# ΠΛΗΡΟΦΟΡΙΑ ΤΥΠΟΥ

## Type Information

# Δυναμική Πληροφορία Τύπου

ΕΠΛ233

- Runtime type information (RTTI): μας επιτρέπει να ανακαλύπτουμε και να χρησιμοποιούμε δυναμικά (at run time) πληροφορίες για τον τύπο των αντικειμένων.
- Η JAVA μας επιτρέπει να ανακαλύπτουμε δυναμικά πληροφορίες σχετικές με τον τύπο αντικειμένων και κλάσεων, με δύο τρόπους:
  - Την "παραδοσιακή" RTTI, όπου υποτίθεται ότι όλοι οι τύποι είναι διαθέσιμοι και γνωστοί ήδη από την στιγμή της μεταγλώττισης
  - Τον μηχανισμό "ενδοσκόπησης" (reflection), ο οποίος μας επιτρέπει να ανακαλύπτουμε και να χρησιμοποιούμε πληροφορία κλάσης/τύπου στο χρόνο εκτέλεσης (δυναμικά)

```
import java.util.*;
abstract class Shape {
 void draw() { System.out.println(this + ".draw()"); }
 abstract public String toString();
}
class Circle extends Shape {
 public String toString() { return "Circle"; }
}
class Square extends Shape {
 public String toString() { return " Square"; }
}
class Triangle extends Shape {
 public String toString() { return " Triangle"; }
}
public class Shapes {
 public static void main(String[] args) {
  List<Shape> shapeList = Arrays.asList(
   new Circle(), new Square(), new Triangle);
   for (Shape shape: shapeList)
      shape.draw();
   }}
```

# Τι τρέχει;

ΕΠΛ233

```
List<Shape> shapeList = Arrays.asList(
          new Circle(), new Square(), new Triangle);
```

- Αναβάθμιση (upcasting) σε Shape. Εδώ χάνουμε την πληροφορία του εξειδικευμενου τύπου των αντικειμένων που μπαίνουν στη λίστα.

```
for (Shape shape: shapeList)
                shape.draw();
```

- Η λίστα κρατάει τα περιεχόμενά της σαν Object. Όταν εξάγουμε ένα αντικείμενο από τη λίστα, αναβαθμίζεται αυτόματα σε Shape.  Εδώ συμβαίνει RTTI καθώς ελέγχεται η συμβατότητα της αναβάθμισης δυναμικά (δηλ. ότι το Object που εξήγαμε είναι Shape.
- Στη συνεχεία συβαίνει ο πολυμορφισμός, ώστε να κληθεί η κατάλληλη Draw.

# Τα αντικείμενα Class

ΕΠΛ233

- Η δυναμική απεικόνιση των τύπων στη JAVA γίνεται χάρη στα αντικείμενα τύπου **Class**.
- Για κάθε κλάση ενός προγράμματος JAVA υπάρχει ένα αντίστοιχο αντικείμενο τύπου Class, το οποίο ενσωματώνει πληροφορίες για την κλάση-τύπο του.
- Κάθε φορά που γράφουμε και μεταγλωττίζουμε μια κλάση JAVA, δημιουργείται ένα αντικείμενο Class, το οποίο αποθηκεύεται σε αρχείο .class
- Για να κατασκευάσουμε ένα αντικείμενο, ο JVM, μέσω του οποίου τρέχει το πρόγραμμά μας, χρησιμοποιεί το υποσύστημα classloader για να φορτώσει δυναμικά το αντικείμενο της αντίστοιχης κλάσης.
- Επομένως, ένα πρόγραμμα JAVA δεν είναι πλήρως φορτωμένο στη μνήμη πριν ξεκινήσει να τρέχει, αλλά τμήματά του φορτώνονται όποτε χρειαστεί.

# Φόρτωση αντικειμένου κλάσης

ΕΠΛ233

- Ο classloader:
  - ελέγχει πρώτα αν το αντικείμενο Class για τον τύπο του αντικειμένου που κατασκευάζεται είναι ήδη φορτωμένο.
  - Αν όχι, αναζητεί το αντίστοιχο .class αρχείο στο σύστημα αρχείων (συμβουλευόμενο και το classpath)
  - Καθώς διαβάζει το αρχείο, επικυρώνει (verifies) το περιεχόμενό του ώστε να επιβεβαιώσει ότι δεν πρόκειται για παραποιημένο κώδικα (για λόγους ασφάλειας).
- Από τη στιγμή που το αντικείμενο Class έχει ανασυσταθεί στην μνήμη του JVM, χρησιμοποιείται για να δημιουργηθούν αντικείμενα αυτού του τύπου.

```
class Candy {
  static {
    System.out.println("Loading Candy");
  }
}

class Gum {
  static {
    System.out.println("Loading Gum");
  }
}

class Cookie {
  static {
    System.out.println("Loading Cookie");
  }
}
```

```
public class SweetShop {

  public static void main(String[] args) {
    System.out.println("inside main");
    new Candy();
    System.out.println("After creating Candy");
    try {
      Class.forName("Gum");
    } catch(ClassNotFoundException e) {
      e.printStackTrace(System.err);
    }
    System.out.println(
      "After Class.forName(\"Gum\")");
    new Cookie();
    System.out.println("After creating Cookie");
  }
}
```

```
public class SweetShop {

  public static void main(String[] args) {
    System.out.println("inside main");
    new Candy();
    System.out.println("After creating Candy");
    try {
      Class.forName("Gum");
    } catch(ClassNotFoundException e) {
      e.printStackTrace(System.err);
    }
    System.out.println(
      "After Class.forName(\"Gum\")");
    new Cookie();
    System.out.println("After creating Cookie");
  }
}
```

```
inside main
Loading Candy
After creating Candy
Loading Gum
After Class.forName("Gum")
Loading Cookie
After creating Cookie
```

# Πρόσβαση σε αντικείμενα Class

ΕΠΛ233

- Τα αντικείμενα τύπου Class, συμπεριφέρονται όπως όλα τα αντικείμενα της JAVA
- Αποκτώντας χειριστήριο σε ένα αντικείμενο Class μπορούμε να έχουμε δυναμική πρόσβαση σε πληροφορία τύπου (κατά την εκτέλεση του προγράμματός μας):
  - Η πληροφορία τύπου βρίσκεται στο αντικείμενο Class.
- Απόκτηση πρόσβασης σε αντικείμενα Class:
  - Class.forName("*sought_class_name*")
  - this.getClass() - αν έχουμε ήδη πρόσβαση σε αντικείμενο της κλάσης
  - *ClassName*.class - "κυριολέκτημα κλάσης" (class literal):ελέγχεται τη στιγμή της μεταγλώττισης για το αν υπάρχει το ClassName)

```
package typeinfo.toys;
interface HasBatteries {}
interface Waterproof {}
interface Shoots {}

class Toy {
  // Comment out the following default
  // constructor to see
  // NoSuchMethodError from (*1*)
  Toy() {}
  Toy(int i) {}
}

class FancyToy extends Toy  implements
        HasBatteries, Waterproof, Shoots {

  FancyToy() { super(1); }

}
```

```
public class ToyTest {

  static void printInfo(Class cc) {
      System.out.println("Class name: " + cc.getName() +
        " is interface? [" +  cc.isInterface() + "]");
      System.out.println(“Simple name: ” + cc.getSimpleName());
      System.out.println(“Canonical name: ” + cc.getCanonicalName());
    }
  }
  public static void main(String[] args) {
      Class c = null;
      try {
        c = Class.forName("typeinfo.toys.FancyToy");
       } catch(ClassNotFoundException e) { System.exit(1); }

      printInfo(c);

      for (Class face:  c.getInterfaces())
            printInfo(face);
```

```
public class ToyTest {

  static void printInfo(Class cc) {
     System.out.println("Class name: " + cc.getName() +
       " is interface? [" +  cc.isInterface() + "]");
     System.out.println(“Simple name: ” + cc.getSimpleName());
     System.out.println(“Canonical name: ” + cc.getCanonicalName());
   }
  }
  public static void main(String[] args) {
     Class c = null;
     try {
       c = Class.forName("typeinfo.toys.FancyToy");
      } catch(ClassNotFoundException e) { System.exit(1); }

     printInfo(c);

     for (Class face:  c.getInterfaces())
          printInfo(face);
```

Επιστρέφει πίνακα αντικειμένων Class για τις διαπροσωπείες που περιέχονται στο αντικείμενο κλάσης

```
Class up = c.getSuperclass();
Object obj = null;
try {
      // Requires default constructor:
      obj = up.newInstance(); // (*1*)
    } catch(InstantiationException e) {
            System.out.println("Cannot instantiate");
            System.exit(1);
    } catch(IllegalAccessException e) {
             System.out.println("Cannot access");
             System.exit(1);
     }
    printInfo(obj.getClass());
  }
}
```

Class name: typeinfo.toys.FancyToy is interface? [false]
Simple name: FancyToy
Canonical name: typeinfo.toys.FancyToy
Class name: typeinfo.toys.HasBatteries is interface? [true]
Simple name: HasBatteries
Canonical name: typeinfo.toys. HasBatteries
Class name: typeinfo.toys.Waterproof is interface? [true]
Simple name: Waterproof
Canonical name: typeinfo.toys.Waterproof
Class name: typeinfo.toys.Shoots is interface? [true]
Simple name: Shoots
Canonical name: typeinfo.toys.Shoots
Class name: typeinfo.toys.Toy is interface? [false]
Simple name: Toy
Canonical name: typeinfo.toys.Toy

# Κυριολεκτήματα Κλάσεων

ΕΠΛ233

- Δημιουργία χειριστηρίου σε αντικείμενο Class με χρήση κυριολεκτήματος κλάσης (class literal). Π.χ.: FancyToy.class
- Στην περίπτωση δημιουργίας χειριστηρίου σε αντικείμενο τύπου Class με χρήση κυριολεκτήματος, **δεν** αρχικοποιείται απαραίτητα το αντικείμενο Class. Βήματα:
  - **Φόρτωση** (loading): από τον class loader - βρίσκει τα bytecodes της κλάσης και δημιουργεί ένα αντικείμενο Class από αυτά.
  - **Σύνδεση** (linking): επικύρωση των bytecodes της κλάσης και δέσμευση μνήμης για στατικά πεδία δεδομένων. Επίλυση αναφορών σε άλλες κλάσεις.
  - **Αρχικοποίηση** (initialization): αρχικοποίηση στατικών μελών (συμπεριλαμβανομένων αρχικοποιητών προγόνων και ιδίας):
    - η αρχικοποίηση καθυστερεί μέχρι να γίνει η πρώτη αναφορά σε στατική μέθοδο ή σε μή στατικά πεδία.

```
import java.util.*;
class Initable {
  static final int staticFinal = 47;
  static final int staticFinal2 =
    ClassInitialization.rand.nextInt(1000);
  static {
    System.out.println("Initializing Initable");
  }
}
class Initable2 {
  static int staticNonFinal = 147;
  static {
    System.out.println("Initializing Initable2");
  }
}
class Initable3 {
  static int staticNonFinal = 74;
  static {
    System.out.println("Initializing Initable3");
  } }
```

```
public class ClassInitialization {
  public static Random rand = new Random(47);
  public static void main(String[] args) throws Exception {
    Class initable = Initable.class;
    System.out.println("After creating Initable ref");
    // Does not trigger initialization:
    System.out.println(Initable.staticFinal);
    // Does trigger initialization:
    System.out.println(Initable.staticFinal2);
    // Does trigger initialization:
    System.out.println(Initable2.staticNonFinal);
    Class initable3 = Class.forName("Initable3");
    System.out.println("After creating Initable3 ref");
    System.out.println(Initable3.staticNonFinal);
  }
}
```

```
After creating Initable ref
47
Initializing Initable
258
Initializing Initable2
147
Initializing Initable3
After creating Initable3 ref
74
```