κλάση που περιέχει μια μέθοδο `main` με την εξής επικεφαλίδα:

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

  Η παράμετρος `args` είναι τύπου `String[]` και περιέχει τα ορίσματα που δίδονται από τη γραμμή εντολής όταν εκτελούμε το πρόγραμμα.
- Κάθε πρόγραμμα Java πρέπει να έχει μια τέτοια μέθοδο `main`. Η κλήση της αποτελεί το αρχικό σημείο της εκτέλεσης του προγράμματος αυτού.

Για να πετύχουμε την ίδια λειτουργικότητα με το παρακάτω C++ πρόγραμμα

```
//cplusplus.cc
#include<iostream>
using namespace std;
int main(){
    cout << "Hello World!" << endl;
}
```

σε Java γράφουμε:

```
// java.java
// The Java code to achieve the same output
// with cplusplus.cc content
class FirstApp {

    public static void main(String[] args) {   //Careful: main should be "public"
        System.out.println("Hello World!");    // otherwise the file compiles
    }                                          // but the program cannot execute
}
```

Αν θέλουμε και είσοδο από τη γραμμή εντολής (και ενδεχόμενα μετατροπή τύπου)

```
class App {
    public static void main(String[] args) {
       // Should check for proper number and type of args though!

       for (String s: args) {   // use the enhanced for statement as args
                                // is array
            System.out.println(s);
            }

// Parse string args to int
// parse<NumType>: convert a String representing a number

       int N = args.length; // length is a public final field of array

       int[] iargs = new int[N];

       for (int i=0; i<N; i++) {
            iargs[i] = Integer.parseInt(args[i]); // Integer.parseInt parses the
                                                  // string argument as a signed
                                                  // decimal integer.
            System.out.println(iargs[i]);
            }
////////////////////////////////////////////////
        System.out.println("////////////////////////////////");

        Employee e1 = new Employee(iargs[0],iargs[1],iargs[2],
                                   iargs[3],iargs[4],iargs[5]);
        e1.print();
        e1.get_population();
        e1.date.print();
        System.out.println(" The e1.population is " +e1.population);
        System.out.println("////////////////////////////////");
          }
}
```

Το πλήρες παράδειγμα βρίσκεται στη σελίδα 54

- Κάθε κλάση μεταγλωττίζεται σε ξεχωριστό αρχείο κλάσης.
- Οι κλάσεις (τύποι) μπορούν να ομαδοποιηθούν σε πακέτα
- Η ορατότητα των κλάσεων μπορεί να περιοριστεί μέσα στο πακέτο ή να είναι public
- Εάν ένα αρχείο περιέχει πολλούς ορισμούς κλάσεων, μόνο μια από αυτές μπορεί να δηλωθεί `public` (και στην περίπτωση που το αρχείο αποτελεί compilation unit ενός πακέτου πρέπει να έχει το ίδιο όνομα με το αρχείο)
- Αν δε δηλωθεί το περιεχόμενο του αρχείου ως μέρος ενός πακέτου (ύπαρξη δήλωσης package στη πρώτη γραμμή μη σχολίου), τότε θεωρείται ότι αυτό είναι μέρος ενός unnamed πακέτου.
- Θεωρούνται ορατά και μπορούν να χρησιμοποιηθούν ονόματα από διάφορα πακέτα του API

## Γενικές έννοιες της Java

- Οργάνωση κώδικα: πακέτα (packages)
- Αντικείμενα, Κλάσεις
- Κληρονομικότητα, Interfaces, Αφηρημένες κλάσεις
- Διαβαθμίσεις ορατότητας:
  - Για τα μέλη
  - Για τους τύπους (κλάσεις)
- Διαβαθμίσεις ορατότητας:
  - Ως προς άλλες κλάσεις
  - Ως προς άλλα πακέτα
- static μέλη
- final ('τελεία και παύλα' !!!)
  - κλάσεις: δεν μπορεί να έχουν υποκλάσεις
  - μέθοδοι: δεν μπορούν να επαναοριστούν στις υποκλάσεις
  - πεδία: είναι σταθερές οι τιμές τους
  - μεταβλητές: είναι σταθερές οι τιμές τους. Στην περίπτωση αναφορών, οι μεταβλητές δεν μπορούν να δείξουν/αναφερθούν κάπου αλλού, τα αντικείμενα όμως στα οποία αναφέρονται μπορεί να αλλάζουν (δεν είναι άμεσο πώς να δηλώνουμε σταθερά αντικείμενα: πρέπει να οριστεί κατάλληλη μη μεταβαλλόμενη κλάση)

## Τύποι στη Java

- Μεταβλητές:
  - (εμβέλειας) κλάσης (static πεδία κλάσης)
  - πεδίο (field) αντικειμένου
  - παράμετρος συνάρτησης
  - τοπικές (local) μεταβλητές συναρτήσεων (μεθόδων).
- Τύποι:
  - primitive data types: `byte` (8-bit signed two's complement integer), `short`, `int`, `long`, `float`, `double`, `boolean`, `char` (16-bit Unicode character)

    ```
    int i;
    ```

  - Πεδία που ο τύπος τους είναι primitive αποκτούν αυτόματα αρχικές τιμές
  - Αναφορές (references)

    ```
    Food fd;
    ```

    όπου `Food` κλάση
  - Τύποι αναφοράς: κλάσεις, interfaces, μεταβλητές τύπων και arrays

Προσοχή: αν οι μεταβλητές δεν είναι μεταβλητές-πεδία τότε δεν αποκτούν αρχικές τιμές. Παράδειγμα:

```
class App {
   public static void main(String[] args) {
        bla();
    }
   static void bla() {
          int i;
          System.out.println("The output of bla() is " + i); // ERROR:
                                                  // variable i not initialized
    }
}
```

## Απλές μεταβλητές και Μεταβλητές Αναφοράς

- Δήλωση μεταβλητής:

  `type t`

  Δηλώνει στο μεταγλωττιστή ότι το `t` πρόκειται να χρησιμοποιηθεί σαν όνομα δεδομένων τύπου `type`.

- Αν πρόκειται για `primitive type` τότε κατά τη δήλωση αυτή δεσμεύεται ο απαιτούμενος χώρος για την μεταβλητή του τύπου.

- Μια μεταβλητή μπορεί να δηλωθεί `final`. Τότε μπορεί να αποκτήσει τιμή άπαξ και πλέον θα περιέχει αυτή την τιμή.

- Αν το `type` είναι κλάση, τότε η τιμή του δεν είναι καθορισμένη με τη δήλωση αυτή. Θα καθοριστεί μόνο όταν δημιουργηθεί ένα αντικείμενο τύπου `type` και της ανατεθεί.

- Αν το `type` είναι κλάση, με την παραπάνω δήλωση ΔΕΝ δημιουργείται αντικείμενο.

- Δημιουργία στιγμιοτύπου/αντικειμένου μιας κλάσης γίνεται με χρήση του τελεστή `new`. Η κλήση του δεσμεύει στη μνήμη τον κατάλληλο χώρο για το αντικείμενο και επιστρέφει μια αναφορά σε αυτό. Επίσης, καλεί την (κατάλληλη) συνάρτηση κατασκευής constructor.

- Δήλωση μεταβλητής αναφοράς και δημιουργίας αντικειμένου ανατεθιμένου σε αυτή:

```
type t = new type(5,10,20);
```

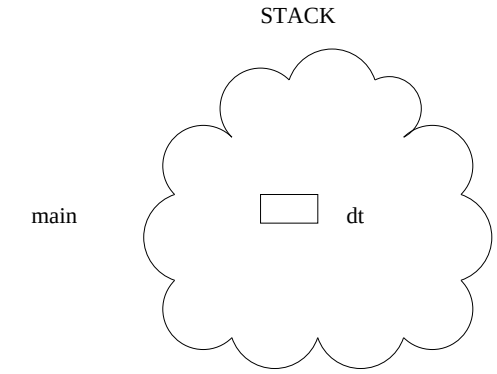
- Μια μεταβλητή τύπου `type` μπορεί να πάρει τιμή μια αναφορά σε ένα στιγμιότυπο του τύπου `type` ή υποκλάσης του ή να έχει την τιμή `null`
- Οι μεταβλητές αναφοράς:
  - Είναι δείκτες σε αντικείμενο του τύπου τους (ή υποκλάσης του) που πάντα γίνονται dereference (όπως οι αναφορές της C++)
  - ΟΜΩΣ δεν είναι σταθεροί, μπορεί να 'δείξουν' σε διαφορετικό αντικείμενο (αντίθετα με τις αναφορές της C++)
  - και μπορεί να μην αναφέρονται σε κανένα αντικείμενο (οπότε ενδέχεται να έχουν τη τιμή `null`)
- Αν μια μεταβλητή που έχει δηλωθεί `final` περιέχει αναφορά σε αντικείμενο, η τιμή (κατάσταση) του αντικειμένου μπορεί να αλλάξει αλλά η μεταβλητή θα αναφέρεται πάντα στο ίδιο αντικείμενο

class Date{ int day; int month; int year}

---

main:

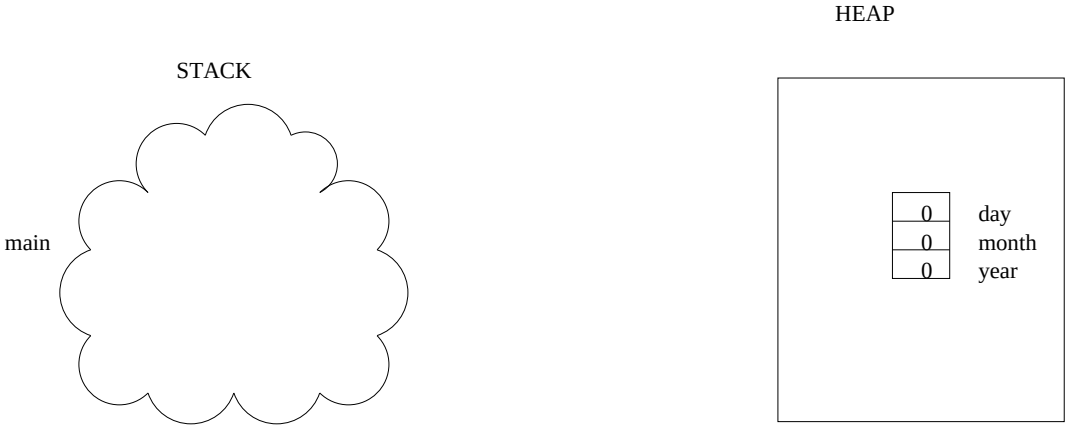
Date dt ;




---

main:

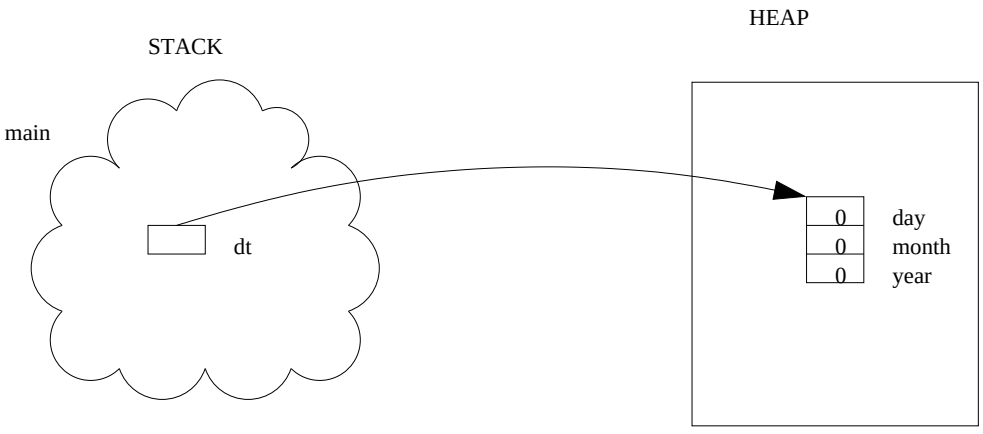
new Date();




---

main:

Date dt = new Date();



## Arrays και Strings

- Arrays: υποκλάση του `Object` που αναπαριστά 'συλλογές' αντικειμένων που περιέχουν σταθερού αριθμού πλήθος τιμών συγκεκριμένου τύπου. Το μήκος τους καθορίζεται από το πλήθος των στοιχείων που συμμετέχουν ή προσδιορίζεται κατά τη δημιουργία τους. Έχουν ειδική αναπαράσταση και χειρισμό από τη JVM.
- String: κλάση που αναπαριστά ακολουθία χαρακτήρων (σταθερή τιμή - όχι array χαρακτήρων).

  Το:

  ```
  String str = "abc";
  ```

  είναι ισοδύναμο με το:

  ```
  char data[] = {'a', 'b', 'c'};
  String str = new String(data);
  ```

## Πώς γίνεται κάποια 'δουλειά'

- Χρήση τελεστών (operators) - κατασκευή εκφράσεων (expressions)
- Χρήση εντολών (statements)
- Χρήση blocks
- Συνοπτικά: à la C
- Προσοχή: όχι δυνατότητα overloading των τελεστών

Πέρασμα παραμέτρων: by value. Για να αλλάξουμε τιμή σε ένα όρισμα primitive τύπου, πρέπει είτε μέσω wrapping στον αντίστοιχο τύπο/κλάση είτε, π.χ.

```
/*
  Primitive data type variables can only be passed by value.
  In case you want a function to change the value of a
  primitive, just return the new value and reassign it
  In more sophisticated cases use wrapper classes
 */

class First {

    public static void main(String[] args) {
        int gear = 1;
        System.out.println(gear);
        gear = change(gear);
        System.out.println(gear);
    }

   static int change(int i) { return ++i ; }
}
```

## Κλάσεις στη Java

- Ορολογία: Πεδία (fields) και Μέθοδοι (methods)
- Class members VS Instance members

## Κατηγορίες Κλάσεων

- Global: δηλώνοντας μια κλάση σαν `public` μπορεί να προσπελαστεί κι έξω από το πακέτο στο οποίο ανήκει
- Nested: ορίζονται μέσα σε κλάσεις οπότε μπορούν να προσδιοριστούν όπως όλα τα άλλα μέλη κλάσης
- `static` μπορεί να προσδιοριστεί κλάση, αν πρόκειται για εσωτερική (nested) κλάση προκειμένου να μπορούν να δημιουργηθούν στιγμιότυπά της με απ΄ ευθείας χρήση της, χωρίς να πρέπει να γίνει μέσω αντικειμένων της εξωτερικής κλάσης (οπότε και να απαιτείται να έχει δημιουργηθεί τέτοιο).
- `abstract`: δε μπορούν να δημιουργηθούν αντικείμενα της κλάσης αυτής και όλες οι μέθοδοί της θεωρούνται abstract (πρέπει να δηλώνονται αμέσως μετά την επικεφαλίδα της κλάσης)
- `final`: δεν επιτρέπεται να οριστούν υποκλάσεις της
- Ανώνυμες
- Τοπικές σε block

## Προσδιορισμοί για τα Μέλη των Κλάσεων

- Τα Πεδία μπορούν να δηλωθούν
  - Ορατότητα:
    - `public`: μπορούν να προσπελαστούν κι έξω από το πακέτο που ορίζεται η κλάση
    - `private`: μπορούν να χρησιμοποιηθούν μόνο από την κλάση μέσα στην οποία ορίζονται
    - `protected`: μπορούν να προσπελαστούν κι από τις υποκλάσεις της κλάσης μέσα στην οποία ορίζονται
    - χωρίς προσδιορισμό: μπορούν να προσπελαστούν μόνο μέσα από το πακέτο που ορίζεται η κλάση
  - Άλλες δηλώσεις:
    - `static`: εμβέλεια κλάσης - όχι απαραίτητο να έχει οριστεί αντικείμενο
    - `final`: δεν μπορεί να αλλάξει άμεσα μετά την αρχικοποίηση
  - ...
- Για τις Μεθόδους:
  - Ορατότητα:
    `public`, `private`, `protected`, χωρίς προσδιορισμό
  - Άλλες δηλώσεις:
    - `static`: δε χρειάζεται να εφαρμοστούν σε αντικείμενο. Δε μπορούν να καλέσουν μεθόδους που δεν είναι `static`.
    - `final`: δε μπορεί να γίνει override
    - `abstract`: δεν έχει υλοποίηση (απλά, δηλώνεται μόνο)
  - ...

## Έννοιες σχετικές με την Αρχικοποίηση Αντικειμένων

- Χρήση constructors: παραδοσιακά. Όμως, μπορούμε μέσα από το σώμα ενός constructor να καλέσουμε έναν άλλο με χρήση του `this`, προσδιορίζοντας μέσω των παραμέτρων ποιόν εννοούμε. Η κλήση πρέπει να γίνει πριν από οποιονδήποτε άλλο κώδικα του constructor.
- Αν δεν έχει οριστεί κάποιος constructor σε μια κλάση, η Java προσφέρει έναν χωρίς ορίσματα, που καλείται default constructor. Αυτός καλεί τον constructor της γονεϊκής κλάσης. Αν η κλάση δεν είναι άμεσα δηλωμένη σαν υποκλάση, καλείται ο constructor της κλάσης `Object`
- Οι constructors καλούνται αυτόματα από την κλήση του τελεστή `new`
- block initialization: προκειμένου να μην επαναλαμβάνουμε κώδικα κοινό για κάθε constructor. Ο μεταγλωττιστής αντιγράφει τον κώδικα του block στην αρχή κάθε constructor.

Όμως, και *αρχικοποίηση πεδίων (initialization)* :

- Αρχικές τιμές στη δήλωση των πεδίων
- final method initialization: Χρήση (final) μεθόδου. Παράδειγμα:

  ```
  varType myVar = initializeInstanceVariable();
  ```

  όπου:

  ```
  protected final varType initializeInstanceVariable()
  ```

  Χρήσιμο αν οι υποκλάσεις θέλουν να χρησιμοποιήσουν τη μέθοδο.

## Παράδειγμα ορισμού μιας απλής κλάσης

```
class App {
    public static void main(String[] args) {
        Dish d = new Dish(100,"giaourti");
    }
}

class Dish{

// fields: package visibility
    int calories;
    String name;

// constructor
    public Dish(int cal, String nam) {
                calories = cal;
                name = nam;
                System.out.println("The new dish named: " + name +
                           " contains " + calories +
                           " calories!");
                }

// methods:
    public void eat() {
           System.out.println("Eating " + name );
           }

    public int get_calories(){
               return calories;
            }
}
```

## ...και ολίγη γεύση προσδιορισμών ορατότητας και overloading στους constructors

```
// Multiple constructors
// Visibility restrictions on Members

class App {
    public static void main(String[] args) {
        Dish d = new Dish();
//        System.out.println(d.name);  // if field is "private" this cases error
        Dish d1 = new Dish("hot dog");
        Dish d2 = new Dish(100, "giaourti");
    }
}

class Dish{
/*
    int calories;
    String name;
*/
/*
  private int calories = 1000;
  private String name = "Junk food";
*/

// Intialization with no default values:
           int calories = 1000;
           String name = "Junk food";


    Dish() {
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
              }

/*
// If default construction with default values needed
   Dish() {
           this(400,"LightFood");
           System.out.println("Again: The new dish named: " + name +
                           " contains " + calories + " calories!");
               }
*/
```

```
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
               }

    void eat() {
           System.out.println("Eating " + name );
           }

    int get_calories(){
               return calories;
            }
}
```

## Παράδειγμα χρήσης block initialization

```
class App {
    public static void main(String[] args) {
        Dish d = new Dish(100,"giaourti");
    }
}


class Dish{

    int calories ;
    String name;

// Block initialization
// in this case NOT WHAT WE WANTED - it is copied in the beginning
   { System.out.println("The new dish named: " + name +
                        " contains " + calories +
                        " calories!"); }

    public Dish(int cal, String nam) {
                calories = cal;
                name = nam;
                }

    public void eat() {
           System.out.println("Eating " + name );
           }

    public int get_calories(){
               return calories;
            }
}
```

## Κληρονομικότητα σε Java

- Η Java προσφέρει μια έτοιμη ιεραρχία κλάσεων
- Όλες οι κλάσεις είναι υποκλάσεις της κλάσης `Object`
- Κλάσεις - Υποκλάσεις (isa → `extends`)
- Αφηρημένες κλάσεις (abstract class- abstract methods)
- Interfaces (static constant fields - method signatures)
- Διαδικασία method dispatch όταν μια μια μη στατική μέθοδος καλείται. Πολυμορφισμός - virtual
- Για τις στατικές μεθόδους πολυμορφισμός δεν υφίσταται διότι δε θεωρείται ότι εφαρμόζονται σε συγκεκριμένο αντικείμενο.

## Interfaces

Ένα interface είναι παρόμοιο με κλάση αλλά

- Τα πεδία του θεωρούνται: public, static και final
- Οι μέθοδοί του θεωρούνται: public και abstract και δίνουμε μόνο την επικεφαλίδα τους κι όχι σώμα (εκτός εν πρόκειται για default).
- Δε μπορούν να οριστούν αντικείμενά τους και χρησιμοποιούνται μόνο για κληρονομικότητα (χρήση και σε πολλαπλή κληρονομικότητα).
- Η λέξη-κλειδί `implements` χρησιμοποιείται από μια κλάση για να δηλώσει ότι 'υλοποιεί' 'ένα' inteface

## Abstract Classes (VS Interfaces)

Abstract κλάσεις:

- Δεν μπορούμε να ορίσουμε αντικείμενά τους
- Μπορούμε να έχουμε μεθόδους που δεν έχουν υλοποίηση αλλά και μεθόδους που έχουν σώμα
- Τα πεδία μπορεί να είναι οποιασδήποτε κατηγορίας (static - non-static, final - not final) και ορατότητας
- Οι μέθοδοι μπορεί να είναι οποιασδήποτε ορατότητας
- Μπορούμε να έχουμε το πολύ μια abstract κλάση σαν υπερκλάση (μια κλάση μπορεί να 'υλοποιεί' οποιονδήποτε αριθμό από interfaces)

## Οι λέξεις-κλειδιά `this` και `super`

`this` αναφέρεται στο τρέχον αντικείμενο

- στο σώμα constructor: για κλήση άλλου constructor της ίδιας κλάσης
- στο σώμα (μη στατικών) συναρτήσεων: για διάκριση του αντικειμένου

`super` αναφέρεται στο τρέχον αντικείμενο, βλέποντάς το σα στιγμιότυπο της υπερκλάσης της κλάσης στην οποία ανήκει

- στο σώμα constructor: για κλήση constructor της γονεϊκής κλάσης
- για πρόσβαση σε μέλη της γονεϊκής κλάσης που έχουν γίνει override

Δε μπορούν να χρησιμοποιηθούν από στατικά μέλη

## Δημιουργία και Αρχικοποίηση Αντικειμένου

- Όταν δημιουργείται ένα στιγμιότυπο μιας κλάσης, δεσμεύεται χώρος ώστε να χωράνε όλα τα μη στατικά πεδία (μεταβλητές στιγμιοτύπου - instance variable) τόσο αυτά που δηλώνονται στη κλάση αυτή όσο κι αυτά που δηλώνονται σε όλη την ιεραρχία των υπερκλάσεών της.
- Τα πεδία αυτά αρχικοποιούνται με τις default τιμές του τύπου τους.
- Στη συνέχεια, εκτελείται η παρακάτω διαδικασία
  1. Υπολογισμός και ανάθεση ορισμάτων στις παραμέτρους του constructor
  2. Αν το σώμα του constructor ξεκινά με κλήση constructor της κλάσης του στιγμιοτύπου (χρησιμοποιώντας τη λέξη-κλειδί `this`), εφάρμοσε τη διαδικασία για τον constructor αυτόν από την αρχή και μεταπήδησε το βήμα 5
  3. Αν δεν ισχύει το παραπάνω και η κλάση δεν είναι η `Object`, τότε η εκτέλεση του σώματος του constructor ξεκινά με κλήση constructor της υπερκλάσης (είτε έμμεσα είτε άμεσα χρησιμοποιώντας τη λέξη-κλειδί `super`) ξεκινώντας τη διαδικασία αρχικοποίησης γι΄ αυτόν τον constructor από την αρχή. Κατόπιν, ο έλεγχος συνεχίζει στο βήμα 4
  4. Εκτελούνται οι αρχικοποιήσεις των μεταβλητών στιγμιοτύπου και οι συναρτήσεις αρχικοποίησης για τις μεταβλητές του στιγμιοτύπου (μεταβλητές στιγμιοτύπου = μη στατικά πεδία του αντικειμένου)
  5. Εκτελείται το υπόλοιπο σώμα του constructor

- Μετά την παραπάνω διαδικασία αρχικοποίησης, επιστρέφεται η αναφορά στο αντικείμενο αυτό

# Αρχικοποίηση υπό κληρονομικότητα - χρήση super. Abstract κλάσεις

```
class App {
    public static void main(String[] args) {
//        Dish d = new Dish();   // Error if Dish is declared abstract
        MainDish md = new MainDish(600,"mbrizola", 15);
    }
}


// abstract class Dish{
// INTERCHANGE WITH THE FOLLOWING LINE (Uncomment the declaration of d in main)
class Dish{
           int calories = 1000;
           String name = "Junk food";
    Dish() {
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
               }

    void eat() {
           System.out.println("Eating " + name );
           }
    int get_calories(){
               return calories;
            }

}
```

```
class MainDish extends Dish{
      int cost = 10;
      MainDish(int cal, String nam, int cst) {
         super(cal, nam);
         cost = cst;
         System.out.println("The cost of the main dish is " + cost);
      }
}
```

## Αρχικοποιήσεις σε κληρονομικότητα και άμεση ανάθεση τιμών στα κληρονομημένα πεδία

```
// Superclass initialization


class App {
    public static void main(String[] args) {
//        Dish d = new Dish();
        MainDish ml = new MainDish(600,"mbrizola", 15);
//        new MainDish(30,"leftover",0);
          }
}


class Dish{
           int calories = 1000;       // package visibility
           String name = "Junk food"; // package visibility
    Dish() {
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
               }
    void eat() {
           System.out.println("Eating " + name );
           }
    int get_calories(){
               return calories;
            }
}
```

```
class MainDish extends Dish{
      int cost = 10;
      MainDish(int cal, String nam, int cst) {
//          super(cal, nam);
         calories = cal;
         name = nam;
         cost = cst;
         System.out.println("The new maindish named: " + name +
                            " contains " + calories + " calories!");
         System.out.println("The cost of the main dish is " + cost);
      }
     int get_cost() {return cost;}
}
```

# Αρχικοποίηση υπό κληρονομικότητα και σύνθεση

```
class App {
    public static void main(String[] args) {
        Dish d = new Dish();
//        MainDish md = new MainDish(600,"mbrizola", 15);
        AirMeal am = new AirMeal();
    }
}

// abstract class Dish{
class Dish{
           int calories = 1000;
           String name = "Junk food";
    Dish() {
        this(2000,"FatAndSpices");
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
               }
    void eat() {
           System.out.println("Eating " + name );
           }
    int get_calories(){
               return calories;
            }
}
```

```
class MainDish extends Dish{
      int cost = 10;
      MainDish(int cal, String nam, int cst) {
         super(cal, nam);
         cost = cst;
         System.out.println("The cost of the main dish is " + cost);
      }
}



class AirMeal {
               MainDish gg = new MainDish(0,"JustAGlassofWater",0);
}
```

## ...ακόμα λίγη κληρονομικότητα και σύνθεση

```
class App {
    public static void main(String[] args) {
//        Dish d = new Dish();
        Meal ml = new Meal(10, "salata", 7,
                           600,"mbrizola", 15,
                           200, "crema karamele", 4);
          }
}


// abstract class Dish{
class Dish{
           int calories = 1000;
           String name = "Junk food";

    Dish() {
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }

    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
              }

    void eat() {
           System.out.println("Eating " + name );
           }
    int get_calories(){
               return calories;
            }
}
```

```
class Starter extends Dish{
      int cost = 5;
      Starter(int cal, String nam, int cst) {
         super(cal, nam);
         cost = cst;
         System.out.println("The cost of the starter is " + cost);
      }
     int get_cost() {return cost;}
}


class MainDish extends Dish{
      int cost = 10;
      MainDish(int cal, String nam, int cst) {
         super(cal, nam);
         cost = cst;
         System.out.println("The cost of the main dish is " + cost);
      }
     int get_cost() {return cost;}
}


class Dessert extends Dish{
      int cost = 3;
      Dessert(int cal, String nam, int cst) {
         super(cal, nam);
         cost = cst;
         System.out.println("The cost of the dessert is " + cost);
      }
     int get_cost() {return cost;}
}
```

```
class Meal {

     Starter starter;
     MainDish maindish;
     Dessert dessert;
     Meal(int cal1, String nam1, int cst1,
          int cal2, String nam2, int cst2,
          int cal3, String nam3, int cst3) {

          starter = new Starter(cal1,nam1,cst1);
          maindish = new MainDish(cal2,nam2,cst2);
          dessert = new Dessert(cal3,nam3,cst3);

          int energy = starter.get_calories() +
                       maindish.get_calories() + dessert.get_calories();
          int bill = starter.get_cost() + maindish.get_cost() +
                     dessert.get_cost();
          System.out.println("You are about to eat a meal of " + energy +
                             " calories and pay : " + bill + " Euros");
      }
}
```

# Παράδειγμα Χρήσης Interfaces

```
class App {
    public static void main(String[] args) {
        Meat mt = new Meat(600,"mbrizola", 15);
        mt.eat();
        System.out.println(mt.get_calories());
        System.out.println(mt.is_late());
        System.out.println(mt.gives_strength());
    }
}


interface Dish{
    void eat();    // public
    int get_calories();   // public
}

interface MainDish extends Dish{
    int MEALTIME = 12;   // public, static, final
    boolean is_late();
}


interface HealthyDish{
    int LOWCALORIES = 300;
    int gives_strength();
}
```

```
class Meat implements MainDish, HealthyDish{
      int calories;
      String name;
      int time;
      Meat(int cal, String nam, int t) {
           calories = cal;
           name = nam;
           time = t;
           System.out.println("You are eating " + name +
                              " at " + time +" having " +
                              calories + " calories");
      }

      public void eat() { System.out.println( "Eating " + name); }    // public
      public int get_calories() { return calories; }   // public
      public boolean is_late() { return ((time > MEALTIME )? true : false); }
      public int gives_strength() {return (calories - LOWCALORIES);}
}
```

## Επιτρεπτές και μη επιτρεπτές αναθέσεις αντικειμένων σε μεταβλητές αναφοράς (παράδειγμα)

```
//References to class pointing to objects of subclass and vice versa

class App {
    public static void main(String[] args) {
        Dish d = new Dish();

//        Dish ff = new MainDish(50,"light",0);
//        MainDish ff = new Dish(50,"light");  //Incompatible types
//        MainDish ff = d;    //Incompatible types

        d.eat();
        Dish md = new MainDish(600,"mbrizola", 15);

//        Bla b = new Bla();
//        md = b;   //Incompatible types

        md.eat();
        md = d;
//        MainDish md1 = md;   //Incompatible types
//        md1 = d;     //Incompatible types
//        md.eat();

        md.eat_more(200);
//        d = md;
//        d.eat();

///// Refs assignment
        MainDish md1 = new MainDish(700,"patates",3);
        MainDish md2 = new MainDish(400,"fakes",1);
        md1 = md2;
        md1.eat_more(3000);
        System.out.println(md1.get_calories());
        System.out.println(md2.get_calories());
    }
}


class Bla {int i;}
```

```
// abstract class Dish{
class Dish{
           int calories = 1000;
           String name = "Junk food";
    Dish() {
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
               }
    void eat() {
           System.out.println("Eating " + name );
           }
    void eat_more(int cal) {
           calories += cal;
           System.out.println("Eating more calories " + calories );
           }
    int get_calories(){
               return calories;
            }
}

class MainDish extends Dish{
      int cost = 10;
      MainDish(int cal, String nam, int cst) {
         super(cal, nam);
         cost = cst;
         System.out.println("The cost of the main dish is " + cost);
      }
    void eat() {
           System.out.println("Eating  Meat " + name );
           }
}
```

Δεν υπάρχει copy constructor, το αποτέλεσμα αυτό το επιτυγχάνουμε υλοποιώντας το Interface `Clonable` ώστε να επαναορίσουμε τη μέθοδο `clone` του `Object`

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

        Dish d = new Dish(20,"salad");
        PersonOrder sallysorder = new PersonOrder(d);
        PersonOrder ladysorder =  (PersonOrder) sallysorder.clone();
         System.out.println(sallysorder.dish ==
                           ladysorder.dish); //not the same Dish object

        Dish d1 = new Dish(400,"cream");
        sallysorder.dish = d1;

        sallysorder.dish.eat();
        ladysorder.dish.eat();  // ladysorder didn't change

        PersonOrder childsorder = new PersonOrder(ladysorder.dish);
         System.out.println(childsorder.dish ==
                           ladysorder.dish); //the same Dish object

        ladysorder.dish = new Dish(600,"IceCream");
        ladysorder.dish.eat();
        childsorder.dish.eat();  // childsorder refers to the old dish
                                 // of ladysorder
         System.out.println(childsorder.dish ==
                           ladysorder.dish); //not the same Dish object any more
          }
}
```

```
class Dish {
           int calories = 1000;
           String name = "Junk food";
    Dish() {
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
               }
    void eat() {
           System.out.println("Eating " + name );
           }
    int get_calories(){
               return calories;
            }
}

class PersonOrder implements Cloneable {
           Dish dish;
    PersonOrder(Dish d){
        dish = d;
        System.out.println("A new PersonOrder is created which is: ");
        dish.eat();
    }

   public Object clone() {
        Dish d = new Dish(this.dish.calories, this.dish.name);
        PersonOrder person = new PersonOrder(d);
        return person;
    }
}
```

## Διαδικασία Αρχικοποίησης για (μεταβλητές) Κλάση(ς) και Interfaces

- Η αρχικοποίηση μιας κλάσης έγκειται στο να εκτελεστούν οι `static block` διαδικασίες αρχικοποίησης και να γίνουν οι αρχικοποιήσεις των στατικών πεδίων της (στατικά πεδία = μεταβλητές (εμβέλειας) κλάσης)
- Η αρχικοποίηση ενός Interface έγκειται στο να εκτελεστούν οι διαδικασίες αρχικοποίησης για τα πεδία του Interface
- Πριν γίνει η αρχικοποίηση μιας κλάσης, γίνεται η αρχικοποίηση της άμεσης υπερκλάσης της (αλλά όχι των Interfaces που υλοποιεί)
- Επίσης, όταν γίνεται αρχικοποίηση ενός Interface, δεν γίνεται αρχικοποίηση των superinterfaces του
- Η αρχικοποίηση κλάσεων ή Interfaces γίνεται αμέσως πριν συμβεί κάποιο από τα εξής (το πρώτο χρονικά εξ αυτών):
  - Αν πρόκειται για κλάση, δημιουργία στιγμιοτύπου της.
  - Αν πρόκειται για κλάση, καλείται μια static μέθοδος της
  - Γίνεται ανάθεση σε static πεδίο της
  - Ένα static πεδίο της χρησιμοποιείται (εκτός αν πρόκειται για πεδίο primitive τύπου ή τύπου `String`, αρχικοποιημένου σε σταθερά).
  - ...

- Μια αναφορά στατικού πεδίου προκαλεί αρχικοποίηση μόνο της κλάσης η οποία το δηλώνει, ακόμα κι αν η αναφορά γίνεται με χρήση υποκλάσης ή, αν πρόκειται για Interface, κλάσης που το υλοποιεί
- Η διαδικασία αρχικοποίησης για ένα static πεδίο αποτιμάται άπαξ και η ανάθεση γίνεται επίσης άπαξ όταν γίνεται η αρχικοποίηση της κλάσης του
- Στη φάση της εκτέλεσης, πρώτα αρχικοποιούνται τα static πεδία που είναι final και αρχικοποιούνται με απλές εκφράσεις (τυπικά: constant expressions)

## Overriding - Hiding - Shadowing

Στη Java συναντάμε τους όρους αυτούς εννοώντας τα:

- Overriding: μεταξύ μη στατικών μεθόδων που επαναορίζονται σε υποκλάση: το αν καλείται η μέθοδος της υποκλάσης ή της υπερκλάσης εξαρτάται από τον τύπο του αντικειμένου πάνω στο οποίο εφαρμόζεται
- Hiding: μεταξύ στατικών μεθόδων που επαναορίζονται σε υποκλάση: το αν καλείται η μέθοδος της υποκλάσης ή της υπερκλάσης εξαρτάται από τον τύπο της μεταβλητής (κι όχι του αντικειμένου).
- Shadowing: μεταξύ μεταβλητών στενότερης και ευρύτερης εμβέλειας

# static methods hiding: Παράδειγμα

```
// Static not overridden by non-static: only by static and then hiding

class App {
    public static void main(String[] args) {
//        Dish d = new Dish();
        MainDish ml = new MainDish(600,"mbrizola", 15);
        ml.eat();
        Dish.eat();
        MainDish.eat();
          }
}



class Dish{
           int calories = 1000;
           String name = "Junk food";

    Dish() {
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(String nam) {
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
                }
    Dish(int cal, String nam) {
        calories = cal;
        name = nam;
        System.out.println("The new dish named: " + name +
                           " contains " + calories + " calories!");
               }

    static void eat() {                    // static
           System.out.println("Eating " );
           }
    int get_calories(){
               return calories;
            }
}
```

```
class MainDish extends Dish{
      int cost = 10;
      MainDish(int cal, String nam, int cst) {
         super(cal, nam);

         cost = cst;
         System.out.println("The new maindish named: " + name +
                            " contains " + calories + " calories!");
         System.out.println("The cost of the main dish is " + cost);
      }
//    void eat() {   // cannot have a non-static overriden
    static void eat() {
           Dish.eat();
           System.out.println("Eating for lunch" );   // static
           }

     int get_cost() {return cost;}
}
```

# non-static methods overriding - static methods hiding - static block initialization: Παράδειγμα

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

        Person.get_population();
        Person p1 = new Person(2,3,1972);
/*
        p1.print();
        p1.get_population();
*/
        Employee.get_population();

        Employee e1 = new Employee(2,3,1972,1,1,1995);
        e1.print();
        e1.get_population();
        e1.date.print();
        System.out.println(e1.population);

        System.out.println("////////////////////////////////");

        p1 = e1;
        p1.print();
        p1.get_population();
        p1.date.print();
        System.out.println(p1.population);
          }
}


class Date {
      int day;  int month; int year;
    Date(int d, int m, int y){
         day = d; month = m; year = y;
         System.out.println("A new Date was created " +  day + "   " +
                            month + "   " + year);
       }
   void print(){
       System.out.println(" " + +  day + "   " + month + "   " + year);
       }
}
```

```
class Person{
            static int population;    // static
            Date date;
            static { System.out.println("The initial population is: "
                                        + population);}  // static block
      Person(int d, int m, int y) {
            date = new Date(d,m,y);
            population++ ;
            System.out.println("A new person was born in ");
            date.print();
             }
     static int get_population() {       // static
             System.out.println("People population is " + population);
             return population;}
     void print() {
             System.out.println("Printing birth date ");
             date.print();
             System.out.println("Age should be non zero");
             }
}


class Employee extends Person{
            static int population;  // static
            Date date;
      Employee(int d1, int m1, int y1,
                int d2, int m2, int y2) {
              super(d1,m1,y1);
              date = new Date(d2,m2,y2);
              population++ ;
            System.out.println("A new Employee in ");
            date.print();
             }
     static int get_population() {      // static
             System.out.println("Employee population is " + population);
             return population;}
     void print() {
             System.out.println("Printing employment date ");
             date.print();
             System.out.println("Age should be over 18");
             }

}
```

Το παράδειγμα της σελίδας 7 ολοκληρωμένο:

```
class App {
    public static void main(String[] args) {
       for (String s: args) {
            System.out.println(s);
            }
       int N = args.length;
       int[] iargs = new int[N];
       for (int i=0; i<N; i++) {
            iargs[i] = Integer.parseInt(args[i]);
            System.out.println(iargs[i]);
            }
////////////////////////////////////////////////
        System.out.println("////////////////////////////////");

        Employee e1 = new Employee(iargs[0],iargs[1],iargs[2],
                                   iargs[3],iargs[4],iargs[5]);
        e1.print();
        e1.get_population();  // CAREFUL: in the general case avoid
                              // referring to static members
                              // via instances: use class names
        e1.date.print();
        System.out.println(" The e1.population is " +e1.population);
        System.out.println("////////////////////////////////");
          }
}

class Date {
      int day;
      int month;
      int year;

    Date(int d, int m, int y)
       { day = d;
         month = m;
         year = y;
         System.out.println("A new Date was created " +  day + "   " +
                            month + "   " + year);
       }
   void print()
       { System.out.println(" " + +  day + "   " + month + "   " + year);
       }
}
```

```
class Person{
            static int population;
            Date date;
            static { System.out.println("The initial population is: "
                                        + population);}
      Person(int d, int m, int y) {
            date = new Date(d,m,y);
            population++ ;
            System.out.println("A new person was born in ");
            date.print();
             }
     static int get_population() {
             System.out.println("People population is " + population);
             return population;}
     void print() {
             System.out.println("Printing birth date ");
             date.print();
             System.out.println("Age should be non zero");
             }
}


class Employee extends Person{
            static int population;
            Date date;
      Employee(int d1, int m1, int y1,
                int d2, int m2, int y2) {
              super(d1,m1,y1);
              date = new Date(d2,m2,y2);
              population++ ;
            System.out.println("A new Employee in ");
            date.print();
             }
     static int get_population() {
             System.out.println("Employee population is " + population);
             return population;}
     void print() {
             System.out.println("Printing employment date ");
             date.print();
             System.out.println("Age should be over 18");
             }
}
```

## Οργάνωση Κώδικα σε Πακέτα (Packages)

- Κάθε πακέτο έχει το δικό του χώρο ονομάτων τύπων
- Τα μέλη των πακέτων είναι κλάσεις και Interfaces
- Οι τύποι που συνιστούν ένα πακέτο αποκαλούνται *μέλη του πακέτου (package members)*
- *Ένα πακέτο περιλαμβάνει μια ή παραπάνω* μονάδες μεταγλώττισης (compilation units). Κάθε μονάδα μεταγλώττισης έχει πρόσβαση σε όλους τους τύπους του πακέτου στο οποίο ανήκει καθώς και στους `public` τύπους του πακέτου της γλώσσας `java.lang`, που προσφέρει κλάσεις που είναι θεμελιώδεις για τη γλώσσα.
- Μια *μονάδα μεταγλώττισης* αποτελείται από τρία (προαιρετικά) μέρη:
  1. Μια δήλωση πακέτου, που περιλαμβάνει το fully qualified name του πακέτου στο οποίο συμμετέχει η μονάδα, π.χ
     ```
     package p1.p11;
     ```
     Αν παραλείπεται η δήλωση, η μονάδα είναι μέρος του unnamed πακέτου
  2. Δηλώσεις `import` που επιτρέπουν να χρησιμοποιούνται τύποι και στατικά μέλη τους από άλλα πακέτα με το σκέτο όνομά τους (όχι το fully qualified), π.χ.
     ```
     import p1.p11.*;
     ```
  3. Δηλώσεις τύπων και Interfaces
- Για να χρησιμοποιήσουμε ένα `public` μέλος πακέτου εκτός πακέτου, πρέπει να κάνουμε ένα από τα εξής:
  - Είτε να χρησιμοποιήσουμε το fully qualified όνομά του

  - Είτε να το κάνουμε `import` μέσω μιας δήλωσης `import` που το αναφέρει ρητά, οπότε μπορούμε να αναφερθούμε σε αυτό με το σκέτο όνομά του
  - Είτε να κάνουμε `import` μέσω μιας δήλωσης `import` όλα τα ονόματα του πακέτου (με χρήση wildcard)
- Υπάρχει η εκδοχή static import για να κάνουμε import τις σταθερές και τις στατικές μεθόδους κλάσεων ενός πακέτου

## Πακέτα και Αρχεία

- Η οργάνωση των ονομάτων των πακέτων είναι ιεραρχική
- Η οργάνωση των πακέτων ΔΕΝ είναι με την έννοια των δικαιωμάτων πρόσβασης
- Παρόλο που το standard της Java δεν απαιτεί ιεραρχικό σύστημα αρχείων, συνήθως οργανώνουμε τα αρχεία του κώδικά μας ως εξής:
  - Γράφουμε τον κώδικά κάθε κλάσης (τύπου, γενικότερα) σε ένα αρχείο με όνομα το όνομα της κλάσης και κατάληξη `.java`
  - Τοποθετούμε το αρχείο αυτό σε έναν κατάλογο με όνομα το όνομα του πακέτου που ανήκει ο τύπος
  - Αντίστοιχη οργάνωση έχουν και τα αρχεία `.class` που προκύπτουν χωρίς να είναι απαραίτητο να βρίσκονται οι κατάλογοι στα ίδια μονοπάτια
  - Το πλήρες μονοπάτι του καταλόγου κάτω από τον οποίο βρίσκεται η ιεραρχία των `.class` αρχείων λέγεται class path και προσδιορίζεται από τη μεταβλητή περιβάλλοντος/συστήματος `CLASSPATH`

## Παράδειγμα με χρήση πακέτων

```
// directory: ./p1
// file: cp1.java

//class cp1 is public, should be declared in a file named cp1.java
package p1;

public class cp1{
   public static void main(String[] args){
        System.out.println("These tricky packages!!!!!!");
        p1.inner.cp2.fun(); // ERROR:fun() is not public in cp2;
                            // cannot be accessed from outside package
                            // Note that there are two alternative
                            // method headers in that file
        cp2.foo();
        p2.cp2.blee();
         }

  static void bla() {
        System.out.println("This is p1.cp1.bla() executing ");
     }
}




////////////////////////////////////////////////////////////////////////
// directory: ./p1
// file: cp2.java

package p1;

public class cp2{

     public static void bloo() {
            System.out.println("This is p1.cp2.bloo() executing ");
      }

   static void foo(){
        System.out.println("Inside p1.cp2.foo()");
        p1.inner.cp2.fun();
        bla();
         }
```

```
private static void bla() {

     System.out.println("This is p1.cp2.bla() executing ");
     }
}

// package p2; // Not allowed
////////////////////////////////////////////////////////////////////////
// directory: ./p1/inner
// file: cp2.java

package p1.inner;

//import p1;  //Cannot be written like that
import p1.*;  // import (public) types from p1 to use without name qualification

public class cp2{

//   static void fun(){
  public static void fun(){
        System.out.println("This is p1.inner.cp2.fun() executing ");
//        p1.cp1.bla();
//        cp1.bla(); // ERROR: bla() is not public in cp1;
                     // cannot be accessed from outside package
         }
}
////////////////////////////////////////////////////////////////////////
// directory: ./p2
// file: cp2.java

package p2;

import p1.*;

public class cp2{

     public static void blee() {
            System.out.println("This is p2.cp2.blee() executing ");
            p1.cp2.bloo();   // needs full path qualification
            p1.inner.cp2.fun();
      }

}
```

## Άλλα (κρίσιμα) θέματα

- Destructors: `finalize()`
- Για μετατροπή primitive types στους αντίστοιχους τύπους αναφοράς: Autoboxing (Unboxing: το αντίστροφο)
- Annotations
- Threads
- Exceptions
- Generics

## Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον συνεργάτη του μαθήματος, Νίκο Ποθητό, τόσο για τις παρατηρήσεις του στις σημειώσεις αυτές αλλά και για την πάντα πρόθυμη συμβολή του, όλα αυτά τα χρόνια, ώστε να πετύχει το μάθημα το στόχο του: όχι την εκπαίδευση σε μια γλώσσα αντικειμενοστραφούς προγραμματισμού αλλά τη μύηση στις αρχές του δια μέσου τους.

## Βιβλιογραφία Σημειώσεων


- "Oracle Java Tutorials" (`http://docs.oracle.com/javase/tutorial/`)
- James Gosling, Bill Joy, Guy Steel, Gilad Bracha, Alex Buckley, "The Java Language Specification — Java SE 7 Edition", 2013
- Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley, "The Java Virtual Machine Specification — Java SE 7 Edition", 2013


## Εκτενέστερη Βιβλιογραφία


- Bruce Eckel, "Thinking In Java (Fourth Edition)", Prentice Hall, 2006
- Russel Winder, Graham Roberts, "Developing Java Software" (3nd Edition), John Wiley and Sons, 2006
- Eric Roberts, "The Art and Science of Java ", Prentice Hall, 2007
- Ken Arnold, James Gosling, David Holmes, "The Java Programming Language (Fourth Edition)", Addison-Wesley Professional, 2005
- James Gosling, Henry McGilton, "The Java Language Environment A White Paper", May 1996 (http://www.oracle.com/technetwork/java/langenv-140151.html)