Διάλεξη 02: Αντικειμενοστρεφής Προγραμματισμός με την JAVA

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:
Εισαγωγή στις έννοιες:

- Επισκόπηση της JAVA
- Περισσότερες Πληροφορίες στο μάθημα ΕΠΛ233
  http://www.cs.ucy.ac.cy/courses/EPL233/

Διδάσκων: Παναγιώτης Ανδρέου

Προγράμματα

Τα προγράμματα (*software)*, είναι βασικά ένα σύνολο από οδηγίες στον υπολογιστή. Χωρίς τα προγράμματα ο υπολογιστής είναι μία άδεια μηχανή.

Δεδομένου ότι ο υπολογιστής δεν καταλαβαίνει την φυσική γλώσσα αλλά μόνο την γλώσσα μηχανής (machine language), χρειαζόμαστε κάποιες ειδικές γλώσσες (γλώσσες προγραμματισμού) για να επικοινωνήσουμε μαζί του.

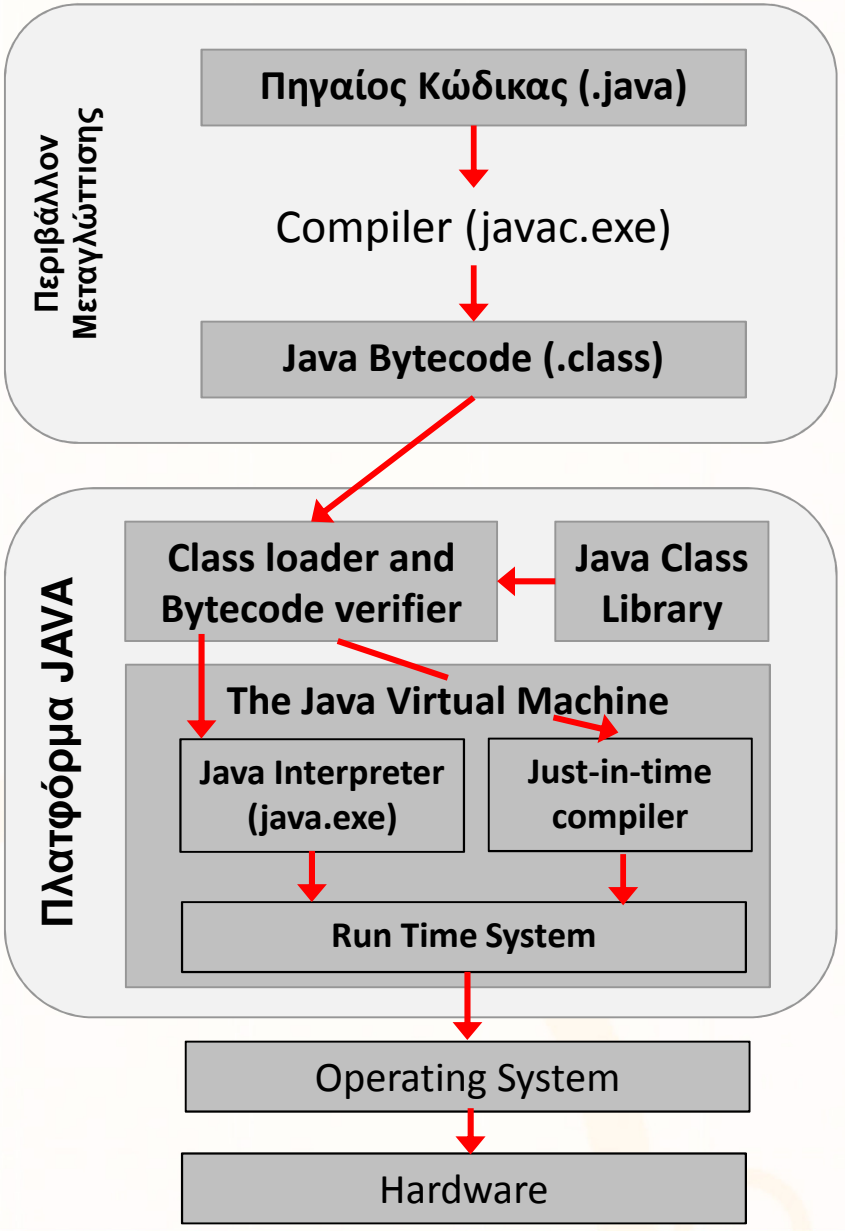
Παραδείγματα γλωσσών προγραμματισμού: C, C++, JAVA, C#, Visual Basic, Fortran, κ.τ.λ.

ΠΑΡΑΔΕΙΓΜΑ ΠΡΟΓΡΑΜΜΑΤΟΣ σε JAVA

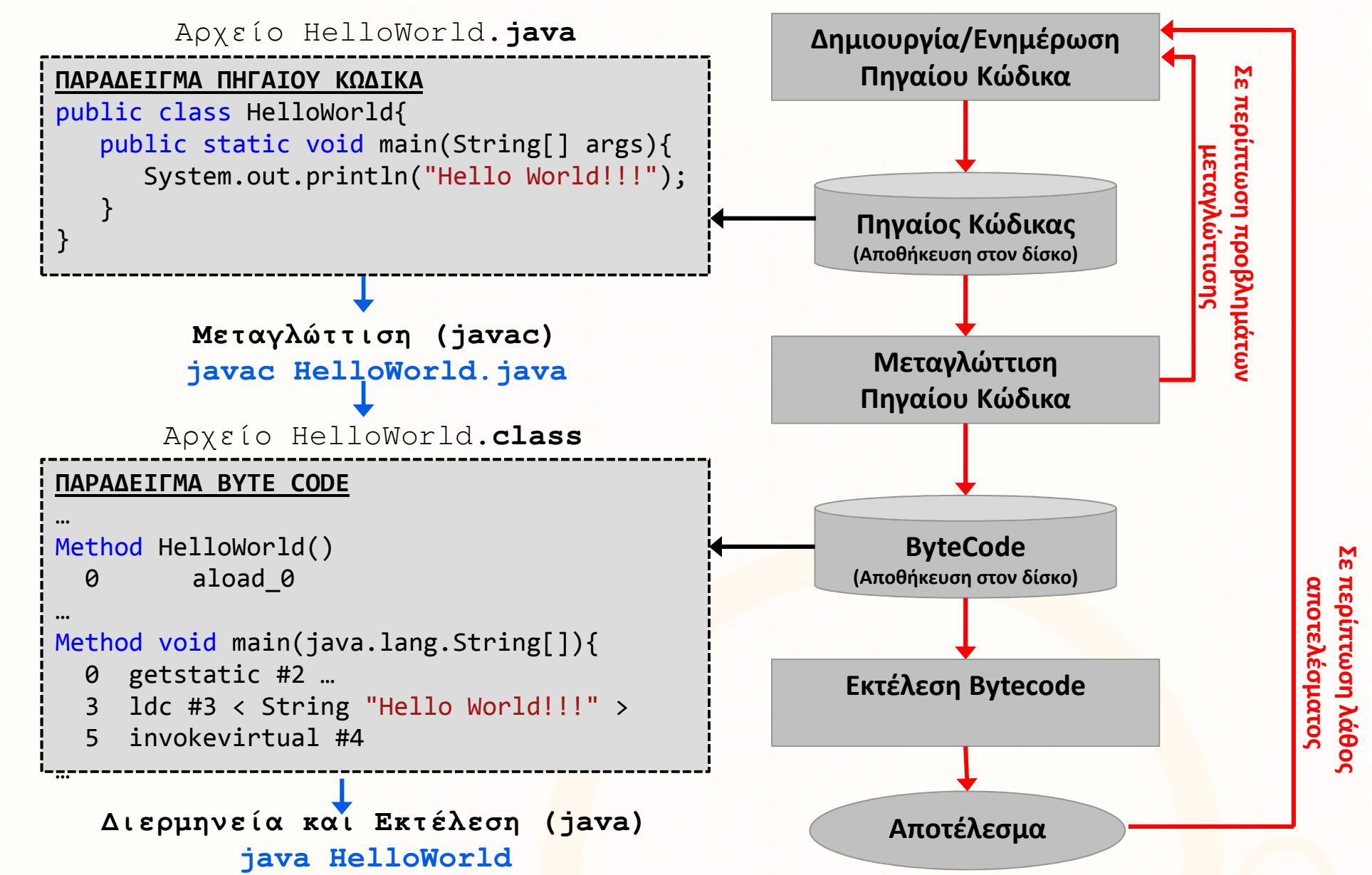
```
public class HelloWorld{
   public static void main(String[] args){
      System.out.println("Hello World!!!");
   }
}
```

Η πλατφόρμα της JAVA

- Πλατφόρμα:
  - Περιβάλλον λογισμικού και υλικού στο οποίο εκτελείται ένα πρόγραμμα.
  - Συνήθως είναι συνδυασμός του Λειτουργικού Συστήματος και του Υλικού Υποστρώματος του ΛΣ.
  - Δημοφιλείς πλατφόρμες: Microsoft Windows, Linux, Solaris OS, Mac OS.
- Πλατφόρμα Java: Σύστημα λογισμικού που τρέχει πάνω σε διάφορες πλατφόρμες υλικού. Αποτελείται από:
  - Την Εικονική Μηχανή JAVA: Java Virtual Machine
  - Την Προγραμματιστική Διαπροσωπεία Εφαρμογών της JAVA (Java Application Programming Interface - API)



Τυπικός Κύκλος Ζωής ενός Προγράμματος JAVA

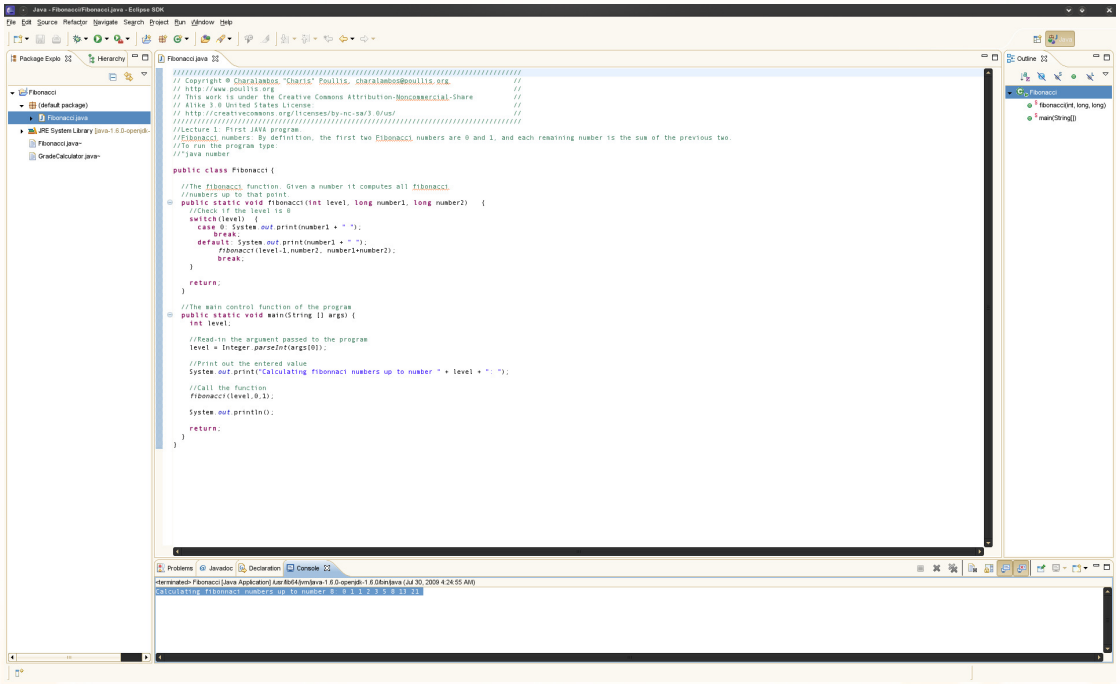


Ενσωματωμένο Περιβάλλον Ανάπτυξης (IDE)

Ένα Ενσωματωμένο Περιβάλλον Ανάπτυξης (Integrated Development Environment (IDE)) συνήθως περιλαμβάνει:

- Ένα συντάκτη πηγαίου κώδικα (π.χ., Notepad)
- Εργαλεία αυτοματοποίησης (π.χ., αυτόματη μεταγλώττιση κώδικά, παρουσίαση συντακτικών λαθών)
- Ένα ελεγκτή/παρατηρητή (debugger)

Παράδειγμα – Eclipse IDE



Παραδείγματα: Κλάση

```
class Circle {
  //Η ακτίνα αυτού του κύκλου
  double radius = 1.0;

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  Circle () {
  };

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  //με συγκεκριμένη ακτίνα
  Circle (double newRadius) {
    radius = newRadius;
  };

  //Επέστρεψε το εμβαδό αυτού του κύκλου
  double getArea () {
    return radius * radius * π;
  }

}
```

Δεδομένα/ Μεταβλητές

Κατασκευαστές

Μέθοδοι

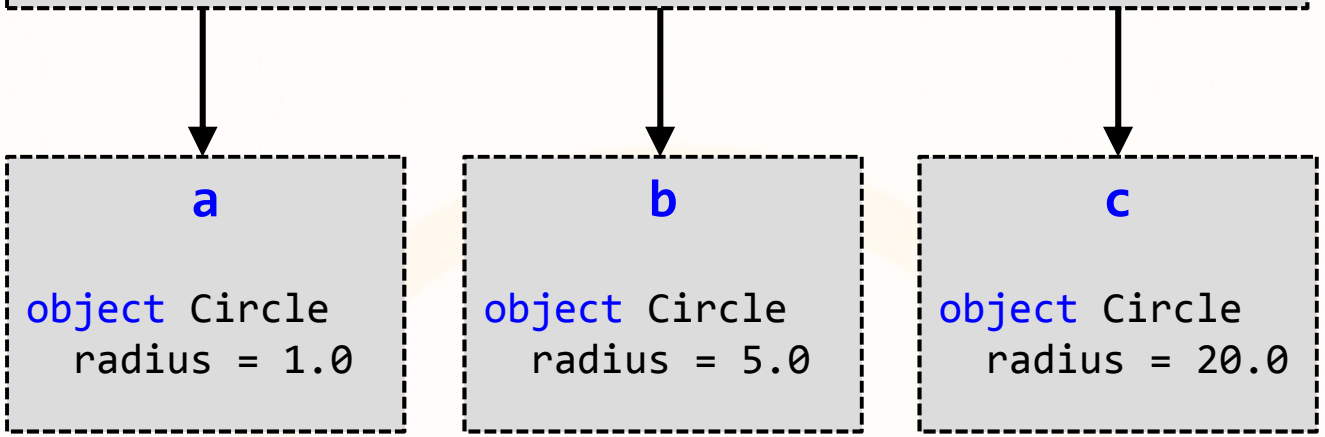
Παραδείγματα: Αντικείμενα

Κλάση Circle

```
class Circle {
  //Η ακτίνα αυτού του κύκλου
  double radius = 1.0;

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  Circle () {
  };

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  //με συγκεκριμένη ακτίνα
  Circle (double newRadius) {
    radius = newRadius;
  };

  //Επέστρεψε το εμβαδό αυτού του κύκλου
  double getArea () {
    return radius * radius * π;
  }

}
```

Κλάση Test: περιλαμβάνει την μέθοδο main

```
public class Test{

  public static void main(String[] args){
    Circle a = new Circle();
    Circle b = new Circle(5);
    Circle c = new Circle(20);

…
```



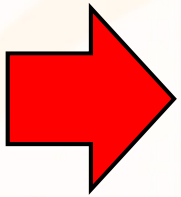
3 αντικείμενα της κλάσης Circle

Απλό if … else

- Θυμηθείτε το πρόγραμμα για τη δημιουργία κύκλων με ακτίνα
- **Πρόβλημα:** Τι γίνεται αν δώσουμε αρνητική τιμή για την ακτίνα του κύκλου; ➔ λάθος αποτέλεσμα
- **Λύση:** Μόνο αν ο χρήστης δώσει θετική τιμή τότε αλλάζουμε την τιμή της radius
- Σύνταξη απλού if: **if (** <boolean expression> **) { … } else { … }**

```
class Circle {
  //Η ακτίνα αυτού του κύκλου
  double radius = 1.0;

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  Circle () {
  };

  //Δημιούργησε ένα αντικείμενο τύπου κύκλος
  //με συγκεκριμένη ακτίνα
  Circle (double newRadius) {
    radius = newRadius;
  };

  //Επέστρεψε το εμβαδό αυτού του κύκλου
  double getArea () {
    return radius * radius * π;
  }

}
```

```
class Circle {
  double radius = 1.0;
  …
  Circle (double newRadius) {
    radius = newRadius;
  };
  …
}
```



```
class Circle {
  double radius = 1.0;
  …
  Circle (double newRadius) {
    if ( newRadius >= 0 ){
      radius = newRadius;
    }
    else {
      radius = 0;
    }
…
```

if … else if … else

- Τι συμβαίνει αν υπάρχουν πολλές συνθήκες για την ίδια μεταβλητή;
- **Παράδειγμα:** Υπάρχουν οι ακόλουθοι φόροι σύμφωνα με τον μισθό κάποιου ατόμου:

  μισθός <=20,000 → 0% φόρος
  μισθός >20,000 και <=30,000 → 20% φόρος
  μισθός >30,000 και <=40,000 → 30% φόρος
  μισθός >40,000 → 40% φόρος

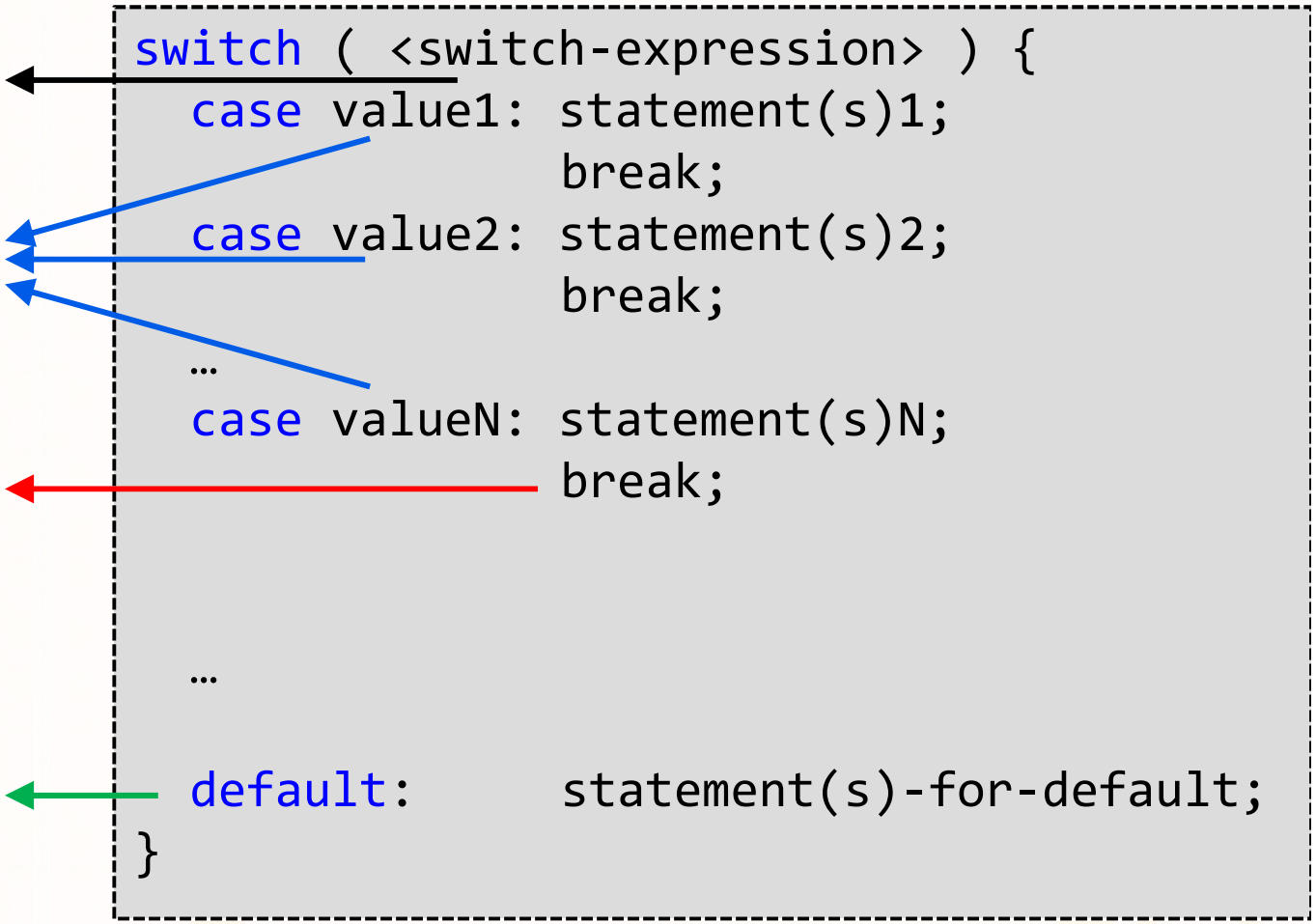
- **Πρόβλημα:** Χρειάζεται να γράψουμε 3x if statements; → **ΟΧΙ**
- **Λύση:** Χρήση του if … else if … else

```
computeTax (double salary) {
  double tax = 0.0;
  if ( salary <= 20000 ) { tax = 0.0; }
  else if ( salary > 20000 && salary <= 30000) { tax = 0.2; }
  else if ( salary > 30000 && salary <= 40000) { tax = 0.3; }
  else { tax = 0.4; }
  return tax * salary;
}
```

Λογικός Τελεστής ΚΑΙ (&&)
(στις επόμενες διαφάνειες)

switch … case

- Το switch ομαδοποιεί πολλά if
- Σύνταξη:



Ο τριαδικός τελεστής συνθήκης (ternary operator)

- Η ακόλουθη δήλωση

```
if (x > 0)
  y = 1;
else
  y = -1;
```

είναι ισοδύναμη με

```
y = (x > 0) ? 1 : -1;
```

- **Σύνταξη Τριαδικού Τελεστή: (<boolean-exp>) ? exp1 : exp2;**

- Παραδείγματα

```
System.out.println(     ( num % 2 == 0 ) ?  // (boolean-exp)
                        num + "is even"  :  // exp1
                        num + "is odd");    // exp2
```

Προγραμματισμός με βρόγχους (loops)

3 είδη δηλώσεων προγραμματισμού με βρόγχους

- **while**

```
while ( <boolean expression> ) {
  //δηλώσεις
}
```

- **do … while**

```
do {
  //δηλώσεις
} while ( <boolean expression> );
```

- **for**

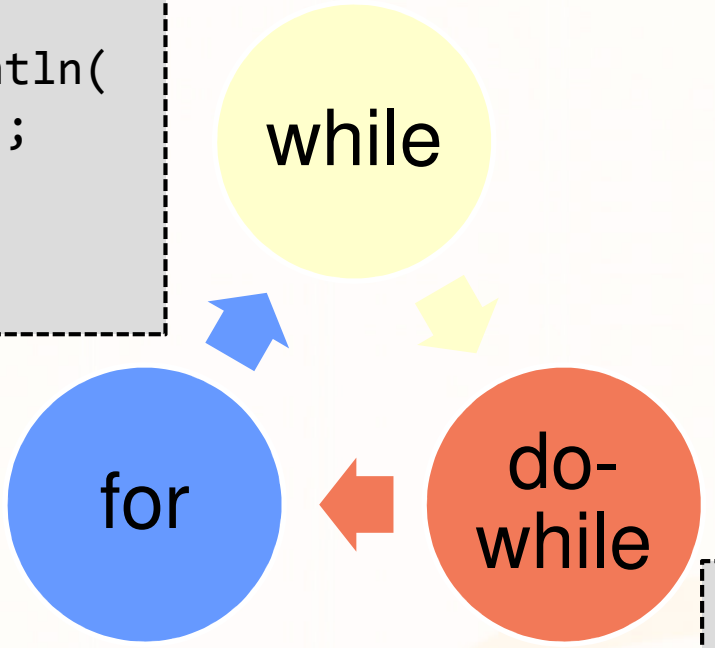
```
for( <initial actions>; <boolean expr.>; actions after step )
  //δηλώσεις
}
```

- **Όλα τα είδη βρόγχων είναι ισοδύναμα**.

Παραδείγματα προγραμματισμού με βρόγχους

Παράδειγμα <while>

```
int i = 0;
while (i < 10) {
  System.out.println(
          "while");
  i ++;
}
```



Παράδειγμα <for>

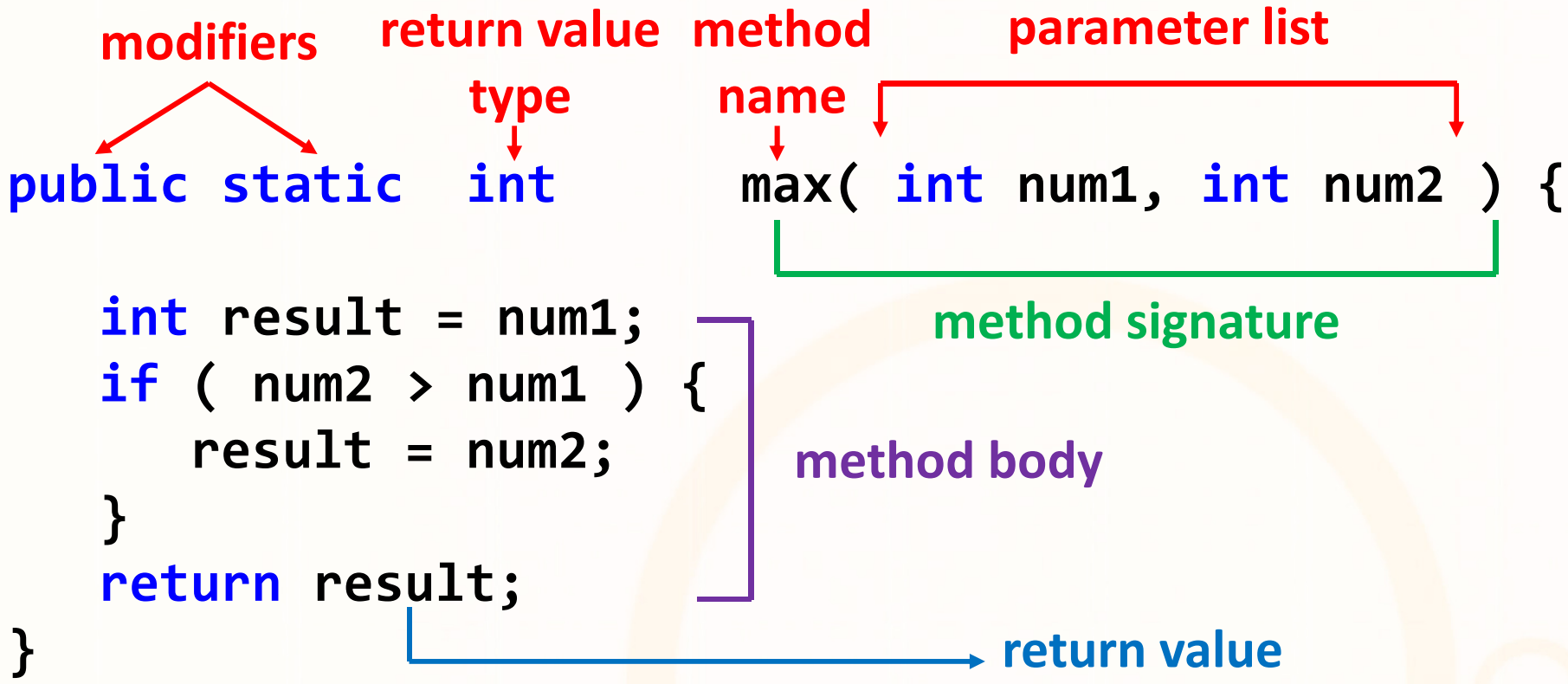
```
for ( int i=0; i<10; i++) {
  System.out.println(
          "for");
}
```

Παράδειγμα <do-while>

```
int i = 0;
do {
  System.out.println(
          "do-while");
  i++;
} while (i < 10);
```

Μέθοδοι (methods)

- **Μέθοδος:** μία συλλογή από ομαδοποιημένες δηλώσεις οι οποίες εκτελούν κάποια (ες) λειτουργία (ες).
- Η **υπογραφή μίας μεθόδου** αποτελείται από το **όνομά της** και τη **λίστα με της παραμέτρους** που δέχεται.
- Σύνταξη:



Πέρασμα Παραμέτρων

- **Πέρασμα διά τιμής:** οι τιμές των μεταβλητών αντιγράφονται στις παραμέτρους της μεθόδου ==> οι αρχικές μεταβλητές δεν αλλάζουν
- **Πέρασμα διά αναφοράς:** **δεν υπάρχει!**
- Παράδειγμα

```
public class Increment {
   public static void main( String[] args ) {
      int x = 1;
      System.out.println(
               "before the call, x is " + x);
      increment(x);
      System.out.println(
               "after the call, x is " + x);
   }
   public static void increment( int n ) {
      n++;
      System.out.println(
               "n inside the method is " + n);
   }
}
```

1. x=1

2. x=1

3. x=1 copy to n

7. x=1

4. n=1

5. n=2

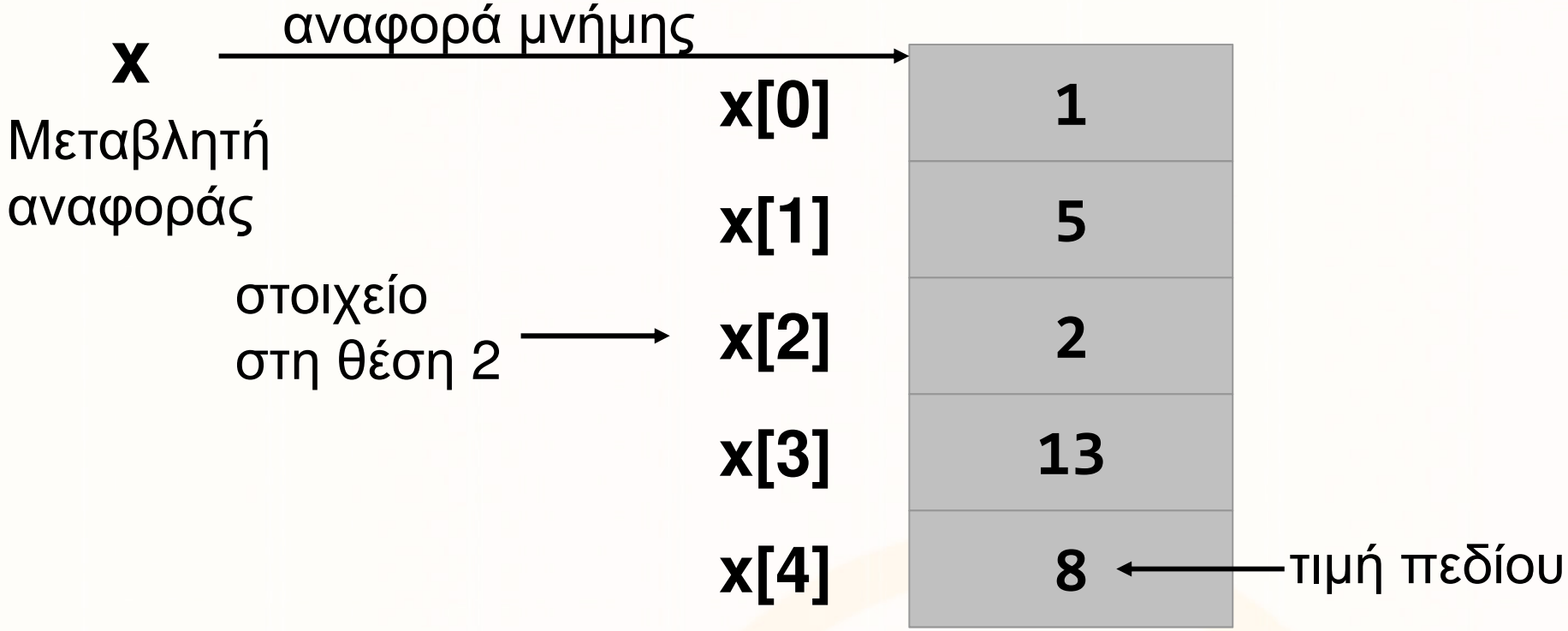
6. n=2

Πίνακες

- **Πίνακας:** μία δομή δεδομένων που αντιπροσωπεύει μία συλλογή με στοιχεία του ίδιου τύπου.
- Οι πίνακες στην JAVA έχουν σχεδιαστεί κατά τρόπον ώστε να ξεπερνιούνται οι δυσκολίες του προγραμματισμού πινάκων της C/C++.
- Στη JAVA είναι εξασφαλισμένο ότι ένας πίνακας θα αρχικοποιηθεί και ότι δεν θα επισυμβεί πρόσβαση εκτός των ορίων του.
- Τα χαρακτηριστικά αυτά υλοποιούνται με κάποιο σχετικό κόστος μνήμης και χρόνου εκτέλεσης (κατά την εκτέλεση γίνεται έλεγχος κατά πόσο δεν γίνεται υπέρβαση των ορίων του πίνακα).
- Κατά τη δημιουργία ενός πίνακα, κατ' ουσίαν δημιουργείται ένας πίνακας χειριστηρίων (Handles), τα οποία αρχικοποιούνται σε null.
- Είναι ευθύνη του προγραμματιστή να αρχικοποιήσει σωστά τα χειριστήρια, ώστε να παραπέμπουν σε αντικείμενα.

Πίνακες (συν.)

- Παράδειγμα αναπαράστασης πίνακα: `int[] x = new int[5];`



- Πρόσβαση σε κάθε στοιχείο του πίνακα με το ευρετήριο του. π.χ., το στοιχείο στη θέση 2 = x[2].
- Ιδιότητα (property) <**length**>: Επιστρέφει το μέγεθος του πίνακα. π.χ., x.length = 5

Παραδείγματα χρήσης πίνακα

- Τύπωμα των στοιχείων ενός πίνακα

```
int[] x = new int[5];
...
for(int i=0; i<x.length; i++) {
   System.out.println( x[i] );
}
```

- Πρόσθεση των στοιχείων ενός πίνακα

```
int[] x = new int[5];
int sum=0;
...
for(int i=0; i<x.length; i++) {
   sum = sum + x[i];
}
```

Πολυδιάστατοι πίνακες

Σύνταξη: **<τύπος δεδομένων>[ ][ ]... <όνομα πίνακα>**, π.χ., int[][] x

Δήλωση Πινάκων

- `int[][]    x;    //Δήλωση πίνακα 2D με ακέραιους`
- `char[][][] a;    //Δήλωση πίνακα 3D με χαρακτήρες`

Αρχικοποίηση Πινάκων

- Η αρχικοποίηση μπορεί να γίνει με τη δήλωση `new`
  `x = new int[2][3]; //Δημιουργία πίνακα με 2 γραμμές και 3 στήλες`
- ή μπορεί να γίνει με την αυτόματη ανάθεση στοιχείων
  `x = { {1, 2, 3}, {4, 5, 6} };` **μόνο κατά την δήλωση!**

Δήλωση και Αρχικοποίηση σε ένα Βήμα

- `int[][]  x= new int[2][3];`

Συλλογές (collections)

- Οι συλλογές (collections) είναι δομές παρόμοιες με πίνακες (arrays) τις εξής διαφορές:
  - Το μέγεθος του μεταβάλλεται δυναμικά
  - Οποιοδήποτε είδος αντικειμένου μπορεί να εισαχθεί σε μία συλλογή
  - ΔΕΝ υποστηρίζουν εισαγωγή αρχέγονων τύπων (π.χ., int).
- Παραδείγματα συλλογών: Vector, BitSet, HashTable, Stack
- **ArrayList**: πολυχρησιμοποιημένη δομή (παρόμοια με Vector)
  - Δεν είναι συγχρονισμένη, αντίθετα με vector
  - Όταν αυξάνει δυναμικά το μέγεθος κατά το μισό του υπάρχον μέγεθος (ο vector αυξάνει κατά το διπλάσιο)
  - Είναι πιο γρήγορη λόγω του ότι δεν είναι συγχρονισμένη

Boxing

- **Οι συλλογές ΔΕΝ δέχονται πρωτόγονους τύπους αλλά μόνο δείκτες σε αντικείμενα (reference types).**
- Ερώτηση: Τι γίνεται όταν χρειαζόμαστε πρωτόγονους τύπους;
- Απάντηση: Boxing

Ορισμοί:

- **Box**
  Ένα στιγμιότυπο μίας κλάσης περιτυλίγματος (wrapper) η οποία αποθηκεύει την τιμή ενός πρωτόγονου τύπου.
- **Boxing**
  Δημιουργία ενός box για μία τιμή πρωτόγονος τύπου
- **Unboxing**
  Επιστροφή της τιμής πρωτόγονου τύπου από το box

Manual/Auto boxing and unboxing

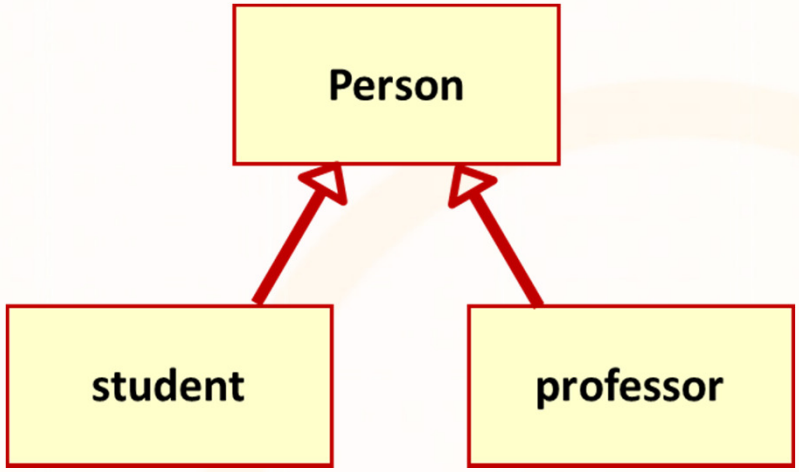
- Οι αρχέγονοι τύποι δεν μπορούν να χρησιμοποιηθούν στις περισσότερες περιπτώσεις -**you need a "wrapper"**
  - myVector.add(new Integer(5));
- Αντίστοιχα δεν επιτρέπεται να χρησιμοποιηθεί ένα αντικείμενο εκεί που χρειάζεται αρχέγονος τύπος. --**you need to "unwrap" it**
  - int n = ((Integer)myVector.lastElement()).intValue();

- Integer iNumber = new Integer(10); → Manual boxing
- Integer iNumber = 10; → Auto-boxing
- iNumber = new Integer(iNumber.intValue()++); → Manual unboxing
- iNumber++; → Auto-unboxing

Κληρονομικότητα

Δύο βασικές έννοιες:

- Ένα είδος σχέσης
- Ένας προγραμματιστικός μηχανισμός για επαναχρησιμοποίηση

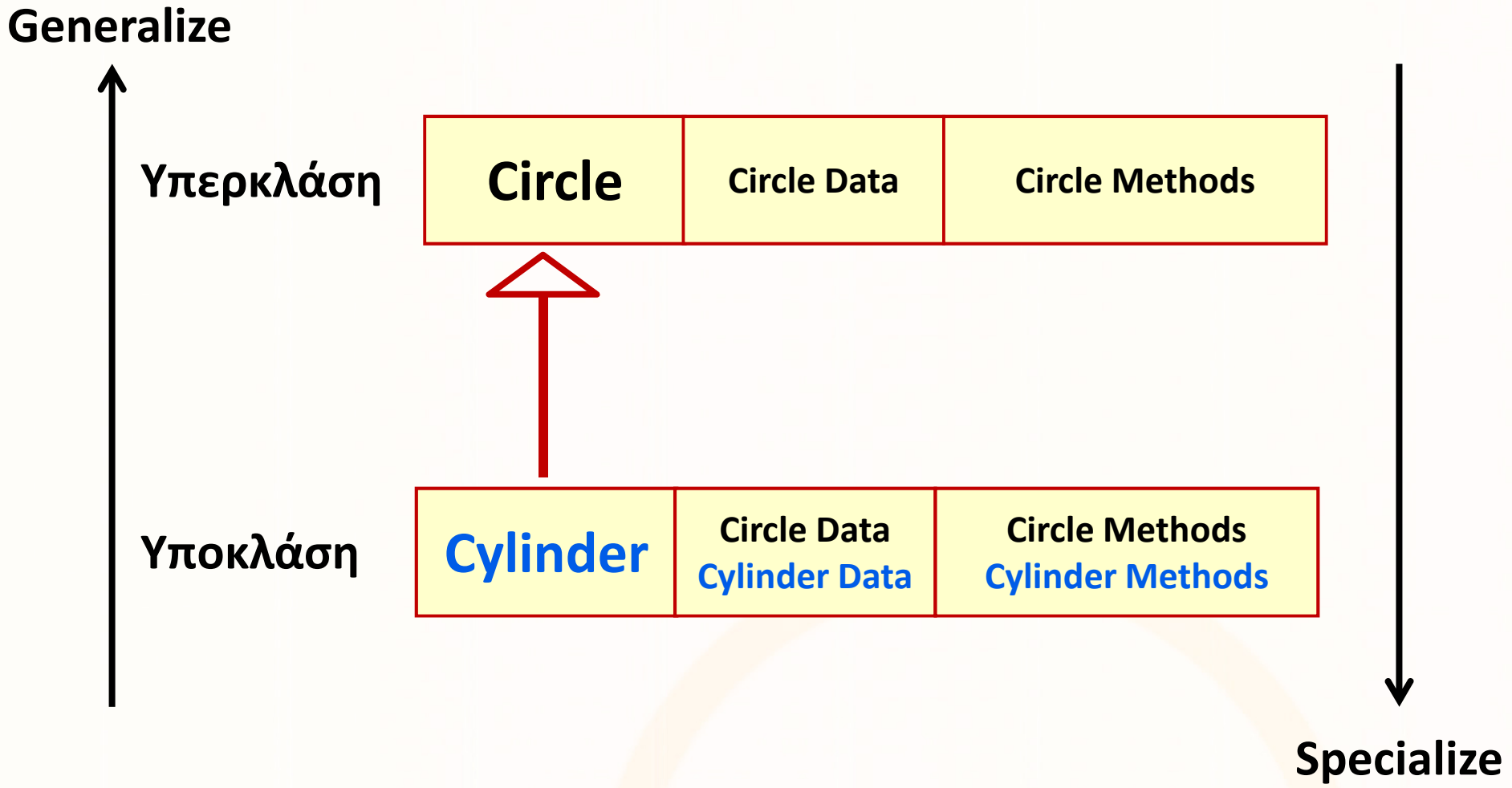


Προγραμματιστικός Μηχανισμός Κληρονομικότητας

- **Μία κλάση μπορεί να κληρονομήσει όλα τα "επιτρεπτά" στοιχεία (πεδία και μεθόδους) από τον "πατέρα"** της από τον πατέρα του πατέρα της, κτλ.

- Μία υποκλάση μπορεί να **προσθέσει καινούρια πεδία**
- Μία υποκλάση μπορεί να **προσθέσει καινούριες μεθόδους**
- Μία υποκλάση μπορεί να **επεκτείνει υφιστάμενες μεθόδους (overloading)**
- Μία υποκλάση μπορεί να **επανακαθορίσει υφιστάμενες μεθόδους (overriding)**

Υποκλάσεις και Υπερκλάσεις



Υποκλάσεις και Υπερκλάσεις (συν.)

```
class Circle {
  // Η ακτίνα αυτού του κύκλου
  private double radius;

  // Δημιούργησε ένα κύκλο
  public Circle () {radius = 1.0;}

  // Δημιούργησε ένα κύκλο
  // συγκεκριμένη ακτίνα
  public Circle(double newRadius)
  {
    radius = newRadius;
  }

  // Επέστρεψε την ακτίνα
  public double getRadius() {
    return radius;
  }

  // Επέστρεψε το εμβαδό
  public double getArea() {
    return radius * radius * π;
  }
}
```

```
class Cylinder extends Circle {
  // Το μήκος αυτού του κύλινδρου
  private double length;

  // Δημιούργησε ένα κύλινδρο
  public Cylinder() {
    super();
    this.length= 1.0;}

  // Δημιούργησε ένα κύλινδρο με
  // συγκεκριμένο μήκος
  public Cylinder(int length) {
    this.length= length; }

  // Επέστρεψε το μήκος
  public double getLength() {
    return length; }

  // Επέστρεψε τον όγκο
  // αυτού του κύλινδρου
  public double getVolume () {
    return getArea() * length;}
}
```

Χειριστήριο Υπερκλάσης (super)

- Η λέξη κλειδί **super** αναφέρεται στην υπερκλάση
- Μπορεί να χρησιμοποιηθεί με δύο τρόπους:
  - Για το **κάλεσμα του constructor** της υπερκλάσης
  - Για το **κάλεσμα μία μεθόδου** της υπερκλάσης
- Στην περίπτωση που καλείται ο constructor της κλάσης τότε πρέπει να είναι η πρώτη δήλωση στον κώδικα.

- Σημείωση: Η λέξη κλειδί **this** αναφέρεται στην υποκλάση και είναι ένα χειριστήριο προς το ίδιο το αντικείμενο.

Απόκρυψη Ονομάτων (Name Hiding)

Πεδία

- Μέσα σε μία κλάση, κάθε πεδίο πρέπει να έχει μοναδικό όνομα
- Μέσα από την κληρονομικότητα προκύπτουν περιπτώσεις που ένα πεδίο στην υποκλάση μπορεί να έχει το ίδιο όνομα με ένα πεδίο στην υπερκλάση
- Σε αυτή την περίπτωση, το πεδίο στην υποκλάση κρύβει/επισκιάζει (hides) το πεδίο στην υπερκλάση
- Για να έχουμε πρόσβαση στο πεδίο της υπερκλάσης πρέπει να χρησιμοποιήσουμε την λέξι κλειδί super

Μεθόδοι

- Παρόμοια υπάρχει περίπτωση μία μέθοδος στην υποκλάση να έχει την ίδια υπογραφή με μία μέθοδο της υπερκλάσης
- Σε αυτή την περίπτωση, η μέθοδος της υποκλάσης υπερσκελίζει (overrides) την μέθοδο της υπερκλάσης

Παράδειγμα Υπερσκέλισης (Overriding)

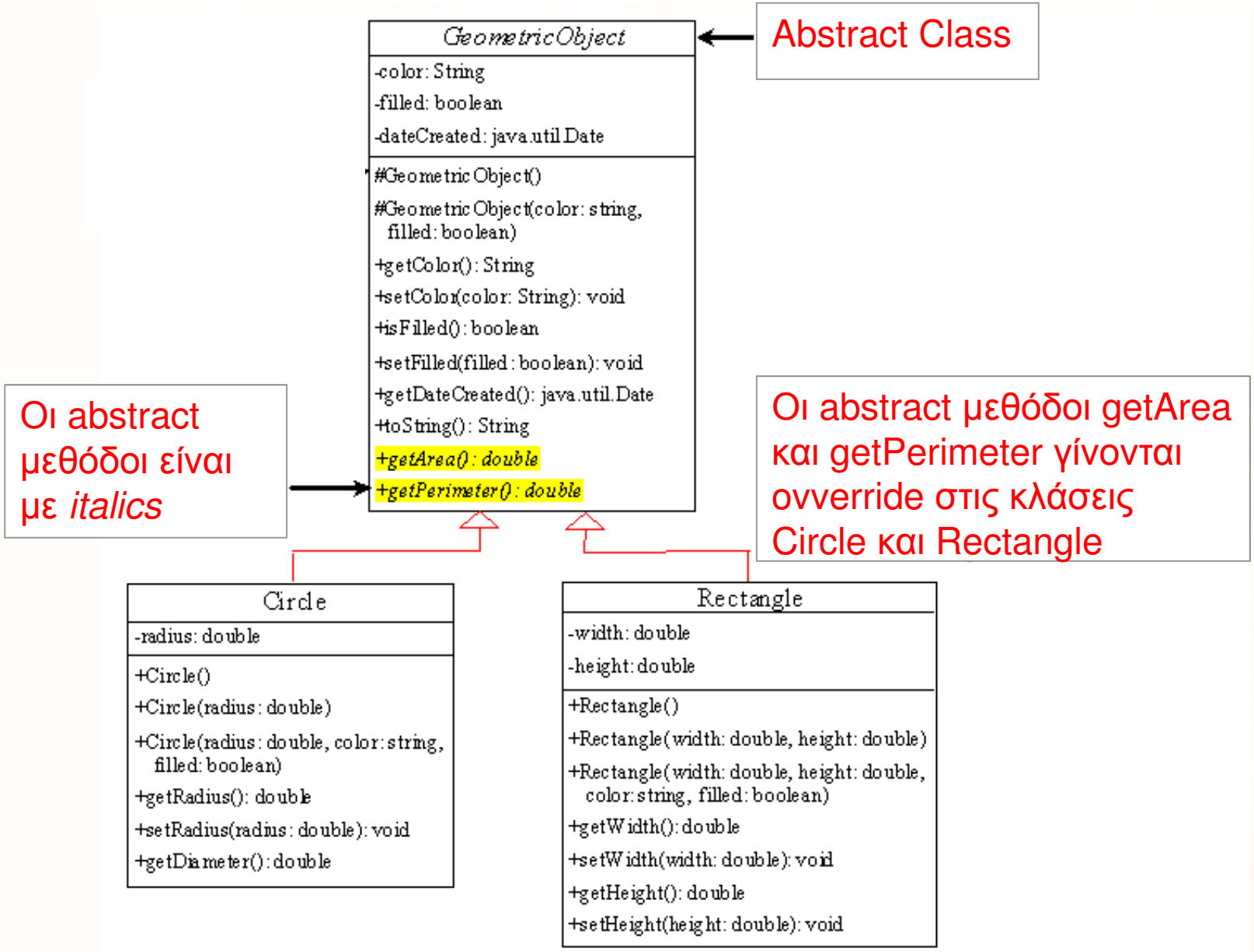
```
public class Circle extends GeometricObject {
  // Other methods are omitted

  /** Override the toString method
      defined in GeometricObject */
  public String toString() {
    return super.toString() +
           "\nradius is " + radius;
  }
}
```

Αφαιρετικές (abstract) κλάσεις και μεθόδοι

- **Μία αφαιρετική κλάση - abstract class**
  - Δεν μπορεί να αρχικοποιηθεί αντικείμενο της κλάσης
  - Πρέπει να επεκταθεί (κληρονομικότητα) και να υλοποιηθεί από τις υποκλάσεις

- **Μία αφαιρετική μέθοδος - abstract method**
  - Η υπογραφή μίας μεθόδου χωρίς υλοποίηση

Παράδειγμα: Αφαιρετικές Κλάσεις και Μέθοδοι



Αφαιρετικές Μεθόδοι

- Η Java μας επιτρέπει να δηλώνουμε **ρητά** (explicit) ορισμένες μεθόδους μιας κλάσης σαν αφαιρετικές ώστε:
  - Να έχουμε τη δυνατότητα να προσδιορίζουμε την διεπαφή της κλάσης που τις περιέχει.
  - Να αποτρέπεται η εκ παραδρομής κλήση τους από άλλες μεθόδους.

- Ο προσδιορισμός μιας μεθόδου ως αφαιρετικής γίνεται ως εξής:

abstract void X();

- **Τα abstract methods μπορούν να υπάρχουν μόνο σε abstract classes**

Αφαιρετικές κλάσεις

- Μια κλάση η οποία περιλαμβάνει έστω και μια «αφαιρετική» μέθοδο, καθίσταται επίσης αφαιρετική **και πρέπει να δηλωθεί ως αφαιρετική**.
- Όταν μία κλάση κληρονομεί από μια αφαιρετική κλάση θα πρέπει να υλοποιήσουμε **όλες τις αφαιρετικές μεθόδους της υπερκλάσης**. Διαφορετικά ο μεταφραστής επιβάλλει να προσδιορίσουμε την κλάση μας σαν αφαιρετική.
- Μπορούμε τέλος να δηλώσουμε μια κλάση σαν αφαιρετική, χωρίς ωστόσο η κλάση αυτή να περικλείει αφαιρετικές μεθόδους. Γιατί να θέλουμε κάτι τέτοιο; **Για να αποκλείσουμε τη δημιουργία αντικειμένων αυτής της κλάσης**.

Διαπροσωπείες (Interfaces)

- **Μία διαπροσωπεία (interface)** είναι μία δομή (παρόμοια με την κλάση) ή οποία **περιέχει:**
  - **Δηλώσεις μεθόδων** και **όχι καθορισμό τους (εμμέσως είναι abstract)**
  - **Σταθερές (constant) μεταβλητές**

- Καθορίζει την μορφή που πρέπει να υπάρχει σε μία κλάση, ένα συμβόλαιο το οποίο δηλώνει πως δουλεύει μία κλάση

- Δηλώνεται με την λέξη **`interface`**

- Οι Διαπροσωπείες είναι **παρόμοιες με τις Αφαιρετικές κλάσεις, αλλά όχι οι ίδιες**

Δήλωση Διαπροσωπείας (Interface Declaration)

- Σύνταξη δήλωσης διαπροσωπείας:

```
public interface InterfaceName {
  //constant declarations
  //method signatures
}
```

- Παράδειγμα

```
public interface Edible{
  // Describe how to eat
  public String howToEat();
}
```

Σταθερές Μεταβλητές (Constant Variables)

- Παράδειγμα δήλωσης Σταθερών Μεταβλητών

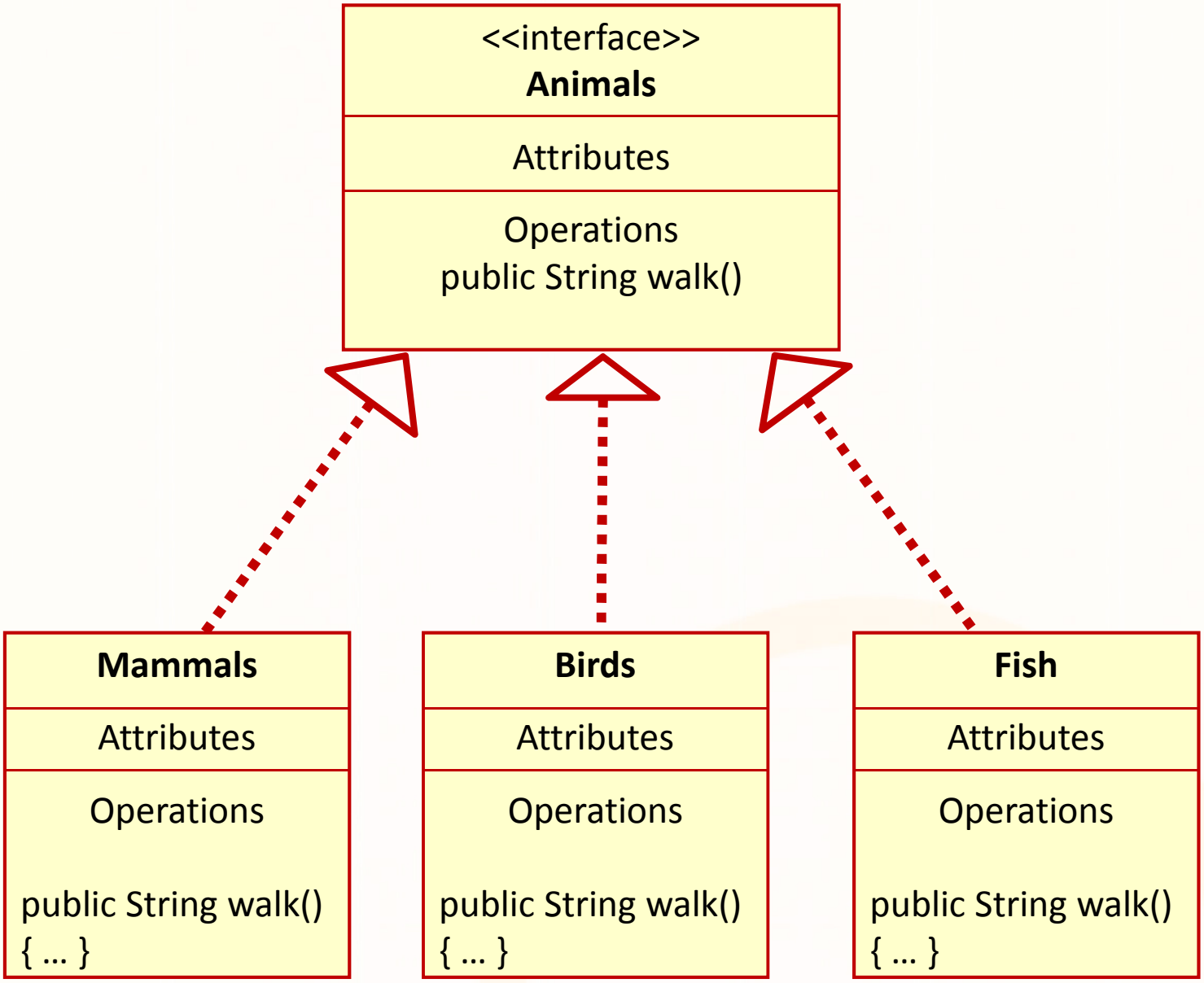
```
public interface InterfaceName {
  //constant declarations
  int x=1;
  double PI=3.1415;
}
```

- Το πιο πάνω είναι ισοδύναμο με το πιο κάτω

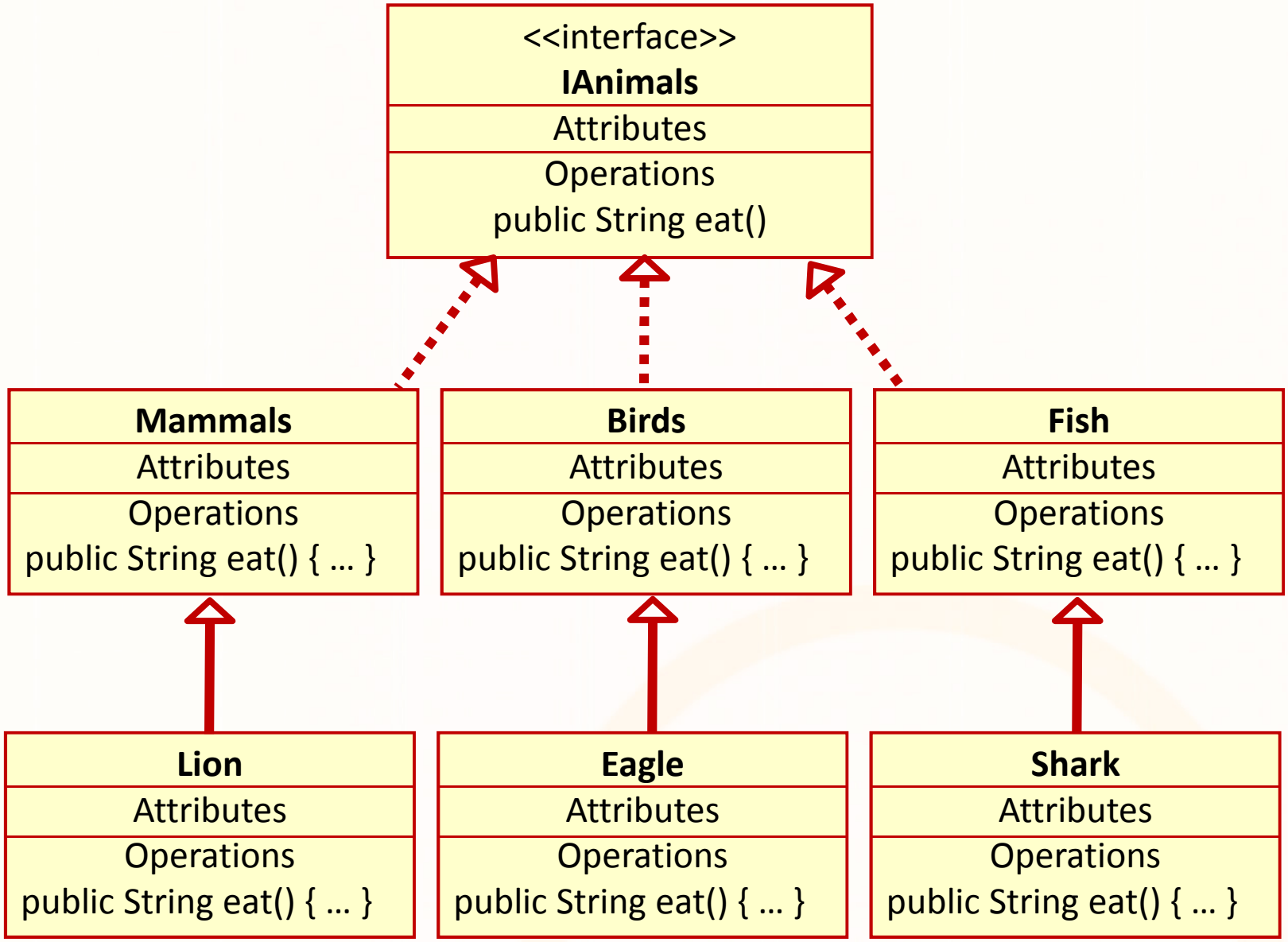
```
public interface InterfaceName {
  //constant declarations
  public static final int x=1;
  public static final double PI=3.1415;
}
```

- Πρόσβαση με <διαπροσωπεία>.<μεταβλητή>
  π.χ., **InterfaceName.x**

Παράδειγμα Διαπροσωπείας 1



Παράδειγμα Διαπροσωπείας και Κληρονομικότητας



Παράδειγμα Διαπροσωπείας και Κληρονομικότητας

```
public interface IAnimals {
abstract void eat();
String getName();
}

class Mammals implements IAnimals {
  public void eat() { }
  public String getName() {}
}

class Birds implements IAnimals {
  public void eat() { }
  public String getName() {}
}

class Fish implements IAnimals {
  public void eat() { }
  public String getName() {}
}
```

```
class Lion extends Mammals {
  public void eat() { }
  public String getName() {}
}

class Eagle extends Birds {
  public void eat() { }
  public String getName() {}
}

class Shark extends Fish {
  public void eat() { }
 public String getName() {}
}
```

Σημαντικές διαπροσωπείες: Comparable

- Η διαπροσωπεία Comparable: χρησιμοποιείται για καθορισμό του πως γίνεται σύγκριση αντικειμένων (π.χ., String, Date)

  java.lang Interface Comparable

  ```
  int compareTo(T o)
  ```

- Καθορίζει μία φυσική σειρά για τα αντικείμενα όπως ισχύει για αριθμητικά δεδομένα, π.χ., 1<2, “ab”<”ac”
- Περιλαμβάνει την μέθοδο `compareTo:` Συγκρίνει το αντικείμενο this με το αντικείμενο που δίνεται σαν παράμετρος
- Ακολούθως συγκρίνουμε αντικείμενα σαν να είναι αριθμοί, π.χ.,

Παράδειγμα Χρήσης Comparable

```
class ComparableCircle
      extends Circle
        implements Comparable {

  public ComparableCircle(double radius) {
    super(radius);
  }

  /** Implement the compareTo method defined in Comparable */
  public int compareTo(Object o) {
    if (getRadius() > ((ComparableCircle) o).getRadius())
      return 1;
    else if (getRadius() < ((ComparableCircle) o).getRadius())
      return -1;
    else
      return 0;
  }
}
```

Πολυμορφισμός (polymorphism)

- **Πολύ:** πολλές, πολλαπλές, …
- **Μορφή:** χαρακτήρας, εμφάνιση, απεικόνιση, …

Πολλαπλές μορφές

- Ο πολυμορφισμός είναι **μία από τις πιο βασικές έννοιες του αντικειμενοστρεφή προγραμματισμού.**
- Σχετίζεται με την **αποσύνδεση των μεθόδων από τους τύπους**

Είδη Πολυμορφισμού

- **Υπερφόρτωση (Overloading)**
  - Μεθόδων (Method Overloading)
  - Τελεστών (Operator Overloading (C++, C#))
- **Υπερσκέλιση Μεθόδων (Method Overriding)**
- **Δυναμική Πρόσδεση (Late (Dynamic) Binding)**
- **Upcasting/Downcasting**

Παράδειγμα Method Overloading

```
public static void method() { … }
public static void method(int x) { … }
public static void method(int x, String y) {
… }
```

Constructor Overloading

```
class Circle {
  double radius;
  Circle () { this (1.0); }
  Circle (double newRadius) {
     radius = newRadius; }
}
```

Παράδειγμα Operator Overloading (C#)

Υποθέστε κλάση MyObj με μεταβλητές int a, int b

```
public static MyObj operator + (MyObj x, MyObj y) {
      return new MyObj(x.a+y.a, x.b+y.b);
}
```

```
public static MyObj operator == (MyObj x, MyObj y) {
      if (x.a == y.a && x.b == y.b) return true;
      return false;
}
```

```
public static MyObj operator != (MyObj x, MyObj y) {
      if (x.a != y.a || x.b != y.b) return true;
      return false;
}
```

Υπερσκέλιση (Override) vs.Υπερφόρτωση (Overload)

- Η Υπερφόρτωση (overload) μπορεί να πραγματοποιηθεί:
  - είτε στην ίδια κλάση με ορισμό μεθόδων με το ίδιο όνομα
  - ή μέσω κληρονομικότητας με ορισμό μεθόδων με το ίδιο όνομα

- **Η Υπερσκέλιση (Overriding) μπορεί να συμβεί μόνο μέσω κληρονομικότητας**

- Οι ακόλουθες δηλώσεις **μεθόδων ΔΕΝ μπορούν υπερσκελιθούν:**
  - **private**
  - **static (εκτός και αν…)**
  - **final**

Πρόσδεση (Binding)

- **Πρόσδεση (Binding) συμβαίνει όταν:**
  - Συνδέεται μία μεταβλητή με μία τιμή
  - Συνδέεται το κάλεσμα μίας μεθόδου με υλοποίησή της
- Δύο είδη Πρόσδεσης
  - **Early (static) Binding:** συμβαίνει πριν να τρέξει το πρόγραμμα από τον μεταγλωττιστή ή linker. Η compilers της C υποστηρίζουν μόνο αυτό.
  - **Late (dynamic) Binding:** συμβαίνει όταν η πρόσδεση γίνεται κατά τη διάρκεια εκτέλεσης του προγράμματος

Late (Dynamic) Binding

- Ο μεταγλωττιστής δεν γνωρίζει εκ’ των προτέρων ποιος τύπος θα χρησιμοποιηθεί
- Για να πραγματοποιηθεί το late binding πρέπει να υπάρχει κάποιος μηχανισμός που καθορίζει τον τύπου του αντικειμένου και την μέθοδο που πρέπει να καλεστεί την συγκεκριμένη στιγμή εκτέλεσης

- **Η JAVA πραγματοποιεί όλα τα method bindings με late binding**

- **Εξαίρεση:** αν οι μεθόδοι είναι δηλωμένες σαν static ή final
- **Προσοχή:** οι μεθόδοι που είναι δηλωμένες σαν private είναι εμμέσως final

Παράδειγμα: Late (Dynamic) Binding

```
public class PolymorphismDemo {
  public static void m (Object x)  {
    System.out.println(x.toString());}
  public static void main(String[] args) {
    m( new GraduateStudent() );
    m( new Student() );
    m( new Person() );
    m( new Object() );
  }
}
class GraduateStudent extends Student { }
class Student extends Person {
  public String toString() { return "Student"; }
}
class Person extends Object {
  public String toString() {return "Person"; }
}
```

Η μέθοδος m() παίρνει σαν παράμετρο Object

➔Μπορεί να καλεστεί με οποιοδήποτε αντικείμενο αφού όλα τα αντικείμενα κληρωνόμουν από το Object

Παράδειγμα: Late (Dynamic) Binding

```
public class PolymorphismDemo {
  public static void m (Object x)   {
    System.out.println(x.toString());}
  public static void main(String[] args) {
    m( new GraduateStudent() );
    m( new Student() );
    m( new Person() );
    m( new Object() );
  }
}
class GraduateStudent extends Student { }
class Student extends Person {
  public String toString() { return "Student"; }
}
class Person extends Object {
  public String toString() {return "Person"; }
}
```

Η μέθοδος m() μπορεί να καλεστεί με οποιοδήποτε αντικείμενο κληρονομεί από το αντικείμενο που δέχεται σαν παράμετρος

➔ **πολυμορφισμός**

Παράδειγμα: Late (Dynamic) Binding

```
public class PolymorphismDemo {
  public static void m (Object x)   {
    System.out.println(x.toString());}
  public static void main(String[] args) {
    m( new GraduateStudent() );
    m( new Student() );
    m( new Person() );
    m( new Object() );
  }
}
class GraduateStudent extends Student { }
class Student extends Person {
  public String toString() { return "Student"; }
}
class Person extends Object {
  public String toString() {return "Person"; }
}
```

Όταν εκτελείτε η μέθοδος m() το αντικείμενο μπορεί να είναι του τύπου:

- **GraduateStudent ή**
- **Student ή**
- **Person ή**
- **Object**

Η κάθε κλάση έχει δική της υλοποίηση για τη μέθοδο toString()

Ποια θα καλεστεί; Αποφασίζεται από την JVM run-time

Late (Dynamic) Binding

- Έστω ότι ένα αντικείμενο ο είναι ένα στιγμιότυπο της κλάσης C_1, C_2,… C_n.

- Αν το αντικείμενο ο καλέσει την μέθοδο p() το JVM ψάχνει την υλοποίηση της μεθόδου p() στις κλάσεις C_1, C_2,… C_n με την σειρά μέχρι να τη βρει.

- Η πρώτη εκτέλεση που θα βρεθεί, θα εκτελεστεί

Μετατροπές αντικειμένων (Casting Objects)

- Το casting μπορεί να χρησιμοποιηθεί για να μετατραπεί ένα αντικείμενο από ένα τύπο σε κάποιο άλλο της ιεραρχία της κληρονομικότητας

- Παράδειγμα: `m( new Student() );`
  μετατρέπει το Student σε Object

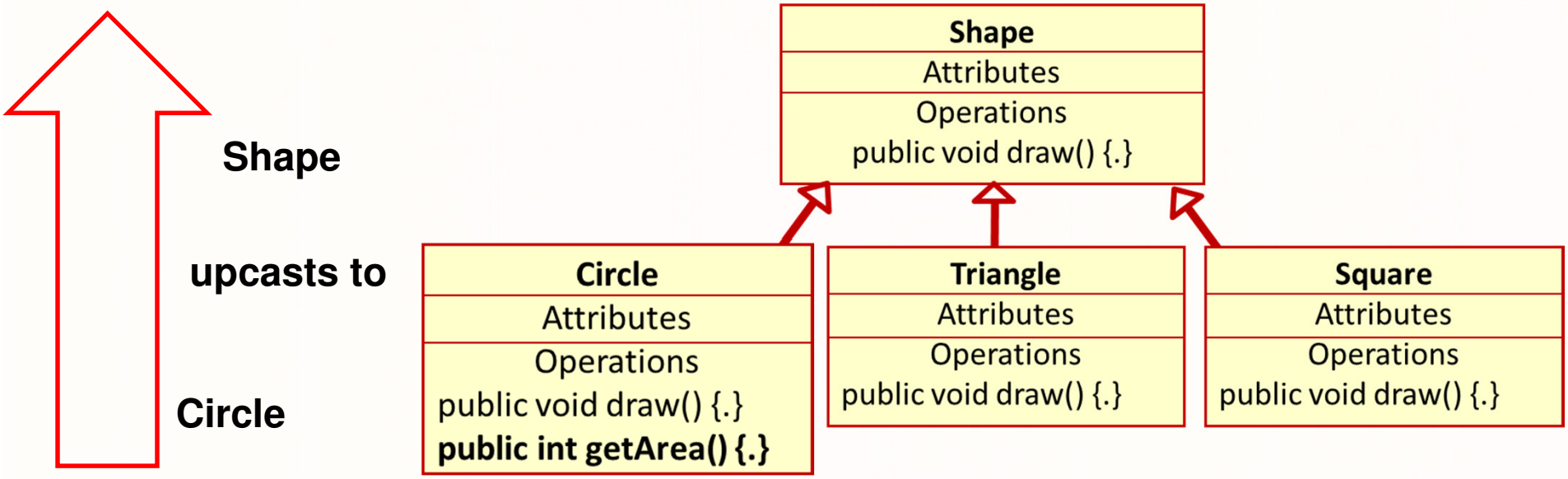
- Το πιο πάνω είναι ισοδύναμο με

```
Object o = new Student();
m( o );
```

- Έχουμε μετατρέψει ένα αντικείμενο μίας υποκλάσης σε ένα αντικείμενο μίας υπερκλάσης ➔ upcasting

Upcasting

- `Shape s = new Circle ();`



- Μέσα στο heap δημιουργείται ένα καινούριο αντικείμενο Circle (= new Circle).
- **Υπακούει όμως την διαπροσωπεία της κλάσης Shape**
- Αυτό «περιέχει» ένα αντικείμενο τύπου Shape
- `s.draw()`: late binding ➔πολυμορφική κλήση στην Circle.draw()
- s.getArea() ➔ **compile error (δεν υπακούει την διαπροσωπεία Shape)**

Upcasting με Abstract Classes/Interfaces

- Μία αφαιρετική κλάση μπορεί να χρησιμοποιηθεί σαν τύπος, π.χ., GeometricObject array = new GeometricObject[10]; (μέσω από πολυμορφισμό)
- Δείκτες σε αντικείμενα μπορούν να έχουν τύπο διαπροσωπείας (μέσω από πολυμορφισμό)
- Για παράδειγμα:
  `IAnimals a = new Lion();`
  - Το `IAnimals` είναι διαπροσωπεία
  - Το Lion είναι κλάση
- Τι έχουμε πετύχει με την πιο πάνω δήλωση;
- **Μόνο μεθόδοι που ανήκουν στην διαπροσωπεία `IAnimals` μπορούν να κληθούν!**

Down-Casting vs. Up-Casting

- Το πιο κάτω δημιουργεί σφάλμα μεταγλώττισης
  `Student s = new Object(); //downcasting`
- Αυτό `(Object o = new Student();)` όμως ΟΧΙ.
- Ο λόγος είναι ότι
  - Ένα αντικείμενο τύπου Student περιέχει/είναι ένα αντικείμενο τύπου Object
  - Το αντίστροφο όμως δεν ισχύει,
    π.χ., `Object o = new Apple(); //OK`
    `Student s = o;` σημαίνει ότι ο s είναι μήλο!
- Πρέπει να χρησιμοποιηθεί άμεσο/explicit casting για να επιτραπεί από τον μεταγλωττιστή
  π.χ., `Student s = (Student) new Object();`

Φωλιασμένες Κλάσεις (Nested Classes)

- Οι κλάσεις listener που υλοποιήσαμε έχουν σχεδιαστεί για να δημιουργηθούν διαχειριστές συμβάντων για διάφορα αντικείμενα (π.χ., για ένα κουμπί)
- Αυτές οι κλάσεις δεν θα χρησιμοποιηθούν από καμία άλλη κλάση είτε εντός ή εκτός εφαρμογής
- Για αυτό το λόγο είναι πιο σωστό αυτές οι κλάσεις να ισχύουν μόνο σε επίπεδο της κλάσης που έχουν οριστεί
- Αυτές οι κλάσεις αποτελούν «πεδία» των κλάσεων που έχουν οριστεί

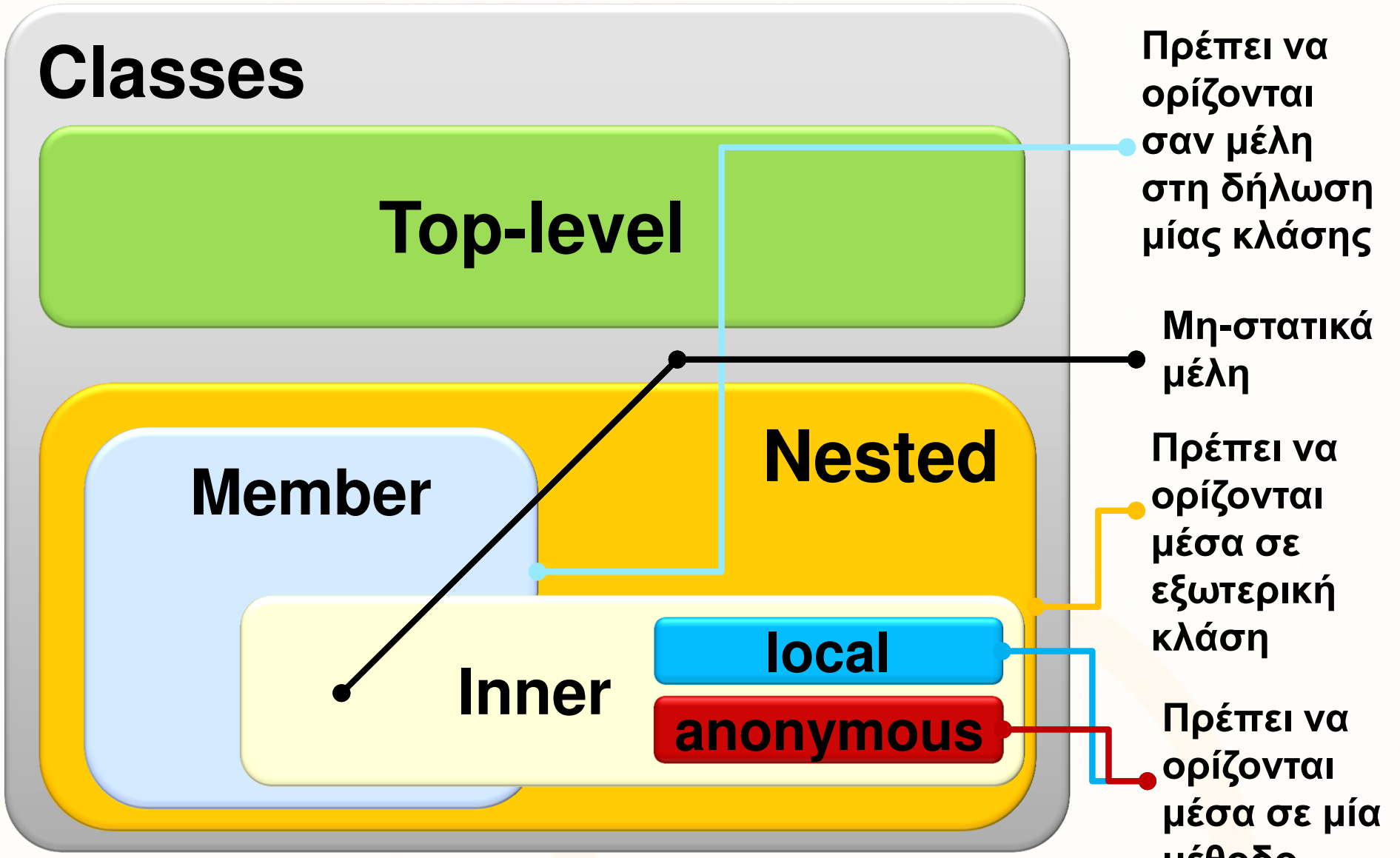
- **Κλάσεις που ορίζονται μέσα σε μία άλλη κλάση → Φωλιασμένες Κλάσεις (Nested Classes)**

Παράδειγμα Φωλιασμένων Κλάσεων

```
import …
public class NestedClassExample extends JFrame {
  // Create two buttons
  JButton jbtOK = new JButton("OK");
  JButton jbtCancel = new JButton("Cancel");

  class OKListenerClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("OK");}
  }

  class CanceListenerClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println("Cancel");}
  }

  public NestedClassExample () {
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
    add(jbtOK);
    add(jbtCancel);
    jbtOK.addActionListener(new OKListenerClass());
    jbtCancel.addActionListener(new CanceListenerClass());
  }

  public static void main(String[] args) {
    NestedClassExample frame = new NestedClassExample(); …
}
```

Παραδείγματα Φωλιασμένων Κλάσεων

Ταξινομία Κλάσεων



https://blogs.oracle.com/darcy/entry/nested_inner_member_and_top

Φωλιασμένες Κλάσεις-Αποθήκευση σε Αρχεία (συν.)

```
class OuterClass {

  static class NestedClass {}

  class InnerMemberClass {}

  OuterClass(){
    class InnerLocalClass { int x;}
  }

  private Object method() {
    return new Object() { int x=5; };
  }

  public static void main(
                          String[] args) {
    OuterClass o = new OuterClass();
  }
}
```

OuterClass.class

OuterClass$NestedClass.class

OuterClass$InnerMemberClass.class

OuterClass$1InnerLocalClass.class

OuterClass$1.class

Χρήσιμες Βιβλιοθήκες (java.lang.Math)

- Σταθερές Κλάσης:
  - PI: το π = 3.1415...
  - E: το e = 2.718...
- Μέθοδοι Κλάσης:
  - Trigonometric Methods: π.χ., sin(double a), cos(double a)
  - Exponent Methods: π.χ., log10(double a), pow(double a, double b), sqrt(double a)
  - Rounding Methods: π.χ., double ceil(double x), double floor(double x)
  - Other methods: min, max, abs, and random

- Παραδείγματα:

```
Math.sin(Math.PI / 2) = 1.0
Math.cos(0) = 1.0

Math.exp(1) = 2.71
Math.log(2.71) = 1.0
Math.pow(2, 3) = 8.0

Math.ceil(2.1) = 3.0
Math.floor(2.1) = 2.0
Math.ceil(-2.1) = -2.0

Math.max(2, 3) = 3
Math.min(2.5, 3.6) = 2.5
Math.abs(-2) = 2
0 <= Math.random() < 1.0
(int)(Math.random() * 10) = [0..9]
```