ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων

Αντικειμενοστρεφής Προγραμματισμός (Object Oriented Programming)

Εκτέλεση εξωτερικών προγραμμάτων και εντολών – The Java Reflection – Αρχικοποίηση (Initialization)

Παναγιώτης Σφέτσος, PhD

http://aetos.it.teithe.gr/~sfetsos/

sfetsos@it.teithe.gr

Περιεχόμενα Μαθήματος

- **Εκτέλεση εξωτερικών προγραμμάτων και εντολών**
- **The Java Reflection**
- **Αρχικοποίηση (Initialization)**

Εκτέλεση εξωτερικών προγραμμάτων και εντολών

- Από ένα πρόγραμμα Java μπορούμε να εκτελέσουμε ένα εξωτερικό πρόγραμμα π.χ. το Notepad των Windows, κάποιο .class αρχείο, ή κάποια εντολή του DOS, καλώντας το CMD.exe

Η κλάση Runtime της Java:

- Χρησιμοποιείται για να ***επικοινωνεί με το περιβάλλον εκτέλεσης της java (JVM)***
- Παρέχει μεθόδους για την εκτέλεση μιας εντολής ή προγράμματος σε ξεχωριστή διαδικασία (processes), για την λήψη του ποσού της ελεύθερης και συνολικής μνήμης κατά την εκτέλεση του προγράμματος, κλπ.

Μερικές από τις μεθόδους:

Αρ.	Μέθοδος	Περιγραφή
1)	public static Runtime getRuntime()	Επιστρέφει το Runtime Object της κλάσης που εκτελείται
2)	public void exit(int status)	Τερματίζει την τρέχουσα JVM
3)	public Process exec(String command) throws IOException	Εκτελεί την εντολή σε μια ξεχωριστή διαδικασία
4)	public long freeMemory()	Επιστρέφει το ποσό της ελεύθερης μνήμης της JVM
5)	public long totalMemory()	Επιστρέφει το συνολικό ποσό της JVM μνήμης

Η μέθοδος Java Runtime exec()

Η μέθοδος Java Runtime exec()

- Εκτέλεση προγράμματος ή εντολής σε ξεχωριστή διαδικασία. Παράδειγμα η εκτέλεση του notepad:

```
class Runtime1{
 public static void main(String args[])throws Exception{
  Runtime.getRuntime().exec("notepad");
 }
}
```

Η κλάση **Process** της Java *(java.lang.Process)* :

- **Παρέχει μεθόδους** για εκτέλεση είσοδο από την διαδικασία, έξοδο στην διαδικασία, αναμονή για τον τερματισμό της διαδικασίας, έλεγχο της κατάστασης της διαδικασίας και καταστροφής της διαδικασίας.

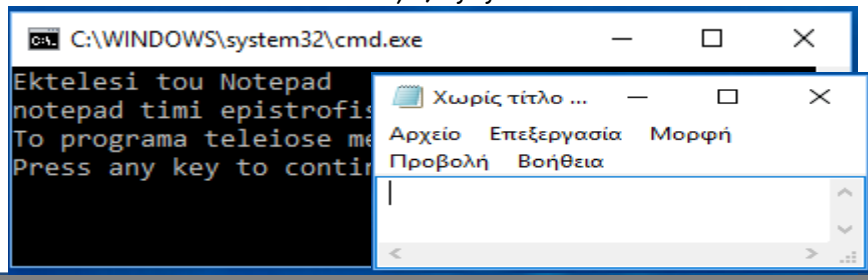
Η κλάση Process και οι μέθοδοι της

- Οι μέθοδοι **Runtime.exec()** και **ProcessBuilder.start()** δημιουργούν μια διαδικασία *(process)* (αντικείμενο της κλάσης) που χρησιμοποιείται για τον έλεγχο της διαδικασίας και την λήψη πληροφοριών για αυτήν.

Αρ.	Μέθοδος	Περιγραφή
1)	abstract void destroy()	Τερματίζει την διαδικασία.
2)	abstract int **exitValue()**	Επιστρέφει την τιμή εξόδου – τερματισμού της διαδικασίας.
3)	abstract InputStream getInputStream()	Επιστρέφει το ρεύμα εισόδου της διαδικασίας (συνδεδεμένη με την default έξοδο).
4)	abstract OutputStream getOutputStream()	Επιστρέφει το ρεύμα εξόδου της διαδικασίας (συνδεδεμένη με την default είσοδο).
5)	abstract int **waitFor()**	Σταματά το τρέχον νήμα και περιμένει μέχρι η διαδικασία να ολοκληρωθεί.
6)	boolean waitFor(long timeout, TimeUnit unit)	Όπως και ανωτέρω, αλλά και με την ολοκλήρωση της χρονικής καθυστέρησης.

Οι μέθοδοι exitValue() και waitFor()

```
class MainClass {
 public static void main(String args[]) {
  Runtime r = Runtime.getRuntime();
  Process p = null;
  String cmd[] = { "notepad"};
  try {
   System.out.println("Ektelesi tou Notepad");
   p = r.exec(cmd);
   p.waitFor();
 }catch (Exception e) {
   System.out.println("Lathos stin ektelesi " + cmd[0]);}
   System.out.println(cmd[0] + " timi epistrofis " +
                                          p.exitValue());
   if(p.exitValue()==0)
    {System.out.println("To programa teleiose me epityhia");
  }else
     System.out.println("Lathos stin ektelesi");}}
```

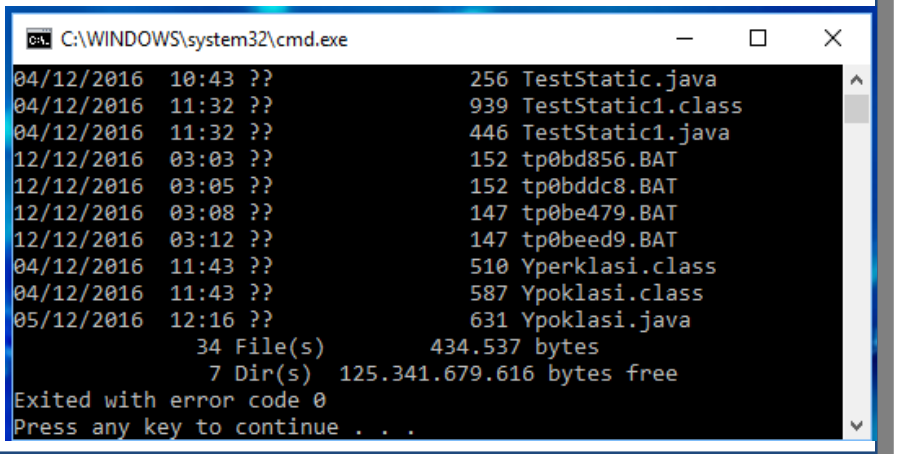


Οι μέθοδοι **waitFor()** και **printStackTrace()**

Παράδειγμα εκτέλεσης εντολής (dir)

```
import java.io.*;
class Main {
  public static void main(String args[]) {
   try {
    Runtime rt = Runtime.getRuntime();
    Process pr = rt.exec("cmd /c dir");
    BufferedReader input = new BufferedReader(new
                        InputStreamReader(pr.getInputStream()));
    String line=null;
    while((line=input.readLine()) != null) {
      System.out.println(line);}
      int exitVal = pr.waitFor();
      System.out.println("Exited with error code "+exitVal);
  } catch(Exception e) {
      System.out.println(e.toString());
      e.printStackTrace();}
}}
```

printStackTrace():
Εκτυπώνει το throwable και όλα τα προηγούμενα εκτελεσθέντα στην στάνταρ έξοδο



Java Reflection *(1/10)*

The Java Reflection:

- Χρησιμοποιείται όταν θέλουμε να **ελέγχουμε** ή να **τροποποιούμε** την συμπεριφορά εκτέλεσης των προγραμμάτων, που εκτελούνται στην JVM.
- Η δυνατότητα μιας γλώσσας **να ελέγχει** και **να καλεί δυναμικά** κλάσεις, αντικείμενα, μεθόδους και χαρακτηριστικά, κατά την εκτέλεση του προγράμματος.
- Βασική χρήση η ***εύρεση πληροφοριών για κάποιο αντικείμενο*** κατά την εκτέλεση του προγράμματος.
- Χρησιμοποιείται πολύ στα εργαλεία ελέγχων κώδικα, debuggers, και στα Integrated Development Environment *(IDE)*, όπως Eclipse, NetBeans, κλπ.
- Το Reflection API χρησιμοποιείται για τον χειρισμό της κλάσης και των μελών της, όπως πεδία, μεθόδους, δομητές, κ.ά. κατά τον **χρόνο εκτέλεσης**.
- Το πακέτο **java.lang.reflect** παρέχει πολλές κλάσεις για να υλοποιηθούν οι μέθοδοι *(reflection java.Methods)* της κλάσης **java.lang.Class**

Java Reflection *(2/10)*

The Java Reflection:

- Η κλάση **Class** δημιουργεί αντικείμενα που περιέχουν ορισμούς και πληροφορίες των κλάσεων κατά την εκτέλεση του προγράμματος.
- Χρήσιμη για την Java καθώς έχει το πλεονέκτημα του **πολυμορφισμού**.
  - ✓ Νέες κλάσεις προστίθενται εύκολα
  - ✓ Συμπεριφορές κληρονομούνται από την υπερκλάση
  - ✓ Καμία επίδραση στις άλλες υποκλάσεις και την υπερκλάση.
  - ✓ Κατά την εκτέλεση, η JVM έχει το πλεονέκτημα της δυναμικής σύνδεσης *(late dynamic binding)*
  - ✓ Τα μηνύματα κατευθύνονται στις σωστές μεθόδους

Java Reflection *(3/10)*

- Κάθε **κλάση που φορτώνεται στην JVM δημιουργεί ένα Class Αντικείμενο**:
  - ✓ Αντίστοιχο στο αρχείο .class
  - ✓ **Με αναφορά** στην κλάση **ClassLoader** που είναι υπεύθυνη για την ***εύρεση και το φόρτωμα της κλάσης στην JVM*** *(πέρα από την δημιουργία του)*.

public abstract class ClassLoader extends Object

- Η ClassLoader ανήκει στο **Java Runtime Environment** *(JRE)* με σκοπό να φορτώνει τα .class αρχεία στην JVM κατόπιν απαίτησης από το πρόγραμμα *(on demand)*.

- Με την **δημιουργία του αντικειμένου:**
  - ✓ Η JVM ελέγχει αν η κλάση έχει ήδη φορτωθεί
  - ✓ Βρίσκει και φορτώνει την κλάση όταν χρειάζεται *(dynamically)*
  - ✓ Με το φόρτωμα της κλάσης δημιουργείται το αντίστοιχο αντικείμενο

Java Reflection *(4/10)*

- Όταν ξεκινά η JVM τρείς τύποι ClassLoader χρησιμοποιούνται, οι παρακάτω:
  - ✓ **Bootstrap** class loader
  - ✓ **Extensions** class loader
  - ✓ **System** class loader

- Ο bootstrap class loader φορτώνει τις βιβλιοθήκες της Java που βρίσκονται στον φάκελο <JAVA_HOME>/jre/**lib**

- Ο extensions class loader φορτώνει τον κώδικα από τον φάκελο: <JAVA_HOME>/jre/lib/**ext**

- Ο system class loader φορτώνει τον κώδικα που βρίσκεται από την **java.class.path**, που αντιστοιχεί στην **CLASSPATH** μεταβλητή.

- Είναι γραμμένοι σε Java και γι' αυτό μπορούμε να γράψουμε τους δικούς μας, χωρίς να γνωρίζουμε τις λεπτομέρειες της JVM. Χρήσιμες για υλοποίηση scripting – γλωσσών, όπως η Jython, για χρήση των κατασκευαστών been, κ.ά.

Java Reflection *(5/10)*

- Συνήθως χρησιμοποιούμενες κλάσεις:
- **java.lang.Class :** Οι κλάσεις και διεπαφές κατά την εκτέλεση του προγράμματος.
- **java.lang.Package :** Οι πληροφορίες για το πακέτο των κλάσεων και διεπαφών.
- **java.lang.ClassLoader :** Παρέχει τους τρείς διαφορετικών τύπων υπηρεσίες φορτώματος *(bootstrap, extension, application)*.

```
public final class Class<T*> extends Object
                    implements Serializable,
                               enericDeclaration, Type,
                               AnnotatedElement
```

(* ο τύπος του αντικειμένου της κλάσης, π.χ. String, <?> για άγνωστο τύπο)

Java Reflection *(6/10)*

Συνήθως χρησιμοποιούμενες **μέθοδοι:**

Αρ.	Μέθοδος	Περιγραφή
1)	**static Class<?> forName(String className)**	Επιστρέφει το αντικείμενο της κλάσης ή διεπαφής της className
2)	**T newInstance()**	Δημιουργεί ένα νέο αντικείμενο, τύπου T, της τρέχουσας κλάσης *(.class)*
3)	**boolean isInterface()**	Καθορίζει αν το συγκεκριμένο αντικείμενο αντιπροσωπεύει μια διεπαφή.
4)	**Class<? super T> getSuperclass()**	Επιστρέφει την υπερκλάση της οντότητας (κλάσης, διεπαφής, βασικός τύπος, κλπ.)
5)	**Class<?>[] getInterfaces()**	Καθορίζει τις διεπαφές που υλοποιούνται από την κλάση ή διεπαφής του αντικειμένου.
6)	**Field[] getFields()**	Επιστρέφει ένα πίνακα με τα πεδία του αντικειμένου της κλάσης.
7)	**Constructor<?>[] getConstructors()**	Επιστρέφει ένα πίνακα με τους δομητές του αντικειμένου τη κλάσης.
8)	**Field[] getDeclaredFields()**	Επιστρέφει ένα πίνακα με τα πεδία του αντικειμένου της κλάσης ή διεπαφής.
9)	**Method[] getDeclaredMethods()**	Επιστρέφει ένα πίνακα με τις μεθόδους του αντικειμένου της κλάσης ή διεπαφής.

Java Reflection *(7/10)*

Μερικές λειτουργίες:

- Εμφάνιση της ***κλάσης ενός αντικειμένου***:

```
void printClassName(Object obj) {
  System.out.println("H klasi tou antikeimenou: " + obj +
                     " einai " + obj.getClass().getName());}
```

- Ανασχεδιασμός του κώδικα με κατάργηση Switch/case και πολλαπλών if – εντολών. Για παράδειγμα, έστω ότι έχουμε μια μέθοδο που επιστρέφει το αντικείμενο μιας ιεραρχίας κλάσεων, ανάλογα με το String που περνάμε σαν παράμετρο:

```
public static Shape getShape(String s)
{
    Shape temp = null;
    if (s.equals ("Circle"))
       temp = new Circle ();
    else
        if (s.equals ("Square"))
           temp = new Square ();
    if (s.equals ("Triangle")
     temp = new Triangle();
    else
    // …
     return temp;
    }
```

Java Reflection *(8/10)*

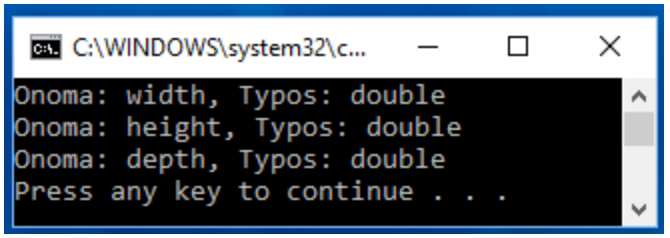
- **Γίνεται:**

```
public static Shape getShape(String s)
{
   Shape temp = null;
   try
   {
    temp = (Shape)Class.forName(s).newInstance();
   }
   catch (Exception e)
   {
   }
   return temp;
}
```

Java Reflection *(9/10)*

- **Παράδειγμα: Εμφάνιση των πεδίων μιας κλάσης (όνομα – τύπος)**

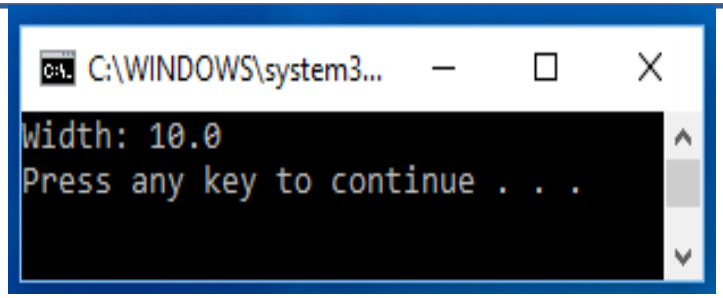
```
import java.lang.reflect.*;
class Box {
 public  double width;
 public  double height;
 public  double depth;}
class EmfanisiPedion {
 public static void main(String[] args) {
 Box b = new Box();
 printFieldNames(b);}
 public static void printFieldNames(Object o) {
  Class c = o.getClass();
  Field[] publicFields = c.getFields();
  for (int i = 0; i < publicFields.length; i++) {
    String fieldName = publicFields[i].getName();
    Class typeClass = publicFields[i].getType();
    String fieldType = typeClass.getName();
    System.out.println("Onoma: " + fieldName + ", Typos: " +
                       fieldType); }}
}
```



Java Reflection *(10/10)*

- **Παράδειγμα: Εμφάνιση πεδίου και της τιμής του**

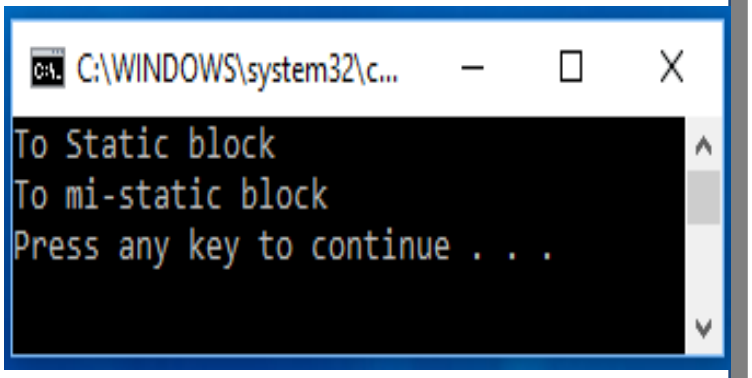
```
import java.lang.reflect.*;
class Box {
 public  double width;
 public  double height;
 public  double depth;
 Box(double w, double h, double d)
 {width = w; height = h; depth = d;}}

 class EmfanisiPediou {
  public static void main(String[] args) {
   Box b = new Box(10, 20, 15);
   printWidth(b); }

 public static void printWidth(Box b) {
  Field w;Double wTimi;
  Class c = b.getClass();
  try {
    w = c.getField("width"); wTimi = (Double)w.get(b);
    System.out.println("Width: " + wTimi.toString());
 }catch (Exception e) {
    System.out.println(e); }}}
```



Αρχικοποίηση Static και μη Static blocks *(1/5)*

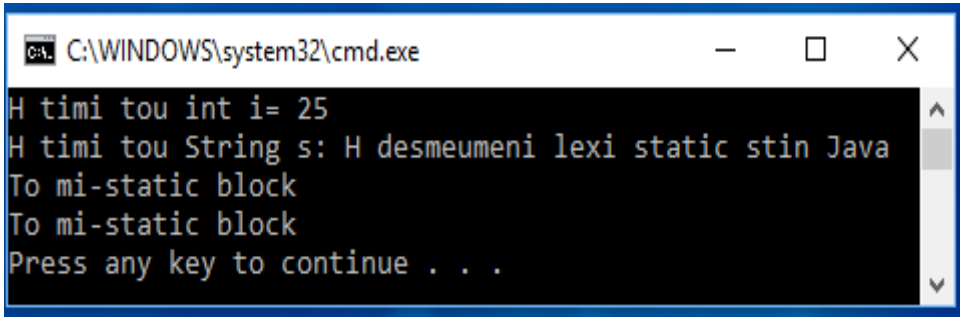
Όταν ορίσουμε ένα **static** και ένα **μη static block** σε μια κλάση, τότε το ***static μπλοκ θα εκτελεστεί πρώτο (μία φορά),*** *με το φόρτωμα της κλάσης,* και μετά το **μη static block** *(της αρχικοποίησης) με την δημιουργία του αντικειμένου)*. Το static block δηλώνεται με την δεσμευμένη λέξη **static.**

```
class TestStatic {
  static //με το φόρτωμα της κλάσης πριν την main()
  {
     System.out.println("To Static block ");
  }
  {   //εκτελείται μετά την δημιουργία του αντικειμένου
     System.out.println("To mi-static block");
  }
  public static void main(String[] args)
  {
    TestStatic t = new TestStatic();
  }}
```



Αρχικοποίηση Static και μη Static blocks *(2/5)*

Παράδειγμα: Το static – block εκτελείται μια φορά – πρώτο, άσχετα με τα αντικείμενα που θα δημιουργηθούν




```
class TestStatic1{
  static int i;
  static String s;
  static{
    i = 25;
    s = "H desmeumeni lexi static stin Java";
    System.out.println("H timi tou int i= "+i);
    System.out.println("H timi tou String s: "+s);}

   {System.out.println("To mi-static block");}

   public static void main(String args[])
   {
      TestStatic1 obj1 = new TestStatic1();
      TestStatic1 obj2 = new TestStatic1();
   }}
```

Αρχικοποίηση Static και μη Static blocks *(3/5)*

Παράδειγμα Αρχικοποίησης:

```
class Static_NoStatic_Seira {
  public static int arxikopoiisi_1 =
                    arxikopoiisi(1);
  public int arxikopoiisi_4 =arxikopoiisi(4);
  static {arxikopoiisi(2);}
  public Static_NoStatic_Seira()
    {arxikopoiisi(5);}

  public static void main(String args[]) {
    arxikopoiisi(3);
    new Static_NoStatic_Seira();}

  public static int arxikopoiisi(int v) {
    System.out.println("Arxikopoiisi " + v);
    return v;} }
```

Σειρά:

Πριν την main():

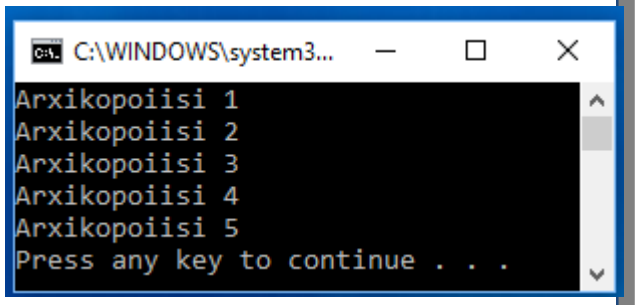
(1) static χαρακτηριστικό

(2) static block

(3) Η main(): (arxikopoiisi(3))

(4) αρχικοποίηση χαρακτηριστικού αντικειμένου

(5) δομητής αντικειμένου



Αρχικοποίηση Static και μη Static blocks *(4/5)*

Με τι σειρά θα εκτελεστούν τα static και μη static *(αρχικοποίησης)* blocks σε μια Κληρονομικότητα;

static block -> Initialization block -> Constructor

```
class Yperklasi {
   public Yperklasi() {
     System.out.println("Domitis Yperklasis");
   }
   static {
    System.out.println("static block - Yperklasis");
   }
   {
    System.out.println("To block arxikopoiisis tis Yperklasis");
   }
}
```

Αρχικοποίηση Static και μη Static blocks *(5/5)*

```
class Ypoklasi extends Yperklasi {
 {
   System.out.println("To block arxikopoiisis tis Ypoklasis");
 }
 static {
   System.out.println("static block - Ypoklasis");
 }

 public Ypoklasi() {
   System.out.println("Domitis Ypoklasis"); }

 public static void main(String[] args) {
   new Ypoklasi();}}
```

Απάντηση - Η σειρά εκτέλεσης στην Κληρονομικότητα:

(1) Τα static blocks των κλάσεων της κληρονομικότητας
(2) Ο δομητής της υποκλάσης καλεί την υπερκλάση, έτσι εκτελείται **πρώτα η αρχικοποίηση** και **μετά ο δομητής** της υπερκλάσης.
(3) Τέλος η αρχικοποίηση και ο δομητής της υποκλάσης