



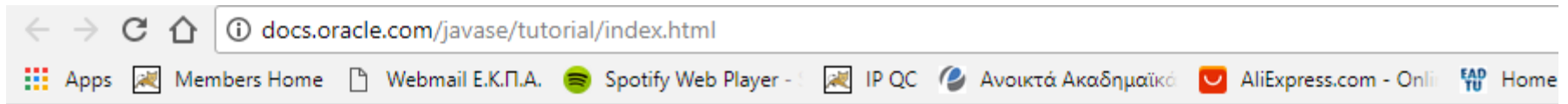
Διάλεξη

Εισαγωγή στη Java, Μέρος Α

1. Τεχνολογία Java
2. Αντικειμενοστραφής προγραμματισμός και συντακτικό της γλώσσας
3. Χρήσιμες κλάσεις
4. Κληρονομικότητα
5. Διεπαφές, Αφηρημένες κλάσεις
6. Πακέτα και εμβέλεια κλάσεων
7. Εργαλεία υλοποίησης



Java tutorial



The Java™ Tutorials

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

The Java Tutorials are practical guides for programmers who want to use the Java programming language to create applications. They include hundreds of complete, w

Trails Covering the Basics

These trails are available in book form as *The Java Tutorial, Sixth Edition*. To buy this book, refer to the box to the right.

- » [Getting Started](#) — An introduction to Java technology and lessons on installing Java development software and using it to create a simple program.
- » [Learning the Java Language](#) — Lessons describing the essential concepts and features of the Java Programming Language.
- » [Essential Java Classes](#) — Lessons on exceptions, basic input/output, concurrency, regular expressions, and the platform environment.
- » [Collections](#) — Lessons on using and extending the Java Collections Framework.
- » [Date-Time APIs](#) — How to use the `java.time` pages to write date and time code.
- » [Deployment](#) — How to package applications and applets using JAR files, and deploy them using Java Web Start and Java Plug-in.
- » [Preparation for Java Programming Language Certification](#) — List of available training and tutorial resources.

□ Τεχνολογία Java

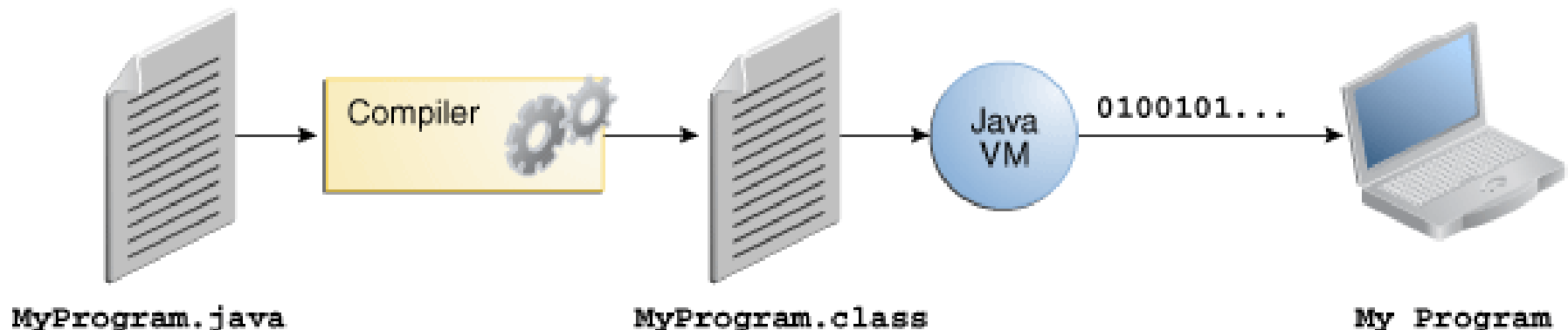
- Αντικειμεστραφής προγραμματισμός και συντακτικό της γλώσσας
- Χρήσιμες κλάσεις
- Κληρονομικότητα
- Διεπαφές, Αφηρημένες Κλάσεις
- Πακέτα και εμβέλεια κλάσεων
- Εργαλεία υλοποίησης

Η τεχνολογία Java (1/3)

- Η Java αποτελεί μια υψηλού επιπέδου γλώσσα προγραμματισμού
- Μερικές από τις λέξεις κλειδιά που την χαρακτηρίζουν:
 - αντικειμενοστρεφής
 - αρχιτεκτονικά ουδέτερη
 - φορητή
 - απλή και εύρωστη
 - ασφαλής
 - Παρέχει έλεγχο πρόσβασης στις κλάσεις που περιγράφουν τα αντικείμενα και τα πεδία των αντικειμένων, κ.α.
- Τα προγράμματα στην Java δομούνται σε ξεχωριστά **.java** αρχεία.

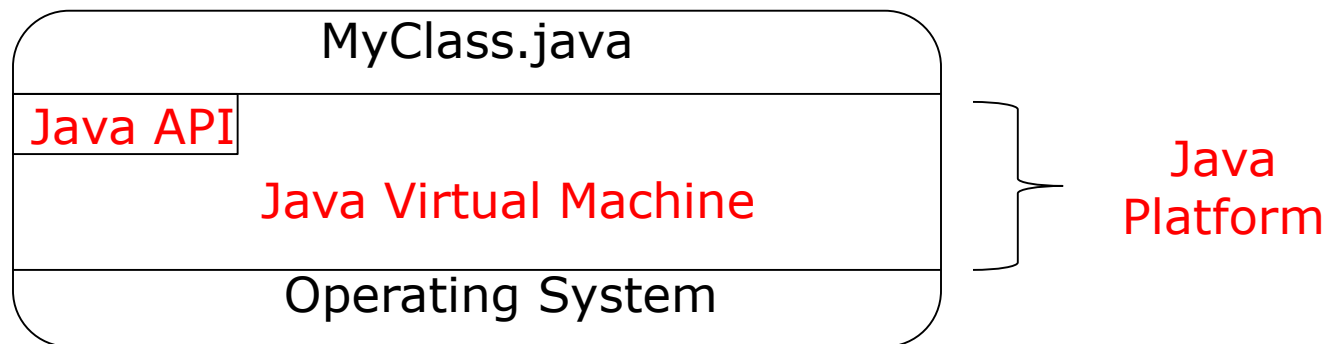
Η τεχνολογία Java (2/3)

- Τα βήματα για να εκτελέσουμε ένα πρόγραμμα σε java
 - Μεταγλώττιση: τα .java αρχεία μεταγλωττίζονται σε .class από τον java compiler (**javac**) και περιέχουν **bytecodes**.
 - Εκτέλεση: η Java Virtual Machine (JVM) διαβάζει τα bytecodes και εκτελεί το πρόγραμμα μας.



Η τεχνολογία Java (3/3)

- Η Java VM:
 - Εξαρτάται από το λειτουργικό σύστημα που χρησιμοποιούμε
 - Επιτρέπει το ίδιο .class αρχείο να μπορεί να εκτελεστεί σε οποιοδήποτε περιβάλλον
- Η Java τεχνολογία, πέρα από την Java VM, μας παρέχει και ένα σύνολο βιβλιοθηκών (Java Application Programming Interface (**Java API**))
 - έτοιμος κώδικας που παρέχει χρήσιμη λειτουργικότητα για τις εφαρμογές μας



-
- Τεχνολογία Java
 - Αντικειμεστραφής προγραμματισμός και συντακτικό της γλώσσας
 - Χρήσιμες κλάσεις
 - Κληρονομικότητα
 - Διεπαφές, Αφηρημένες Κλάσεις
 - Πακέτα και εμβέλεια κλάσεων
 - Εργαλεία υλοποίησης

Βασικές αρχές αντικειμενοστρεφούς προγραμματισμού

- **Προγραμματιστικό μοντέλο** που βασίζεται στην έννοια του **αντικειμένου**
 - Τα πάντα αντιμετωπίζονται ως **αντικείμενα** με συγκεκριμένες **μεθόδους**.

Βασικές αρχές αντικειμενοστρεφούς προγραμματισμού

□ Βασικές αρχές αντικειμενοστρεφούς προγραμματισμού:

- **Κληρονομικότητα** (inheritance): ένα αντικείμενο κληρονομεί και επεκτείνει τα χαρακτηριστικά και τις μεθόδους ενός άλλου αντικειμένου.

□ *Όχημα -> Αυτοκίνητο -> Ηλεκτρικό αυτοκίνητο*

- **Πολυμορφισμός** (polymorphism): Η ίδια μέθοδος μιας κλάσης υλοποιείται με διαφορετικό τρόπο από τις κλάσεις που την κληρονομούν.
- **Ενθυλάκωση** (encapsulation): Η απόκρυψη των χαρακτηριστικών ενός αντικειμένου και η προσπέλαση του μόνο από μια ορισμένη διεπαφή.

Αντικειμενοστραφής προγραμματισμός με Java - Βασικές έννοιες (1)

```
<Access Modifier> class <Class Name> {  
    // field,  
    // constructor,  
    // method declarations  
}
```

- Η έννοια της κλάσης:
 - `public class MyClass{}`
 - Μια κλάση μπορεί να δηλωθεί και ως `private/protected` αρκεί να **εμπεριέχεται σε μια public εξωτερική κλάση** (να είναι δηλαδή inner class).
 - Μια public κλάση αποθηκεύεται σε ένα αρχείο που φέρει το όνομα της, δηλαδή `MyClass.java`
- Τα **πεδία** και οι **μέθοδοι** αποτελούν τα **μέλη μιας κλάσης**

Αντικειμενοστραφής προγραμματισμός με Java - Βασικές έννοιες (2)

- **Μεταβλητές/πεδία:** μοντελοποίηση των χαρακτηριστικών μιας κλάσης
 - **public:** πρόσβαση από όλους
 - **protected:** πρόσβαση από τη κλάση και τις υποκλάσεις της
 - **private:** πρόσβαση μόνο από την ίδια την κλάση
 - **package-private:** πρόσβαση από όλες τις κλάσεις του ίδιου πακέτου
 - **static:** κοινή για όλα τα αντικείμενα μιας κλάσης (στατική θέση στη μνήμη)
 - **final:** τελική τιμή: αρχικοποιείται μία φορά και δεν αλλάζει η τιμή της

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	-	✓	-
no modifier	✓	✓	-	-
private	✓	-	-	-

Αντικειμενοστραφής προγραμματισμός με Java - Βασικές έννοιες (3)

- **Μέθοδοι:** η λειτουργικότητα της κλάσης
 - Δηλώνονται το πως χειρίζονται τα χαρακτηριστικά της (πεδία/μεταβλητές)
 - Πρόσβαση: αντίστοιχη διάκριση με τις μεταβλητές, **public** κτλ

- **Αντικείμενο:** η πραγμάτωση μιας κλάσης
 - Ουσιαστικά το αντικείμενο αποτελεί ένα στιγμιότυπο της κλάσης.
 - `MyClass myObject = new MyClass();`
 - **new** → **τελεστής** για την δέσμευση μνήμης για το αντικείμενο `myObject`.
 - Καλείται ο constructor π.χ. `MyClass();`

Παράδειγμα κλάσης (1/3)

```
public class Bicycle {  
    // the Bicycle class has 3 three fields  
    public int cadence;  
    public int gear;  
    public int speed;  
  
    // the Bicycle class has 1 constructor  
    public Bicycle(int startCadence,  
                   int startSpeed, int startGear) {  
        gear = startGear;  
        cadence = startCadence;  
        speed = startSpeed;  
    }  
  
    // the Bicycle class has 4 methods  
    public void setCadence(int newValue) {  
        cadence = newValue;  
    }  
    public void setGear(int newValue) {  
        gear = newValue;  
    }  
    public void applyBrake(int decrement) {  
        speed -= decrement;  
    }  
    public void speedUp(int increment) {  
        speed += increment;  
    }  
}
```

- Η δήλωση μιας κλάσης ακολουθεί την μορφή

```
<Access Modifier> class <Class Name> {  
    // field, constructor, and method  
    declarations
```

- Το σώμα μιας κλάσης περιλαμβάνει τον κώδικα για τον ορισμό του κύκλου ζωής ενός αντικειμένου μιας κλάσης.
- Οι μεταβλητές μέλη μιας κλάσης ορίζονται και ως τα πεδία μιας κλάσης.
 - Η δήλωση τους ακολουθεί την μορφή:

```
<Access Modifier> <field type> <field name>
```

Παράδειγμα κλάσης (2/3)

```
public class Bicycle {
    // the Bicycle class has 3 three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has 1 constructor
    public Bicycle(int startCadence,
                   int startSpeed,
                   int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }
```

```
// the Bicycle class has 4 methods
public void setCadence(int newValue) {
    cadence = newValue;
}
public void setGear(int newValue) {
    gear = newValue;
}
public void applyBrake(int decrement) {
    speed -= decrement;
}
public void speedUp(int increment) {
    speed += increment;
}
```

□ Constructors:

- Καλούνται κατά την δημιουργία ενός αντικειμένου.
- Μπορούμε να ορίσουμε **κανένα, ένα ή και περισσότερους** constructors για μια κλάση, που θα διαφέρουν στην **λίστα παραμέτρων**.
- Εάν δεν δηλώσουμε εμείς κανένα, τότε η Java μας παρέχει τον default no-argument constructor.
- Η δήλωση τους μοιάζει με αυτή των μεθόδων μόνο που
 1. έχουν σαν όνομα το όνομα της κλάσης
 2. δεν επιστρέφουν κάτι κατά την κλήση τους

Παράδειγμα κλάσης (3/3)

```
public class Bicycle {
    // the Bicycle class has 3 three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has 1 constructor
    public Bicycle(int startCadence,
                   int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has 4 methods
    public void setCadence(int newValue) {
        cadence = newValue;
    }
    public void setGear(int newValue) {
        gear = newValue;
    }
    public void applyBrake(int decrement) {
        speed -= decrement;
    }
    public void speedUp(int increment) {
        speed += increment;
    }
}
```

□ Ορισμός μεθόδων μιας κλάσης

- Method overloading
- Ο ορισμός των μεθόδων ακολουθεί την μορφή

```
<Access Modifier> <return type>
<method name> (<argument list>)
    <exception list>{
        //body
    }
```

Δημιουργία αντικειμένου (1/2)

```
class BicycleDemo {  
    public static void main(String[] args) {  
  
        // Create two different  
        // Bicycle objects  
        Bicycle bike1 = new Bicycle();  
        Bicycle bike2 = new Bicycle();  
  
        // Invoke methods on  
        // those objects  
        bike1.setCadence(50);  
        bike1.speedUp(10);  
        bike1.setGear(2);  
        bike1.printStates();  
  
        bike2.setCadence(50);  
        bike2.speedUp(10);  
        bike2.setGear(2);  
        bike2.setCadence(40);  
        bike2.speedUp(10);  
        bike2.setGear(3);  
        bike2.printStates();  
    }  
}
```

- Η Java VM ξεκινά την εκτέλεση της εφαρμογής από την **main μέθοδο**.
- Η κλάση Bicycle δεν περιέχει στο σώμα της την συνάρτηση main. Γι αυτό το λόγο δεν αποτελεί μια πλήρη εφαρμογή, αλλά ορίζει απλά τον τύπο αντικειμένων.
- Για την δημιουργία ενός αντικειμένου μιας κλάσης χρειάζεται να:
 - **Δηλώσω** μια μεταβλητή τύπου ...Bicycle, π.χ. bike1, bike2
 - **Δεσμεύσω** μνήμη για το αντικείμενο αυτό (χρήση του τελεστή new)
 - **Αρχικοποιήσω** το αντικείμενο αυτό (κλήση ενός constructor της κλάσης) π.χ. Bicycle()

Δημιουργία αντικειμένου (2/2)

```
class BicycleDemo {  
    public static void main(String[] args) {  
  
        // Create two different  
        // Bicycle objects  
        Bicycle bike1 = new Bicycle();  
        Bicycle bike2 = new Bicycle();  
  
        // Invoke methods on  
        // those objects  
        bike1.setCadence(50);  
        bike1.speedUp(10);  
        bike1.setGear(2);  
        bike1.printStates();  
  
        bike2.setCadence(50);  
        bike2.speedUp(10);  
        bike2.setGear(2);  
        bike2.setCadence(40);  
        bike2.speedUp(10);  
        bike2.setGear(3);  
        bike2.printStates();  
    }  
}
```



□ Προσπέλαση των πεδίων ενός αντικειμένου

- Μπορώ να προσπελαύνω απευθείας τα **public** πεδία ενός αντικειμένου μιας κλάσης:

```
objectName.fieldname π.χ.  
bike2.gear = 3;
```

- Κλήση των μεθόδων ενός αντικειμένου (τρέχων παράδειγμα)

- Μπορώ να προσπελαύνω τα **private** πεδία ενός αντικειμένου μιας κλάσης μέσω της κλήσης κατάλληλων μεθόδων:

```
objectName.fieldGetter() /  
objectName.fieldSetter()
```

- Η Java μας επιτρέπει να δημιουργούμε όσα αντικείμενα θέλουμε κατά την εκτέλεση της εφαρμογής μας, χωρίς να πρέπει να αποδεσμεύσουμε ρητά την μνήμη. Το Java runtime environment αποδεσμεύει την μνήμη για όσα αντικείμενα δεν χρησιμοποιούνται πλέον. Αυτή η διεργασία ονομάζεται garbage collection.

ΣΥΝΟΛΙΚΗ ΕΙΚΟΝΑ

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

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
        // those objects
        bike1.setCadence(50);
        bike1.speedUp(10);
        bike1.setGear(2);
        bike1.printStates();

        bike2.setCadence(50);
        bike2.speedUp(10);
        bike2.setGear(2);
        bike2.setCadence(40);
        bike2.speedUp(10);
        bike2.setGear(3);
        bike2.printStates();
    }
}
```

```
public class Bicycle {
    // the Bicycle class has 3 three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has 1 constructor
    public Bicycle(int startCadence,
                   int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has 4 methods
    public void setCadence(int newValue) {
        cadence = newValue;
    }
    public void setGear(int newValue) {
        gear = newValue;
    }
    public void applyBrake(int decrement) {
        speed -= decrement;
    }
    public void speedUp(int increment) {
        speed += increment;
    }
}
```

Παράδειγμα με 2 constructors

```
public class Bicycle {  
    public int gear;  
    private int speed;  
    public static int numberOfBicycles = 0;
```

```
    public Bicycle() {  
        this.gear = -1;  
        this.speed = 0;  
        numberOfBicycles++;  
    }
```

```
    public Bicycle(int speed, int gear) {  
        this.gear = gear;  
        this.speed = speed;  
        numberOfBicycles++;  
    }
```

```
    public void applyBrake(int decrement) {  
        speed -= decrement;  
    }
```

```
    public void speedUp(int increment) {  
        speed += increment;  
    }
```

```
    public int getSpeed() {  
        return speed;  
    }
```

```
public class DemoBicycle {  
    public static void main(String[] args) {  
        Bicycle bike1 = new Bicycle();  
        Bicycle bike2 = new Bicycle(10, 1);  
  
        bike1.speedUp(2);  
        bike1.gear = 2;  
        bike2.applyBrake(2);  
  
        System.out.println("Bike 1 has speed "  
            + bike1.getSpeed() + ", and gear "  
            + bike1.gear);  
        System.out.println("Bike 2 has speed "  
            + bike2.getSpeed() + ", and gear "  
            + bike2.gear);  
        System.out.println("Total number of bicycles is "  
            + Bicycle.numberOfBicycles);  
    }  
}
```

Παράδειγμα προσπέλασης πεδίων αντικειμένου

```
public class DemoBicycle {
    public static void main(String[] args) {
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle(10, 1);

        bike1.speedUp(2);
        bike1.gear = 2;
        bike2.applyBrake(2);

        System.out.println("Bike 1 has speed "
            + bike1.getSpeed() + ", and gear"
            + bike1.gear);
        System.out.println("Bike 2 has speed "
            + bike2.getSpeed() + ", and gear"
            + bike2.gear);
        System.out.println("Total number of bicycles is "
            + Bicycle.numberOfBicycles);
    }
}
```

□ Προσπέλαση των πεδίων ενός αντικειμένου

- Μπορώ να προσπελαύνω απευθείας τα **public** πεδία ενός αντικειμένου μιας κλάσης:

`objectName.fieldName`

- Μπορώ να προσπελαύνω τα **private** πεδία ενός αντικειμένου μιας κλάσης μέσω της κλήσης κατάλληλων μεθόδων:

`objectName.fieldGetter()` /
`objectName.fieldSetter()`

Τι τυπώνει το πρόγραμμά μας;
Ποιός ο ρόλος του τελεστή +
στην `System.out.println()`;

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

- Η **main** μέθοδος απαιτείται ώστε να εκτελεστεί το πρόγραμμα μας.
 - **public**: όλες οι κλάσεις έχουν πρόσβαση σε αυτήν
 - **static**: Ανήκει στην κλάση και όχι στα αντικείμενα της.
 - **void**: Δεν επιστρέφει κάτι.
 - **String[] args**: Λαμβάνει σαν όρισμα έναν πίνακα από συμβολοσειρές. **Τα ορίσματα δίνονται από την γραμμή εντολών.**

- Προφανώς οι παράμετροι εισόδου είναι συμβολοσειρές
 - `Integer.parseInt()`
 - `Double.parseDouble()`
 - Για περισσότερα στο `java.lang` πακέτο του Java API

static πεδία

- Ως **static** μπορούμε να ορίσουμε
 - **πεδία** μιας κλάσης
 - **μεθόδους** μιας κλάσης
- **static πεδία**
 - Είναι πεδία που ανήκουν στην κλάση και όχι στα αντικείμενα της κλάσης.
 - Μπορεί να θεωρηθεί και ως **κοινή μνήμη** που την μοιράζονται όλα τα αντικείμενα (στατική θέση).
 - Τα στατικά πεδία
 - αρχικοποιούνται μόνο μια φορά κατά την έναρξη της εκτέλεσης
 - πριν δημιουργηθεί κάποιο αντικείμενο για την κλάση.
 - Η προσπέλαση ενός static πεδίου γίνεται με την **χρήση του ονόματος της κλάσης και όχι κάποιου αντικειμένου**:

`<class name>.<static field name>` π.χ.

`Bicycle.numberOfBicycles`

static μέθοδοι

- ❑ Είναι μέθοδοι που **ανήκουν στην κλάση** και όχι στα αντικείμενα της.
- ❑ Η κλήση μιας static μεθόδου γίνεται με την χρήση του ονόματος της κλάσης και όχι κάποιου αντικειμένου:

```
<class name>.<static method name>
```
- ❑ Μπορούν να **προσπελάσουν μόνο static πεδία** μιας κλάσης και όχι κάποιο άλλο πεδίο.
- ❑ Μπορούν μόνο να καλέσουν **άλλες static μεθόδους** της κλάσης.
- ❑ Δεν μπορεί να αναφερθούν στα **this & super** keywords.
- ❑ Η **main μέθοδος είναι static για να μπορεί να είναι προσβάσιμη από το περιβάλλον εκτέλεσης χωρίς να χρειάζεται η δημιουργία κάποιου αντικειμένου.**

this

- Η δεσμευμένη λέξη **this** αναφέρεται στο **τρέχον αντικείμενο** μιας κλάσης
 - **this.<field name>**
 - Στην περίπτωση που ένα πεδίο μιας κλάσης αποκρύπτεται από το όνομα μιας παραμέτρου (μέθοδος, constructor)
 - **this.<constructor>**
 - Στο σώμα ενός constructor μπορούμε να καλέσουμε έναν άλλο constructor της κλάσης (explicit constructor invocation).
 - **Για να χειριστούμε το ίδιο το αντικείμενο**
 - Να το περάσουμε **σαν όρισμα** κατά την κλήση μιας συνάρτησης `obj.itIsMe(this);`
 - Για να **το επιστρέψουμε** κατά την κλήση μιας συνάρτησης
`MyClass getInstance() {return this;}`
 - Για να προσπελάσουμε το όνομα της τρέχουσας κλάσης

Τύποι μεταβλητών

□ Η Java υποστηρίζει **8 primitive data types**

(θεμελιώδης τύποι δεδομένων)

1. **byte**: 8-bit ακέραιος (-2^7 έως 2^7-1 , δηλαδή -128 έως 127)
2. **short**: 16-bit ακέραιος (-2^{15} έως $2^{15}-1$, δηλαδή -32768 έως 32767)
3. **int**: 32-bit ακέραιος
4. **long**: 64-bit ακέραιος
5. **float**: 32-bit δεκαδικός
6. **double**: 64-bit δεκαδικός
7. **char**: 16-bit χαρακτήρας (υποστήριξη του Unicode)
8. **boolean**: λογική τιμή (true/false)

Τελεστές

Η Java παρέχει τις ακόλουθες κατηγορίες τελεστών:

- Τελεστές ανάθεσης: `=`
- Αριθμητικοί τελεστές: `+` `-` `*` `/` `%`
- Μοναδιαίοι τελεστές: `++` `--` `!`
- Ισότητας και συσχέτισης: `==` `!=` `>` `>=` `<` `<=`
- Τελεστής σύγκρισης τύπου
 - `objectName instanceof MyClass`
 - Συγκρίνει αν ένα `object` είναι συγκεκριμένου τύπου (επιστρέφει `true/false`)
- Τελεστές για λογικές συνθήκες: `&&` `||`
- Τελεστές για επεξεργασία bits (bitwise): `&` `|` `^` `<<` `>>` `>>>`

Δομές ελέγχου (1)

- Η Java μας παρέχει τις κλασικές δομές ελέγχου:
 - If-then-else, switch, for, while, break, continue, return;
- **if** (condition) { ... }
- **if** (condition) { ... } **else** { ... }
 - Προσοχή!! Το condition είναι τύπου **boolean** με τιμές true ή false. Δεν ισχύει αυτό που γνωρίζουμε από την C, δηλαδή η παράσταση μέσα στο if αν αποτιμάται σε θετικό ακέραιο, τότε είναι αληθής και αν είναι 0 είναι ψευδής
- **switch** (expression){ case value 1: ...; break; default: ...; }
 - Το expression μπορεί να είναι ακέραιος, String ή enumerated type

Δομές ελέγχου (2)

□ **while** (condition){}

- Αν θέλουμε να ορίσουμε infinite loop

- `while(true){...}` και όχι `while(1){...}` όπως ισχύει στην C

□ **for** (initialization; termination; increment){}

- Σε περιπτώσεις που σαρώνουμε arrays ή Collections, μια δυνατή σύνταξη του for loop είναι η ακόλουθη

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};  
  
for (int item : numbers) {  
    System.out.println("Count is: " + item);  
}
```

Πίνακες (1/2)

Ο πίνακας είναι ένα αντικείμενο που διατηρεί ένα σύνολο από αντικείμενα του ίδιου τύπου

- Η αρίθμηση στους πίνακες ξεκινά πάντα από το 0.
- Η **δήλωση ενός πίνακα** γίνεται μέσω του ορισμού του είδους των αντικειμένων που θα φέρει και των [] (αγκύλες)

```
int[] myArray
```

- Ο πίνακας **αρχικοποιείται και δεσμεύεται στη μνήμη** μέσω του τελεστή new

```
int[] myArray = new int[10]
```

- Εναλλακτικά

```
int[] myArray= {100, 200, 300, 400, 500, 600,  
700, 800, 900, 1000}
```

Πίνακες (2/2)

- Το μέγεθος ενός πίνακα δεν μπορεί να μεταβληθεί.
- Στην περίπτωση που ο τύπος των αντικειμένων ενός πίνακα **δεν είναι κάποιος από τους primitive types** της Java, τότε θα πρέπει για κάθε στοιχείο του πίνακα ξεχωριστά να δεσμευτεί μνήμη

```
MyObject[] myArray = new MyObject[10];
```

```
myArray[0] = new MyObject();
```

Πολυδιάστατοι πίνακες (1/2)

- Οι πολυδιάστατοι πίνακες στην Java διαφέρουν από τους αντίστοιχους στην C.
 - Οι 2-διάστατοι πίνακες είναι απλά μονοδιάστατοι πίνακες που φέρουν σαν στοιχεία πίνακες.

```
String[][] names = {{ "Mr.", "Mrs.", "Ms." }, { "Smith", "Jones" }};  
names[0][0] = "Mr.";
```

- Τα πάντα στην Java είναι δείκτες σε μνήμη.
 - Η παρακάτω εντολή **ΔΕΝ** αντιγράφει τον πίνακα names στον πίνακα newNames.

```
String[][] newNames = names;
```
 - Τόσο η μεταβλητή names, όσο και η newNames δείχνουν στον ίδιο χώρο μνήμης (references).

Πολυδιάστατοι πίνακες (2/2)

- Η συνάρτηση **arraycopy** της κλάσης **System** μας επιτρέπει να αντιγράψουμε τα περιεχόμενα ενός πίνακα σε έναν άλλο.

```
System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
```

- Προσοχή! Η μέθοδος αυτή
 - δεν κάνει deep-copy καθώς δεν δημιουργεί νέα αντικείμενα και για τα πεδία του τύπου κλάσης των στοιχείων του πίνακα
 - δεν δουλεύει για Collections
- Ο αριθμός των θέσεων ενός πίνακα δίνεται από το built-in property `length`.

```
myArray.length, names[0].length
```


Παράδειγμα

Τι τυπώνει το πρόγραμμα
μας ;
Τι σημαίνει το System.gc() ;

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

        int[] a = new int[10];

        for(int i = 0; i < a.length; i++) {
            a[i] = i+1;
            System.out.println("My position is " + i + " and i am an element number " + a[i]);
        }

        int[] b = a;
        b[0] = 10;
        System.out.println("b[0] = " + b[0] + " and a[0] = " + a[0]);
        a[0] = 1;

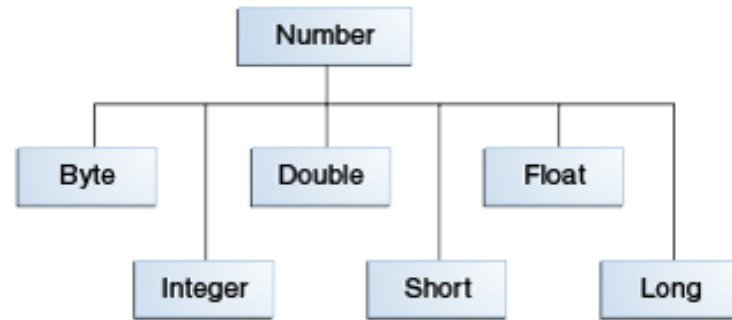
        b = new int[5];
        System.arraycopy(a, 0, b, 0, 5);
        a[0] = 11;
        a[1] = 12;
        for(int i = 0; i < b.length; i++) {
            System.out.print("I am a[" + i + "] = " + a[i] + " -- ")
            System.out.println("I am b[" + i + "] = " + b[i]);
        }

        a = null;
        b = null;
        System.gc();
    }
}
```

-
- Τεχνολογία Java
 - Αντικειμεστραφής προγραμματισμός και συντακτικό της γλώσσας
 - Χρήσιμες κλάσεις
 - Κληρονομικότητα
 - Διεπαφές, Αφηρημένες Κλάσεις
 - Πακέτα και εμβέλεια κλάσεων
 - Εργαλεία υλοποίησης

Number

- Η Java έχει ορίσει την κλάση Number μαζί με ένα σύνολο υποκλάσεων για τον ορισμό αντικειμένων με πεδίο το αντίστοιχο primitive type.



- Στόχος αυτών των κλάσεων είναι να κάνουν **wrap ένα primitive data value σε ένα αντικείμενο**.
- Οι περιπτώσεις που μπορεί να χρειαστούμε κάποιο Number object αντί για ένα primitive:
 - Σαν όρισμα σε μεθόδους που περιμένουν κάποιο object
 - Για να χρησιμοποιήσουμε τις maximum και minimum values των αντίστοιχων data types πχ Integer.MAX_VALUE, Integer.MIN_VALUE.
 - Για την γρήγορη μετατροπή τους από και προς string (θυμήσου την atoi()!!) και άλλους τύπους
- Μια πολύ χρήσιμη κλάση: `java.lang.Math`

String (1)

- Η Java μας παρέχει την κλάση String για να χειριζόμαστε τις συμβολοσειρές σαν αντικείμενα.
- Δημιουργία String αντικειμένων
 - `String greeting = "Hello world!";`
 - `String greeting = new String("Hello world!");`
 - ..και άλλα 11 είδη constructor για να δημιουργήσω ένα αντικείμενο τύπου String
- Η κλάση String είναι immutable, αυτό σημαίνει πως όταν ένα νέο string αντικείμενο δημιουργηθεί δεν μπορεί να αλλάξει.

String (2)

□ Χρήσιμες συναρτήσεις...

- `String palindrome = "Dot saw I was Tod";`

- `int len = palindrome.length();`

- `String name = "My name is ".concat("Rumplestiltskin");`

- Συνένωση string πετυχαίνουμε και με τον τελεστή **+** (θυμήσου τα παραδείγματα με την `System.out.println`)

- `char character = "hello".charAt(0);`

- `"hello".equals("helloooo");`

□ Μετατροπή μεταξύ Number και String

- String to Number

- `<NumberSubclass>.parseXXXX` πχ `Float.parseFloat(myString)`

- Number to String

- `String str = String.valueOf(1);`

Java.lang.System

- Μια χρήσιμη κλάση που μας παρέχει το Java API είναι και η **System**.
- Περιέχει μέσα της ένα σύνολο από static μεθόδους και πεδία.
- Χειρίζεται όλες τις δραστηριότητες που αφορούν την είσοδο και έξοδο του προγράμματος
 - in:"standard" input stream
 - out:"standard" output stream
 - err : "standard" error output stream.
- Μερικές χρήσιμες μέθοδοι
 - **currentTimeMillis()**: επιστρέφει τα δευτερόλεπτα που πέρασαν από την 1/1/1970.
 - **gc()**: αρχικοποιεί τον garbage collector.
 - **exit(int code)**: τερματίζει την εκτέλεση του προγράμματος και επιστρέφει έναν ακέραιο στο run time περιβάλλον.
 - **arraycopy**(Object src, int srcPos, Object dest, int destPos, int length): αντιγραφή μεταξύ πινάκων.

The screenshot shows the Java API documentation for the `System` class. At the top, there are navigation tabs: OVERVIEW, PACKAGE, CLASS (selected), USE, TREE, DEPRECATED, INDEX, and HELP. Below these are links for PREVIOUS CLASS, NEXT CLASS, FRAMES, and NO FRAMES. A summary line indicates the class is NESTED, with links for FIELD, CONSTR, and METHOD, and a detailed view for FIELD, CONSTR, and METHOD. The main content area shows the class name `System` and its inheritance from `Object`. It includes the declaration `public final class System extends Object` and a note that the class cannot be instantiated. A section titled "Field Summary" contains a table with three entries: `err` (standard error output stream), `in` (standard input stream), and `out` (standard output stream). A "Method Summary" section is partially visible at the bottom, showing a table with one entry: `arraycopy` (copies an array from a source to a destination).

Modifier and Type	Field and Description
static <code>PrintStream</code>	<code>err</code> The "standard" error output stream.
static <code>InputStream</code>	<code>in</code> The "standard" input stream.
static <code>PrintStream</code>	<code>out</code> The "standard" output stream.

Modifier and Type	Method and Description
static void	<code>arraycopy</code> (Object src, int srcPos, Object dest, int destPos, int length) Copies an array from the specified source array to the specified destination array.

-
- Τεχνολογία Java
 - Αντικειμεστραφής προγραμματισμός και συντακτικό της γλώσσας
 - Χρήσιμες κλάσεις
 - Κληρονομικότητα
 - Διεπαφές, Αφηρημένες Κλάσεις
 - Πακέτα και εμβέλεια κλάσεων
 - Εργαλεία υλοποίησης

Κληρονομικότητα

- Στην Java μια κλάση μπορεί να κληρονομήσει και να επεκτείνει τα **πεδία** και τις **μεθόδους** μιας άλλης κλάσης.

SuperClass/ParentClass/BaseClass

↓ *extends*

SubClass/ChildClass/ExtendedClass

- Η κλάση **Object** (java.lang.Object) θεωρείται ως η **super class** όλων των κλάσεων.
- Κάθε κλάση που υλοποιούμε και δεν κληρονομεί ρητά κάποια άλλη, θεωρείται υποκλάση της Object.
- Μια κλάση μπορεί να κληρονομήσει **ΜΟΝΟ ΜΙΑ** άλλη κλάση (μία superclass)
- Ο σκοπός της κληρονομικότητας είναι να χρησιμοποιούμε υπάρχουσες κλάσεις και να **προσθέτουμε επιπλέον χαρακτηριστικά** χωρίς να χρειαστεί να ξαναγράψουμε κώδικα.
- Λέξη κλειδί **extends**: SubClass extends SuperClass

Object

- ❑ Βρίσκεται στην **κορυφή της ιεραρχίας** των κλάσεων. Είναι η κλάση που την κληρονομούν έμμεσα ή άμεσα όλες οι κλάσεις στην Java.
- ❑ Η κλάση αυτή ορίζει συμπεριφορά που όλα τα αντικείμενα των κλάσεων πρέπει να διαθέτουν.
- ❑ Μερικές από τις σημαντικότερες μεθόδους της Object που μια κλάση από τον χρήστη μπορεί να κάνει **override** ή και να **χρησιμοποιήσει αυτούσια**:
 - `equals()`: επιστρέφει true ή false ανάλογα αν δυο αντικείμενα είναι όμοια. Με τον όρο όμοια δεν εννοούμε να δείχνουν στο ίδιο αντικείμενο (μνήμη).
 - ❑ Κάνοντας **override** την μέθοδο αυτή μπορούμε να κάνουμε πιο συγκεκριμένα τα κριτήρια για να είναι δυο αντικείμενα όμοια.
 - `getClass()`: μας δίνει ένα runtime representation της κλάσης που ανήκει το εκάστοτε αντικείμενο για να ανακτήσουμε πληροφορίες για την κλάση αυτή, πχ. `myObject.getClass().getName()`
 - `toString()`: μας παρέχει ένα string για την αναπαράσταση ενός object. Συνήθως το κάνουμε override για να παρουσιάσουμε με δικό μας τρόπο τα πεδία μιας κλάσης σε μια εκτύπωση.

Χαρακτηριστικά κληρονομικότητας

- ❑ Μια υποκλάση κληρονομεί όλα τα μέλη μιας υπερκλάσης (**πεδία, μεθόδους, εμφωλευμένες κλάσεις**), αλλά έχει direct access στα **public** και **protected** μέλη της υπερκλάσης.

Constructors

- ❑ Οι constructors δεν θεωρούνται μέλη μιας κλάσης άρα δεν μπορούν να κληρονομηθούν, μπορούν όμως να κληθούν από την υποκλάση.
- ❑ Ο constructor μιας υποκλάσης μπορεί να καλέσει τον constructor της υπερκλάσης είτε έμμεσα ή με την χρήση της **super**.

Πεδία

- ❑ Τα κληρονομημένα πεδία μπορούν να χρησιμοποιηθούν άμεσα όπως οποιοδήποτε άλλο πεδίο της κλάσης.
- ❑ Μπορούμε να ορίσουμε επιπλέον πεδία σε μια υποκλάση που δεν ορίζονται στην υπερκλάση.
- ❑ Μπορούμε να ορίσουμε πεδία στην υποκλάση με ίδιο όνομα με αυτά στην υπερκλάση και να τα αποκρύψουμε (*hiding*).

Χαρακτηριστικά κληρονομικότητας

Μέθοδοι

- ❑ Οι κληρονομημένες μέθοδοι μπορούν να κληθούν από την υποκλάση.
- ❑ Μπορούμε να ορίσουμε νέες μεθόδους σε μια υποκλάση που δεν εμφανίζονται στην υπερκλάση.
- ❑ Μπορούμε να ορίσουμε μεθόδους στην υποκλάση με ίδια υπογραφή με αυτές στην υπερκλάση (**method overriding**).
 - Μπορούμε να καλέσουμε αυτές της υπερκλάσης με τον τελεστή **super**.
- ❑ Μπορούμε να ορίσουμε μια static μέθοδο στην υποκλάση με ίδια υπογραφή με μια static μέθοδο στην υπερκλάση και έτσι να την αποκρύψουμε (hiding).

Παράδειγμα

```
public class MyClass {

    public MyClass () {
    }

    public void myMethod1 () {
        System.out.println("I do nothing");
    }

    public int myMethod2(int x) {
        return x*x;
    }

    public double myMethod3(int y) {
        return Math.sqrt(y);
    }
}
```

```
public class MyClass1 extends MyClass{
    int j = -1;

    public MyClass1 () {
        super ();
        j=2;
    }

    @Override
    public void myMethod1 () {
        System.out.println("I do something");
        j=3;
    }

    @Override
    public int myMethod2(int x) {
        return x*x-j;
    }

    @Override
    public double myMethod3(int y) {
        return Math.sqrt(y)+j;
    }
}
```

Παράδειγμα

Τι τυπώνει το πρόγραμμα μας?
Ποιές αρχές του αντικειμενοστραφούς
προγραμματισμού διακρίνετε?

```
public class Animal {
    public static void testClassMethod() {
        System.out.println("The class" + " method in Animal.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance" + " method in Animal.");
    }
}

public class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The class method" + " ");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method" + " in Cat.");
    }
}

public class Dog extends Animal {
    public static void testClassMethod() {
        System.out.println("The class method" + " in Dog.");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method" + " in Dog.");
    }
}

public class AnimalExample {
    public static void main(String[] args) {
        Cat myCat = new Cat();
        Dog myDog = new Dog();

        Animal myAnimal1 = myCat;
        Animal myAnimal2 = myDog;

        Animal.testClassMethod();

        myAnimal1.testInstanceMethod();
        myAnimal2.testInstanceMethod();
    }
}
```

Method hiding

Method overriding

ΑΠΑΝΤΗΣΗ
The class method in Animal.
The instance method in Cat.
The instance method in Dog.

super

- Υπάρχουν περιπτώσεις στις οποίες θέλουμε να καλέσουμε μια μέθοδο της γονικής κλάσης που κάνουμε override, ή έναν constructor.
- Χρήση της super για την κλήση μεθόδων

```
Class ParentClass {  
    void methodA() {...}  
}  
Class ChildClass extends ParentClass {  
    void methodA(){  
        super.methodA();  
        ....}  
}
```

- Χρήση της super για την κλήση constructors
 - Η κλήση της super(<argument list>); γίνεται πάντα πρώτη.
 - Μπορεί να μην γίνεται ρητά από το πρόγραμμα μας η κλήση της super.
 - Το run time περιβάλλον θα καλέσει τον non-argument constructor (default constructor).
 - Υπάρχει μια αλυσιδωτή κλήση constructors από την κλάση παιδί μέχρι να φτάσουμε στην κλήση του Object constructor (constructor chaining). Στην κληρονομικότητα τα αντικείμενα χτίζονται από την κλάση Object προς τα κάτω.

Type casting

- Ο ορισμός μιας κλάσης μας παρέχει νέους τύπους δεδομένων.
- Ως type casting ορίζουμε την μετατροπή ενός αντικειμένου από έναν αρχικό τύπο σε ένα άλλο.
 - Type casting σε reference του ίδιου τύπου αντικειμένου.
 - **Upcasting**: το να δείχνουμε ένα αντικείμενο μιας κλάσης Child με ένα reference τύπου Parent, όπου Child extends Parent.
 - δεν μπορούμε να καλέσουμε επιπλέον μεθόδους και πεδία που ορίζονται στην Child και όχι στην Parent.
 - Μπορούμε όμως να καλέσουμε τις overridden methods της Child!!
 - **Downcasting**: το να δείχνουμε ένα αντικείμενο της κλάσης Parent με ένα reference τύπου Child. Σε αυτή την περίπτωση θα μας χτυπήσει compile-time error μιας και η μνήμη δεν είναι ίδια. Η μόνη εξαίρεση

```
Parent obj = new Child();  
Child child = (Child) obj;
```

```
Parent obj = new Child();  
if (obj instanceof Child)  
    Child child = (Child) obj
```

Type casting

- **Upcasting**: το να δείχνουμε ένα αντικείμενο μιας κλάσης Child με ένα reference τύπου Parent, όπου Child extends Parent.
- Parent obj = new Child();

- δεν μπορούμε να καλέσουμε τις επιπλέον μεθόδους και πεδία που ορίζονται στην Child και όχι στην Parent.

- Μπορούμε όμως να καλέσουμε τις overridden methods της Child!!

Κοινές ή overridden methods

Extended fields & methods

- Υπάρχει δεσμευμένη μνήμη, δεν υπάρχει πρόσβαση στα επιπλέον

Type casting

- **Downcasting:** το να δείχνουμε ένα αντικείμενο της κλάσης Parent με ένα reference τύπου Child. Σε αυτή την περίπτωση θα μας χτυπήσει compile-time error μιας και η μνήμη δεν είναι ίδια. Η μόνη εξαίρεση

```
Parent obj = new Child();  
Child child = (Child) obj;
```

```
Parent obj = new Child();  
if (obj instanceof Child)  
    Child child = (Child) obj
```

Kοινές ή overridden methods

Extended fields & methods

```
public static void main(String args[]) {           Super obj = new Sub();           Sub sub = (Sub) obj; sub.Sample();           }
```

```
class Super {  
    void Sample() {  
        System.out.println("method of super class");  
    }  
}
```

```
public class Sub extends Super {  
    void Sample() {  
        System.out.println("method of sub class");  
    }  
    public static void main(String args[]) {  
        Super obj =(Super) new Sub(); obj.Sample();           //upcasting  
    }  
}
```

```
public static void main(String args[]) {  
    Super obj = new Sub();  
    Sub sub = (Sub) obj; sub.Sample();           //downcasting  
}  
}
```

```
public class Student {
```

```
private int id;
private String name;
private String surname;
```

Constructors

```
public Student() {
    id = 0;
    name = "empty";
    surname = "empty";
}
```

```
public Student(int id, String name, String surname) {
    this.id = id;
    this.name = name;
    this.surname = surname;
}
```

```
public int getId() {
    return id;
}
```

```
public void setId(int id) {
    this.id = id;
}
```

```
public String getName() {
    return name;
}
```

```
public void setName(String name) {
    this.name = name;
}
```

Πρόσβαση στα private πεδία της κλάσης με getters και setters

```
public String getSurname()
    return surname;
}
```

```
public void setSurname(String surname) {
    this.surname = surname;
}
```

```
@Override
public String toString() {
    String str = "";
    str += "Student id: " + this.id + "\n";
    str += "Student name: " + this.name + "\n";
    str += "Student surname: " + this.surname + "\n";

    return str;
}
```

toString() για να τυπώνουμε πληροφορίες ενός αντικειμένου

```
@Override
public boolean equals(Object obj) {
    boolean bool = false;
```

```
if (!(obj instanceof Student))
    return false;
```

```
Student std = (Student) obj;
if (std.getId() == this.id &&
    std.getName().equals(this.name) &&
    std.getSurname().equals(this.surname))
    bool = true;
```

```
return bool;
}
```

equals() για να συγκρίνουμε δυο αντικείμενα

```
public class StudentDemo {
```

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

```
Student std1 = new Student(123, "Maria", "Papadopoulou");
Student std2 = new Student(456, "Dimitris", "Nikolaou");
```

```
System.out.println("Is std1 equal with std2: " + std1.equals(std2));
```

```
System.out.println(std1.toString());
System.out.println(std2.toString());
}
```

Is std1 equal with std2: false

Student id: 123

Student name: Maria

Student surname: Papadopoulou

Student id: 456

Student name: Dimitris

Student surname: Nikolaou

-
- Τεχνολογία Java
 - Αντικειμεστραφής προγραμματισμός και συντακτικό της γλώσσας
 - Χρήσιμες κλάσεις
 - Κληρονομικότητα
 - **Διεπαφές, Αφηρημένες Κλάσεις**
 - Πακέτα και εμφάνιση κλάσεων
 - Εργαλεία υλοποίησης

Διεπαφές (Interfaces) (1)

- Υπάρχουν περιπτώσεις που θέλουμε να συμφωνήσουμε με τα άλλα μέλη της ομάδας μας στο πως θα επικοινωνούν κομμάτια κώδικα και να μοιράσουμε την υλοποίηση!
- Τα interfaces μας παρέχουν αυτή την δυνατότητα.
- Στην Java ένα interface αποτελεί
 - ένα reference type, παρόμοιο με μια κλάση
 - περιέχει μόνο σταθερές (static & final μεταβλητές)
 - δηλώσεις μεθόδων
 - **Δεν υλοποιείται ΚΑΜΙΑ μέθοδος.**
- Δεν μπορούμε να δημιουργήσουμε αντικείμενα από interfaces.

Διεπαφές (Interfaces) (2)

- Ένα interface μπορεί να γίνει
 - implement από μια κλάση ή
 - extend από ένα άλλο interface.
- Μια κλάση μπορεί να κάνει implement παραπάνω από ένα interfaces (σε αντίθεση με την κληρονομικότητα, όπου μπορεί να κάνει extend μια μόνο κλάση).
- Όταν κάνουμε implement ένα interface, πρέπει να γράψουμε κώδικα για όλες τις μεθόδους που ορίζονται.

Συνολικά

```
public interface MyInterface {  
    public void myMethod1();  
    public int myMethod2(int x);  
    public double myMethod3(int y);  
}
```



implements

```
public class MyClass implements MyInterface{  
  
    public MyClass(){  
    }  
  
    public void myMethod1() {  
        System.out.println("I do nothing");  
    }  
  
    public int myMethod2(int x) {  
        return x*x;  
    }  
  
    public double myMethod3(int y) {  
        return Math.sqrt(y);  
    }  
}
```

extends

```
public class MyClass1 extends MyClass{  
    int j = -1;  
  
    public MyClass1(){  
        super();  
        j=2;  
    }  
  
    @Override  
    public void myMethod1() {  
        System.out.println("I do something");  
        j=3;  
    }  
  
    @Override  
    public int myMethod2(int x) {  
        return x*x-j;  
    }  
  
    @Override  
    public double myMethod3(int y) {  
        return Math.sqrt(y)+j;  
    }  
}
```

Αφηρημένες (Abstract) κλάσεις

- Αποτελούν κλάσεις με μια ή περισσότερες abstract μεθόδους (που δεν έχουν υλοποιηθεί).

```
public abstract class MyAbstractClass {  
  
    // declare fields  
  
    // declare non-abstract methods  
  
    abstract void abstractMethod();  
  
}
```

- Μια κλάση μπορεί να την κάνει extend και να υλοποιήσει όλες τις abstract μεθόδους της.
- Σε σύγκριση με τα interfaces, **οι abstract classes περιέχουν πεδία non static και final, και παρέχουν υλοποίηση για κάποιες από τις μεθόδους τους.**
- Ο λόγος που υπάρχουν είναι για να αφήνουν στον προγραμματιστή που τις κάνει extend την ελευθερία να ολοκληρώσει την υλοποίηση τους.
- **Μια abstract κλάση που περιέχει μόνο abstract μεθόδους είναι καλύτερο να δηλωθεί ως interface.**

-
- Τεχνολογία Java
 - Αντικειμεστραφής προγραμματισμός και συντακτικό της γλώσσας
 - Χρήσιμες κλάσεις
 - Κληρονομικότητα
 - Διεπαφές, Αφηρημένες Κλάσεις
 - **Πακέτα και εμβέλεια κλάσεων**
 - Εργαλεία υλοποίησης

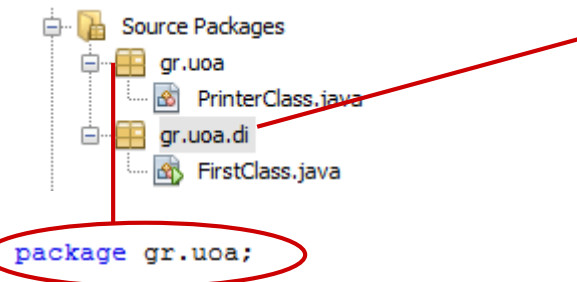
Πακέτα (Packages)

- ❑ Με τον όρο **package** ορίζουμε την ομαδοποίηση συναφών κλάσεων και interfaces σε ένα κατάλογο.
- ❑ Πλεονεκτήματα από τον ορισμό πακέτων
 - Ορισμός συνάφειας μεταξύ τύπων
 - Modularity..
 - Δεν υπάρχουν «συγκρούσεις» στην ονοματολογία των κλάσεων, καθώς κάθε package ορίζει δικό του namespace.
 - Επιτρέπουμε δικαίωμα πρόσβασης μεταξύ κλάσεων στο ίδιο package και περιορίζουμε την πρόσβαση σε κλάσεις έξω από αυτό.
- ❑ Το Java API παρέχει κώδικα ομαδοποιημένο σε πακέτα (java.lang, java.util...)
- ❑ Λέξη κλειδί: **package**

Πακέτα (Packages)

- Σύμβαση ονοματολογίας:
 - το όνομα ενός package είναι με μικρά
 - Πολλές εταιρείες επιλέγουν σαν όνομα package το αντίστροφο του ονόματος του Internet domain τους π.χ package gr.uoa.di
- Αν θέλουμε να αναφερθούμε στον κώδικα μας στις κλάσεις κάποιου πακέτου γράφουμε `import gr.uoa.di.*`

Παράδειγμα



```
package gr.uoa;

public class PrinterClass {
    public PrinterClass() {
    }

    public void printData(String x) {
        System.out.println(x);
    }
}

package gr.uoa.di;

import gr.uoa.PrinterClass;

public class FirstClass {

    public static void main(String[] args){
        if(args.length!=1){
            System.out.println("Program requires exactly one parameter");
            System.exit(1);
        }
        PrinterClass obj = new PrinterClass();
        obj.printData(args[0]);
    }
}
```

Εμβέλεια (Scope)

- ❑ Η Java μας παρέχει ένα σύνολο από access modifiers για να ορίζουμε τα όρια – την εμβέλεια - που είναι ορατή μια κλάση ή ένα πεδίο/μεταβλητή μιας κλάσης.
- ❑ Η εισαγωγή πακέτων ορίζει πεδία εμβέλειας
- ❑ Εμβέλεια κλάσης
 - Μία κλάση μπορεί να δηλωθεί ως public και να υπάρχει πρόσβαση από άλλες κλάσεις, ανεξάρτητα από το πακέτο που βρίσκονται.
 - Αν δεν υπάρχει συγκεκριμένος modifier, θεωρούμε τον default (package-private) με τον οποίο μια κλάση είναι ορατή **MONO** στο πακέτο της.
- ❑ Εμβέλεια πεδίων (Φθίνουσα αυστηρότητα)
 - **private**: μόνο οι μέθοδοι της κλάσης έχουν πρόσβαση στο πεδίο αυτό.
 - **package-private**: μόνο η ίδια η κλάση και κλάσεις που ανήκουν στο ίδιο πακέτο έχουν πρόσβαση στο πεδίο αυτό.
 - **protected**: πρόσβαση σε αυτό έχουν η ίδια η κλάση, κλάσεις που ανήκουν στο ίδιο πακέτο και κλάσεις που την κληρονομούν (ανεξαρτήτως πακέτου)
 - **public**: όλες οι κλάσεις, ανεξαρτήτως πακέτου, έχουν πρόσβαση στο πεδίο αυτό.

-
- Τεχνολογία Java
 - Αντικειμεστραφής προγραμματισμός και συντακτικό της γλώσσας
 - Χρήσιμες κλάσεις
 - Κληρονομικότητα
 - Διεπαφές, Αφηρημένες Κλάσεις
 - Πακέτα και εμβέλεια κλάσεων
 - **Εργαλεία υλοποίησης**

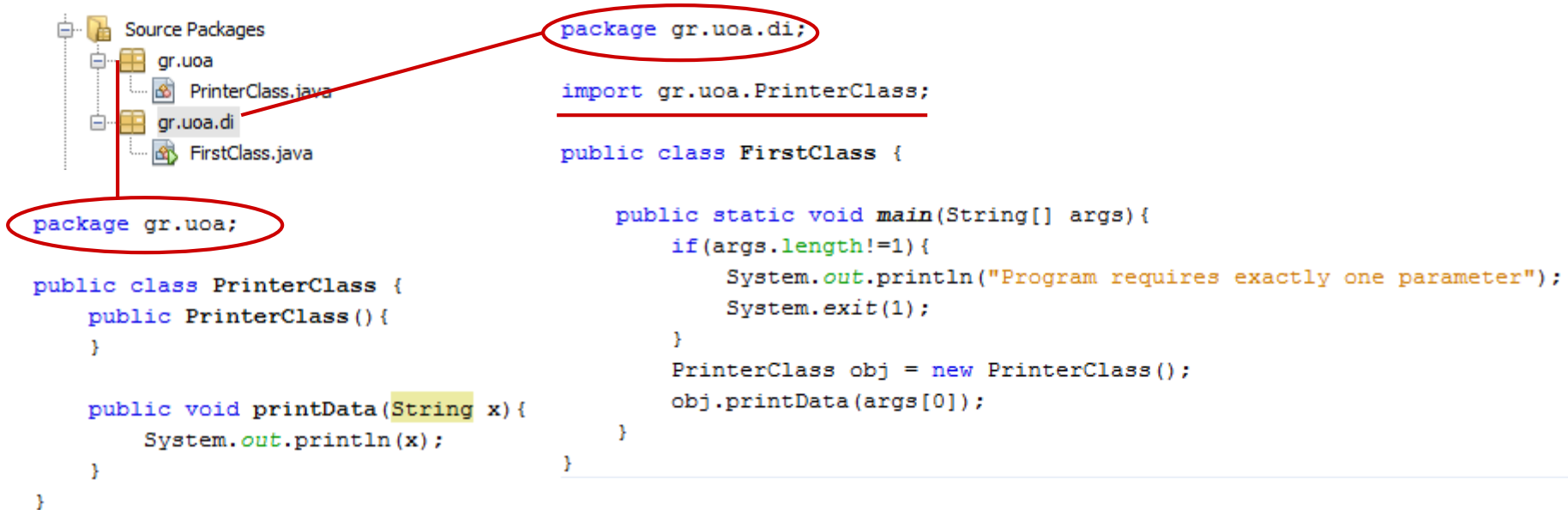
Συγγραφή, Μεταγλώττιση και Εκτέλεση προγράμματος

- Απαραίτητα:
 - Java Development Kit (JDK):
<http://www.oracle.com/technetwork/java/javase/downloads>
 - Συγγραφή κώδικα
 - ένας κειμενογράφος: Notepad, Textpad, WordPad
 - ένα IDE: Eclipse, Netbeans, **IntelliJ IDEA**

- Μεταγλώττιση και εκτέλεση από την γραμμή εντολών
 - `javac -cp path_to_libraries;path_to_classes -sourcepath path_to_sources`
 - `java -cp path_to_libraries;path_to_classes MainClass arg1 arg2 ... argn`
 - Σε περιβάλλον Linux το διαχωριστικό ανάμεσα στις βιβλιοθήκες είναι : αντί για ; των Windows.

Παράδειγμα

```
C:\Users\Panagis\Desktop\JavaApplication4>"C:\Program Files\Java\jdk1.6.0_10\bin\njavac.exe" src\gr\uoa\PrinterClass.java -d classes  
C:\Users\Panagis\Desktop\JavaApplication4>"C:\Program Files\Java\jdk1.6.0_10\bin\njavac.exe" -cp classes src\gr\uoa\di\FirstClass.java -d classes  
C:\Users\Panagis\Desktop\JavaApplication4>java -cp classes/ gr.uoa.di.FirstClass "Hello World"  
Hello World
```



- Μεταγλώττιση
 - javac.exe src\gr\uoa\PrinterClass.java -d classes
 - javac.exe -cp classes src\gr\uoa\di\FirstClass.java -d classes
- Εκτέλεση
 - java -cp classes/ gr.uoa.di.FirstClass "Hello World"
- Αποτέλεσμα
 - Hello World

Αναφορές

- Javadoc
 - <http://download.oracle.com/javase/8/docs/api/>
- Oracle tutorial
 - <http://docs.oracle.com/javase/tutorial/index.html>