ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κλάσεις και Αντικείμενα

Constructors

Μαθήματα από το lab

- Ένα πρόγραμμα αποτελείται από διάφορες κλάσεις και αντικείμενα αυτών των κλάσεων.
- Μία από τις κλάσεις είναι η κύρια κλάση που περιέχει την main.
  - Η κλάση αυτή θα πρέπει να έχει το όνομα του αρχείου.
  - Η main είναι το σημείο έναρξης του προγράμματος.
- Πέραν της main ορίζουμε και επιπλέον κλάσεις και αντικείμενα αυτών των κλάσεων για το πρόγραμμα μας.
  - Στην Άσκηση 2 χρειαζόμαστε την κλάση Car.
  - Η κλάση Car ορίζεται ξεχωριστά από την κύρια κλάση

MovingCar

Αυτό το πρόγραμμα έπρεπε να τροποποιήσετε

```
class Car
{
    private int position = 0;

    public void move(int steps){
       position += steps;
    }

}
```

Δύο διαστάσεις

Δύο ορίσματα

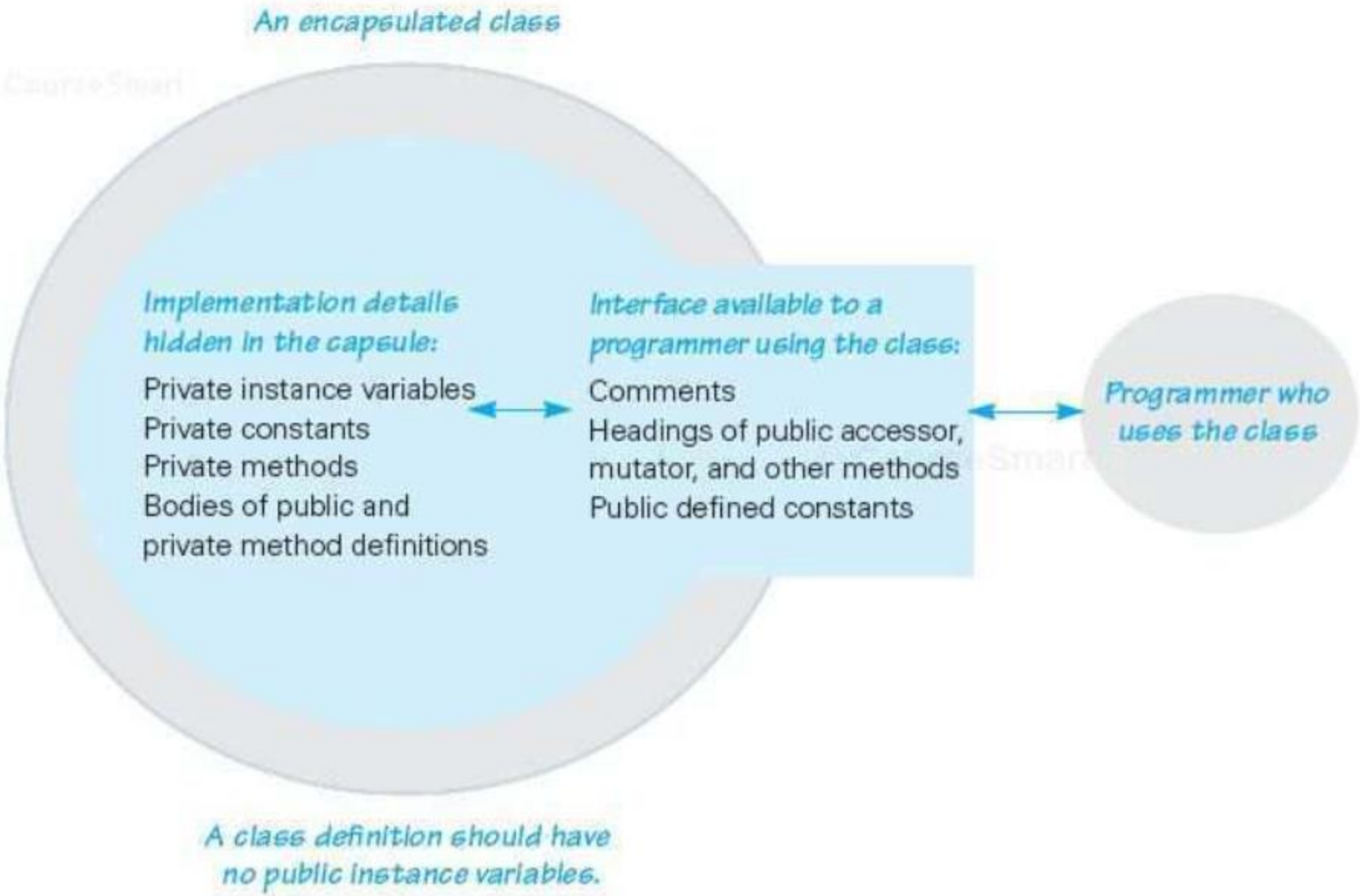
Έλεγχος ορίων και επιστροφή boolean

```
class MovingCar
{
    public static void main(String args[]){
       Car myCar = new Car();
       myCar.move();
    }
}
```

Loop όσο είναι σωστό

Ενθυλάκωση

- Η ομαδοποίηση λογισμικού και δεδομένων σε μία οντότητα (κλάση και αντικείμενα της κλάσης) ώστε να είναι εύχρηστη μέσω ενός καλά ορισμένου interface, ενώ οι λεπτομέρειες υλοποίησης είναι κρυμμένες από τον χρήστη.
- API (Application Programming Interface)[Έι-Πι-Άι]
  - Μια περιγραφή για το πώς χρησιμοποιείται η κλάση μέσω των public μεθόδων της.
    - Java docs είναι ένα παράδειγμα.
  - Το API είναι αρκετό για να χρησιμοποιήσετε μια κλάση, δεν χρειάζεται να ξέρετε την υλοποίηση των μεθόδων.


An encapsulated class
Implementation details hidden in the capsule:
Private instance variables
Private constants
Private methods
Bodies of public and private method definitions
Interface available to a programmer using the class:
Comments
Headings of public accessor, mutator, and other methods
Public defined constants
Programmer who uses the class
A class definition should have no public instance variables.

Accessor and Mutator methods

- Πολλές φορές χρειαζόμαστε να διαβάσουμε ή να αλλάξουμε ένα πεδίο ενός αντικειμένου
  - Π.χ., να διαβάσουμε τη θέση του οχήματος, ή να τοποθετήσουμε το όχημα σε μια συγκεκριμένη θέση.
  - Πως θα το κάνουμε αφού τα πεδία είναι private?
- Ορίζουμε ειδικές μεθόδους
  - Μέθοδος προσπέλασης (accessor method) για διάβασμα
  - Μέθοδος μεταλλαγής (mutator method) για γράψιμο
- Σύμβαση: Στη Java η ονοματολογία των μεθόδων αυτών γίνεται με συγκεκριμένο τρόπο:
  - get<ονομα μεταβλητης> για την πρόσβαση
    - getPosition
  - set<ονομα μεταβλητης> για την μετάλλαξη
    - setPosition

```
class Car
{
  private int position = 0;

  public void setPosition(int position){
    this.position = position;
  }

  public int getPosition(){
    return position;
  }

  public void move(){
    position ++ ;
  }
}

class MovingCar5
{
  public static void main(String args[]){
    Car myCar = new Car();
    myCar.setPosition(10);
    myCar.move();
    System.out.println(myCar.getPosition());
  }
}
```

Παράδειγμα

- Μία κλάση που να αποθηκεύει ημερομηνίες
  - Η κλάση θα παίρνει την ημέρα, μήνα και χρόνο σαν νούμερα (π.χ., 7 3 2013) και θα μπορεί να τυπώνει την ημερομηνία με το όνομα του μήνα (π.χ., 7 Μαρτίου 2013)
  - Στο πρόγραμμα βάλετε μια ημερομηνία και τυπώστε την.

Constructors (Δημιουργοί)

- Όταν δημιουργούμε ένα αντικείμενο συχνά θέλουμε να μπορούμε να το αρχικοποιήσουμε με κάποιες τιμές
  - Ένα Person να αρχικοποιείται με ένα όνομα
  - Ένα Car να αρχικοποιείται με μία θέση
- Μπορούμε να το κάνουμε με μία συνάρτηση set αυτό, αλλά
  - Μπορεί να έχουμε πολλές μεταβλητές να αρχικοποιήσουμε
  - Θέλουμε η αρχικοποίηση να είναι μέρος της δημιουργίας του αντικειμένου
- Την αρχικοποίηση μπορούμε να την κάνουμε με ένα Constructor (Δημιουργό)

Constructors (Δημιουργοί)

- Ο Constructor είναι μια «μέθοδος» η οποία καλείται όταν δημιουργούμε το αντικείμενο χρησιμοποιώντας την new.
- Αν δεν έχουμε ορίσει Constructor καλείται ένας default constructor χωρίς ορίσματα που δεν κάνει τίποτα.
- Αν ορίσουμε constructor, τοτε καλείται ο constructor που ορίσαμε.

Παράδειγμα

```
class Person
{
    private String name;

    public Person(String name){
        this.name = name;
    }

    public void speak(String s){
        System.out.println(name+":"+s);
    }
}

public class HelloWorld3
{
    public static void main(String[] args){
        Person alice = new Person("Alice");
        alice.speak("Hello World");
    }
}
```

Constructor: μια μέθοδος με το ίδιο όνομα όπως και η κλάση και χωρίς τύπο (ούτε void)

Αρχικοποιεί την μεταβλητή name

Constructor: καλείται όταν δημιουργείται το αντικείμενο με την new και μόνο τότε

Παράδειγμα

```
class Car
{
        private int position;

        public Car(int position){
                this.position = position;
        }

        public void move(int delta){
                position += delta ;
        }

}

class MovingCar7
{
        public static void main(String args[]){
                Car myCar1 = new Car(1);
                Car myCar2 = new Car(-1);
                myCar1.move(-1);
                myCar2.move(1);
        }
}
```

Παράδειγμα

```
class Car
{
  private int position=0;
  private int ACCELERATOR = 2;

  public Car(int position){
    this.position = position;
  }

  public void move(int delta){
    position += ACCELERATOR * delta ;
  }
}

class MovingCar8
{
  public static void main(String args[]){
          Car myCar1 = new Car(1);
          Car myCar2 = new Car(-1);
          myCar1.move(-1);
          myCar2.move(1);
  }
}
```

Η εκτέλεση αυτών των αρχικοποιήσεων γίνεται πριν εκτελεστούν οι εντολές στον constructor

Η τελική τιμή του position θα είναι αυτή που δίνεται σαν όρισμα

Υπερφόρτωση

- Είδαμε μια περίπτωση που είχαμε μια συνάρτηση move η οποία μετακινεί το όχημα κατά μία θέση, και μια συνάρτηση moveManySteps η οποία το μετακινεί όσες θέσεις ορίζει το όρισμα.
  - Το να θυμόμαστε δυο ονόματα είναι μπερδεμένο, θα ήταν καλύτερο να είχαμε μόνο ένα. Και στις δύο περιπτώσεις η λειτουργία που θέλουμε να κάνουμε είναι move
- Η Java μας δίνει αυτή τη δυνατότητα μέσω της διαδικασίας της υπερφόρτωσης (overloading)
  - Ορισμός πολλών μεθόδων με το ίδιο όνομα αλλά διαφορετικά ορίσματα, μέσα στην ίδια κλάση

```
class Car
{
        private int position;

        public Car(int position){
                this.position = position;
        }

        public void move(){
                position ++ ;
        }

        public void move(int delta){
                position += delta ;
        }
}

class MovingCar9
{
        public static void main(String args[]){
                Car myCar = new Car(1);
                myCar.move();
                myCar.move(-1);
        }
}
```

Υπερφόρτωση Δημιουργών

- Είναι αρκετά συνηθισμένο να υπερφορτώνουμε τους δημιουργούς των κλάσεων.

```
class Car
{
        private int position;

        public Car(){
                this.position = 0;
        }

        public Car(int position){
                this.position = position;
        }

        public void move(){
                position ++ ;
        }

        public void move(int delta){
                position += delta ;
        }

}

class MovingCar10
{
        public static void main(String args[]){
                Car myCar1 = new Car(1); myCar1.move();
                Car myCar2= new Car(); myCar2.move(-1);
        }
}
```

Υπερφόρτωση - Προσοχή

- Όταν ορίζουμε ένα constructor, ο default constructor παύει να υπάρχει. Πρέπει να τον ορίσουμε μόνοι μας.
- Η υπερφόρτωση γίνεται μόνο ως προς τα ορίσματα, ΌΧΙ ως προς την επιστρεφόμενη τιμή.
- Λόγω της συμβατότητας μεταξύ τύπων μια κλήση μπορεί να ταιριάζει με διάφορες μεθόδους. Καλείται αυτή που ταιριάζει ακριβώς, ή αυτή που είναι πιο κοντά.
- Αν υπάρχει ασάφεια θα χτυπήσει ο compiler.

```
class SomeClass
{
  public int aMethod(int x, double y){
    System.out.println("int double");
    return 1;
  }

  public double aMethod(int x, double y){
    System.out.println("int double");
    return 1;
  }

  public int aMethod(double x, int y){
    System.out.println("double int");
    return 1;
  }

  public double aMethod(double x, int y){
    System.out.println("double int");
    return 1;
  }

}
```

A

B

C

D

Ποιοι συνδυασμοί είναι αποδεκτοί?

A	B	✗
A	C	✓
A	D	✓
B	C	✓
B	D	✓
C	D	✗

```
class SomeClass
{

  public int aMethod(int x, int y){
    System.out.println("int int");
    return 1;
  }

  public float aMethod(float x, float y){
    System.out.println("float float");
    return 1;
  }

  public double aMethod(double x, double y){
    System.out.println("double double");
    return 1;
  }
}

class OverloadingExample
{
  public static void main(String args[])
  {
    SomeClass anObject = new SomeClass();
    anObject.aMethod(1,1);
  }
}
```

Τι θα τυπώσει η κλήση της μεθόδου?

Τυπώνει “int int”
γιατί ταιριάζει ακριβώς με τις παραμέτρους που δώσαμε

```
class SomeClass
{
  /*
  public int aMethod(int x, int y){
    System.out.println("int int");
    return 1;
  }
  */

  public float aMethod(float x, float y){
    System.out.println("float float");
    return 1;
  }

  public double aMethod(double x, double y){
    System.out.println("double double");
    return 1;
  }
}

class OverloadingExample
{
  public static void main(String args[])
  {
    SomeClass anObject = new SomeClass();
    anObject.aMethod(1,1);
  }
}
```

Τι θα τυπώσει η κλήση της μεθόδου?

Τυπώνει “float float”
γιατί είναι πιο κοντά ακριβώς με
τις παραμέτρους που δώσαμε

Ασάφεια

```
class SomeClass
{
  public double aMethod(int x, double y){
    System.out.println("int double");
    return 1;
  }

  public int aMethod(double x, int y){
    System.out.println("double int");
    return 1;
  }
}

class OverloadingExample
{
  public static void main(String args[])
  {
    SomeClass anObject = new SomeClass();
    anObject.aMethod(1.0,1);
    anObject.aMethod(1,1);
  }
}
```

Τι θα τυπώσει η κλήση της μεθόδου σε κάθε περίπτωση?

Τυπώνει “double int”

Ο compiler μας πετάει λάθος γιατί η κλήση είναι ασαφής (ambiguous)