Παράδειγμα

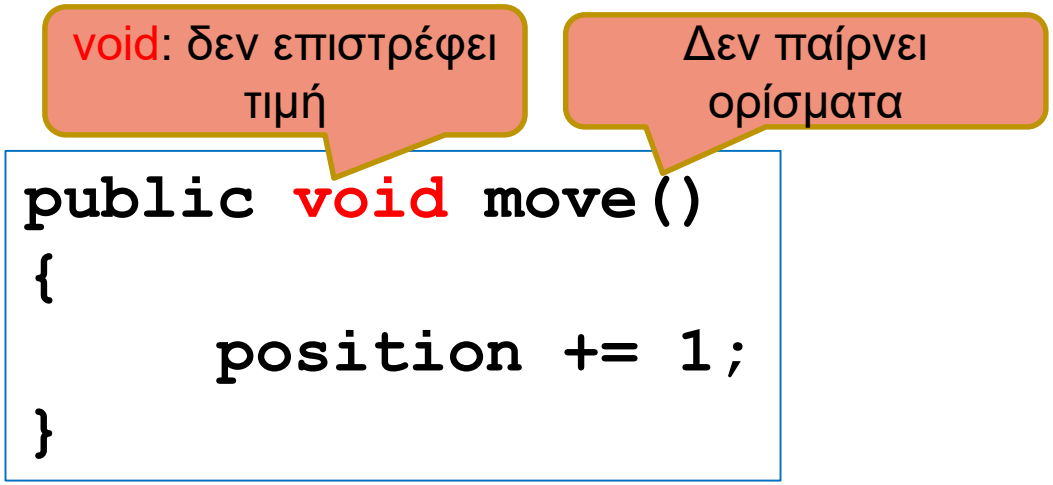
- Θέλουμε ένα πρόγραμμα που να προσομοιώνει την κίνηση ενός αυτοκινήτου, το οποίο κινείται και τυπώνει τη θέση του.

MovingCar

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

    public void move(){
        position += 1;
    }

    public void printPosition(){
        System.out.println("Car at position "+position);
    }
}

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

Μέθοδοι

- Οι μέθοδοι που έχουμε δει μέχρι τώρα είναι πολύ απλές
  - Δεν έχουν παραμέτρους (δεν παίρνουν ορίσματα)
  - Δεν επιστρέφουν τιμή



Παράδειγμα 2

- Εκτός από την κίνηση κατά μία θέση θέλουμε να μπορούμε να κινούμε το όχημα όσες θέσεις θέλουμε είτε προς τα δεξιά (+) είτε προς τα αριστερά (-).

Παράμετροι

- Οι μέθοδοι μπορούν να έχουν παραμέτρους
  - Μας επιτρέπουν να περάσουμε τιμές στην μέθοδο μας

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

Ορισμός παραμέτρου

- Μία παράμετρος ορίζεται όπως οποιαδήποτε άλλη μεταβλητή.
  - Πρέπει να έχει συγκεκριμένο τύπο και όνομα
  - Είναι τοπική μεταβλητή της μεθόδου

```
int x = 10;
myCar.moveManySteps(x);
myCar.moveManySteps(10);
```

Όρισμα στην κλήση της μεθόδου

- Όταν καλούμε την μέθοδο, περνάμε ένα το όρισμα
  - Το όρισμα είναι μια έκφραση (κάτι που θα μπορούσε να είναι στο δεξί μέρος μιας ανάθεσης)
  - Θα πρέπει να συμφωνεί στον τύπο με την παράμετρο
  - Είναι σαν να κάνουμε ανάθεση `steps = x` ή `steps = 10`

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

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

class MovingCar2
{
    public static void main(String args[])
    {
        Car myCar = new Car();
        int x = 10;
        myCar.moveManySteps(x);
        myCar.moveManySteps(10);
        myCar.moveManySteps(2*x+10);
    }
}
```

Στον ορισμό της μεθόδου ορίζουμε και την παράμετρο της μεθόδου, όπως ορίζουμε μια μεταβλητή. Έχει ένα τύπο και ένα όνομα

Όταν καλούμε την μέθοδο περνάμε μια τιμή σαν όρισμα στην μέθοδο
Σαν όρισμα μπορεί να είναι μια οποιαδήποτε έκφραση.
Αρκεί ή αποτίμηση της έκφρασης να έχει τύπο συμβατό με αυτόν της παραμέτρου (int στην περίπτωση μας)

Κατά την κλήση της μεθόδου ουσιαστικά εκχωρείται η τιμή της έκφρασης στην μεταβλητή steps. Αυτό λέγεται και πέρασμα παραμέτρου.

ΜΑΘΗΜΑ 4

6. ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ - ΜΕΘΟΔΟΙ

Παράδειγμα 1

- Θέλουμε ένα πρόγραμμα που να προσομοιώνει την κίνηση ενός αυτοκινήτου, το οποίο κινείται πάνω σε μία ευθεία πάντα κατά μία θέση, και τυπώνει τη θέση του.

MovingCar

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

    public void move(){
        position += 1;
    }

    public void printPosition(){
        System.out.println("Car at position "+ position);
    }
}

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

Ορισμός κλάσης

Ορισμός (και αρχικοποίηση) πεδίου

Ορισμός μεθόδου

Χρήση πεδίου

Ορισμός αντικειμένου

Κλήση μεθόδου

Παράδειγμα 2

- Θέλουμε να μπορούμε να κινούμε το όχημα όσες θέσεις θέλουμε είτε προς τα δεξιά (+) είτε προς τα αριστερά (-).

- Για να το κάνουμε αυτό η move θα πρέπει να παίρνει σαν παράμετρο τον αριθμό των θέσεων

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

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

class MovingCar2
{
    public static void main(String args[])
    {
        Car myCar = new Car();
        int x = 10;
        myCar.moveManySteps(x);
        myCar.moveManySteps(10);
        myCar.moveManySteps(2*x+10);
    }
}
```

Στον ορισμό της μεθόδου ορίζουμε και την παράμετρο της μεθόδου, όπως ορίζουμε μια μεταβλητή. Έχει ένα τύπο και ένα όνομα

Όταν καλούμε την μέθοδο περνάμε μια τιμή σαν όρισμα στην μέθοδο
Σαν όρισμα μπορεί να είναι μια οποιαδήποτε έκφραση.
Αρκεί ή αποτίμηση της έκφρασης να έχει τύπο συμβατό με αυτόν της παραμέτρου (int στην περίπτωση μας)

Κατά την κλήση της μεθόδου ουσιαστικά εκχωρείται η τιμή της έκφρασης στην μεταβλητή steps. Αυτό λέγεται και πέρασμα παραμέτρου.

Πέρασμα παραμέτρων

- Όταν καλούμε μια μέθοδο με μία τιμή σαν όρισμα, ουσιαστικά εκχωρούμε αυτή την τιμή στην παράμετρο της μεθόδου

Η κλήση

```
myCar.moveManySteps(2*x+10);
```

όπου η μεταβλητή x έχει την τιμή 10

Αποτιμάται η τιμή της έκφρασης και εκχωρείται

Ισοδυναμεί με τον κώδικα:

```
{
        int steps = 30;
        position += steps;
}
```

Το πέρασμα μεταβλητών με αυτό τον τρόπο λέγεται πέρασμα δια τιμής (pass by value). Η μέθοδος δεν έχει πρόσβαση στην μεταβλητή μόνο στην τιμή

Πέρασμα παραμέτρων δια τιμής

- Όταν το πέρασμα παραμέτρων γίνεται δια τιμής, το πρόγραμμα μας έχει πρόσβαση μόνο στην τιμή της παραμέτρου και όχι στην μεταβλητή που χρησιμοποιήσαμε στο όρισμα.
  - Σε όλες τις γλώσσες πλέον το πέρασμα παραμέτρων γίνεται δια τιμής

- Αν η παράμετρος είναι ένα αντικείμενο τα πράγματα γίνονται πιο σύνθετα
  - Η τιμή της μεταβλητής που έχουμε σαν παράμετρο είναι διεύθυνση μνήμης. Δεν μπορούμε να αλλάξουμε την διεύθυνση μνήμης αλλά μπορούμε να αλλάξουμε τα περιεχόμενα της.

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

    public void moveManySteps(int steps, String direction)
    {
       if (direction.equals("right"){ position += steps;}
       if (direction.equals("left") { position -= steps;}
    }
}

class MovingCar3
{
    public static void main(String args[]){
       Car myCar = new Car();
       myCar.moveManySteps(10, "left");
    }
}
```

Μέθοδος με πολλές παραμέτρους

Τα ορίσματα θα πρέπει να συμφωνούν με το πλήθος και τους τύπους των παραμέτρων στην αντίστοιχη θέση

Κλήση της μεθόδου

Τύποι παραμέτρων και ορισμάτων

- Οι παράμετροι μιας μεθόδου έχουν συγκεκριμένο τύπο
- Τα ορίσματα στην κλήση της μεθόδου θα πρέπει να συμφωνούν με τον τύπο της παραμέτρου, θέση προς θέση.
- Ισχύουν οι μετατροπές τύπου που ξέρουμε
  - byte → short → int → long → float → double

- Μία μέθοδος μπορεί να πάρει ως όρισμα και ένα αντικείμενο μιας κλάσης.
  - Το πώς δουλεύει αυτό θα το μάθουμε όταν μιλήσουμε για αναφορές.

Μέθοδοι που επιστρέφουν τιμές

- Μέχρι τώρα οι μέθοδοι που φτιάξαμε δεν επιστρέφουν τιμή
  - Είναι τύπου void.

- Σε πολλές περιπτώσεις θέλουμε η μέθοδος να μας επιστρέφει τιμή
  - Π.χ., μία μέθοδος που υπολογίζει το άθροισμα δύο αριθμών

Παράδειγμα 3

- Το αυτοκίνητο μας δεν μπορεί να μετακινηθεί έξω από το διάστημα [-10,10]. Θέλουμε η moveManySteps να μας επιστρέφει μια λογική τιμή αν η μετακίνηση έγινε η όχι.

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

    public boolean moveManySteps(int steps)
    {
       if ((position + steps < -10) || (position + steps > 10)){
          return false;
       }else{
           position += steps;
           return true;
       }
    }
}
```

Όταν ορίζουμε μια μέθοδο που επιστρέφει τιμή θα πρέπει να ορίσουμε τον τύπο της τιμής που επιστρέφει.

Π.χ. αυτή η μέθοδος επιστρέφει τιμή boolean

Μια μέθοδος μπορεί να επιστρέφει και ένα αντικείμενο μιας κλάσης

Επιστρέφουμε μια τιμή μέσα στον κώδικα χρησιμοποιώντας την εντολή return.

Η εντολή return

- Η εντολή return χρησιμοποιείται για να επιστρέψει μια τιμή μια μέθοδος.
- Συντακτικό:
  - `return <έκφραση>`
- Κάθε μονοπάτι εκτέλεσης του κώδικα θα πρέπει να επιστρέφει μια τιμή.
- Η κλήση της return σε οποιοδήποτε σημείο του κώδικα σταματάει την εκτέλεση της μεθόδου και επιστρέφει τιμή.
  - Μπορούμε να το χρησιμοποιήσουμε αυτό για να απλοποιήσουμε τον κώδικα.

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

    public boolean moveManySteps(int steps)
    {
       if ((position + steps < -10) || (position + steps > 10)){
            return false;
       }
       position += steps;
       return true;
    }
}
```

Αν μπούμε μέσα στο if η return θα σταματήσει την εκτέλεση του κώδικα και θα μας βγάλει από την μέθοδο. Επιστρέφεται η τιμή false.

Δεν χρειάζεται πλέον το else

Ο τύπος μιας μεθόδου

- Μια μέθοδος που επιστρέφει τιμή ορίζεται με συγκεκριμένο τύπο. Π.χ.
  - `public boolean moveManySteps(int steps)`
  - `public double division(int x, int y)`
  - `public String getUserName()`
  - `public Car getCar()`

- Αν έχουμε μια συνάρτηση που επιστρέφει τιμή τύπου T
  - Π.χ. `public double division(int x, int y)`

η έκφραση στο return πρέπει να επιστρέφει μία τιμή τύπου (συμβατού με το) T. (π.χ., `return x/(double)y`)

```
import java.util.Scanner;

class Car
{
    private int position = 0;

    public boolean moveManySteps(int steps){
        if ((position + steps < -10) || (position + steps > 10)){
            return false;
        }
        position += steps;
        return true;
    }
    public void printPosition(){
        System.out.println("Car at position "+position);
    }
}

class MovingCar4b{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        Car myCar = new Car();
        int steps = input.nextInt();
        boolean carMoved = myCar.moveManySteps(steps);
        if (carMoved) { myCar.printPosition();}
        else { System.out.println("Car could not move");}
    }
}
```

Κλήση της μεθόδου

```
import java.util.Scanner;

class Car
{
    private int position = 0;

    public boolean moveManySteps(int steps)
    {
        if ((position + steps < -10) || (position + steps > 10)){
            return false;
        }
        position += steps;
        return true;
    }
    public void printPosition(){
        System.out.println("Car at position "+position);
    }
}

class MovingCar4c
{
    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        Car myCar = new Car();
        int steps = input.nextInt();
        myCar.moveManySteps(steps);
        myCar.printPosition();
    }
}
```

Δεν είναι υποχρεωτικό να χρησιμοποιούμε πάντα την επιστρεφόμενη τιμή

Η moveManySteps επιστρέφει τιμή, αλλά η κλήση της την αγνοεί

Η printPosition θα επιστρέψει 0 αν δεν κινήθηκε το όχημα

Η εντολή return

- Μπορούμε να καλέσουμε την return και σε μία void μέθοδο
  - Χωρίς επιστρεφόμενη τιμή.
    - `return;`
  - Σταματάει την εκτέλεση της μεθόδου

```
public void printIfPositive()
{
    if (position < 0){
        return;
    }
    System.out.println("position = " + position);
}
```

Η εντολή return

- Μπορούμε να καλέσουμε την return και σε μία void μέθοδο
  - Χωρίς επιστρεφόμενη τιμή.
    - `return;`
  - Σταματάει την εκτέλεση της μεθόδου

```
public void moveManySteps(int steps, String direction)
{
        if (steps < 0){
                return;
        }
        if (direction.equals("right"){ position += steps;}
        if (direction.equals("left") { position -= steps;}
}
```

Παράδειγμα 4

- Θέλουμε να μπορούμε να κινούμε το όχημα όσες θέσεις θέλουμε είτε προς τα δεξιά (+) είτε προς τα αριστερά (-), και να τυπώνεται η θέση σε κάθε κίνηση.

- Υλοποίηση: Θα ορίσουμε μια βοηθητική μεταβλητή delta την οποία θα προσθέτουμε στο position σε κάθε βήμα. Η default τιμή του θα είναι delta = 1. Αν η παράμετρος steps είναι αρνητική θα την μετατρέπουμε σε θετική και θα θέσουμε delta = -1.

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

    public void moveManySteps(int steps)
    {
        int delta = 1;
        if (steps < 0){
            steps = -steps; delta = -1;
        }
        for (int i = 0; i < steps; i ++){
            position += delta;
            System.out.println("Car at position "+position);
        }
    }
    public void printPosition(){
        System.out.println("Car at position "+position);
    }
}

class MovingCar5
{
    public static void main(String args[]){
        Car myCar = new Car();
        int steps = -10;
        myCar.moveManySteps(steps);
        System.out.println("--: " + steps);
    }
}
```

Το `delta` είναι τοπική μεταβλητή της μεθόδου. Ορίζεται μέσα στην μέθοδο και υπάρχει μόνο μέσα στην μέθοδο. Στο τέλος της μεθόδου η μεταβλητή χάνεται.

Η παράμετρος `steps` λειτουργεί ως τοπική μεταβλητή της συνάρτησης και χάνεται μετά την κλήση της μεθόδου.

Η μεταβλητή `steps` στην main είναι διαφορετική από την παράμετρος `steps` . Το πέρασμα παραμέτρων γίνεται δια τιμής και άρα η τιμή της μεταβλητής του ορίσματος δεν μεταβάλλεται

Τυπώνει `--:-10`

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

    public void moveManySteps(int steps)
    {
        int delta = 1;
        if (steps < 0){
            steps = -steps; delta = -1;
        }
        for (int i = 0; i < steps; i ++){
            position += delta;
            printPosition();
        }
    }

    public void printPosition(){
        System.out.println("Car at position "+position);
    }
}
```

Μπορούμε να κάνουμε την εκτύπωση καλώντας την printPosition()

Κάθε μέθοδος που ορίζουμε μέσα σε μία κλάση μπορούμε να την χρησιμοποιήσουμε και μέσα στην κλάση

Παράδειγμα 4

- Όταν καλούμε την συνάρτηση move() το όχημα μας θα κινείται ένα τυχαίο αριθμό από βήματα στο διάστημα (-3,3)

Υλοποίηση

- Θα φτιάξουμε μια βοηθητική συνάρτηση που θα μας επιστρέφει τον τυχαίο αριθμό από βήματα.

private: δεν χρειάζεται να φαίνεται έξω από την κλάση

```
private int computeRandomSteps()
{
     int radomSteps;
     // do the computation

     return randomSteps;
}

public void move(){
     int steps = computeRandomSteps();
     moveManySteps(steps);
}
```

Κλήση της συνάρτησης και χρήση της επιστρεφόμενης τιμής

```
import java.util.Random;

class Car
{
  private int MAX_VALUE = 3;
  private int position = 0;
  private Random randomGenerator = new Random();

  private int computeRandomSteps()
  {
     int randomSteps = randomGenerator.nextInt(2*MAX_VALUE + 1) - MAX_VALUE;
     return randomSteps;
  }

  public void move(){
     int steps = computeRandomSteps();
     moveManySteps(steps);
  }

  public void moveManySteps(int steps) { ... }

  public void printPosition(){
    System.out.println("Car at position "+position);
  }
}

class MovingCar6
{
  public static void main(String args[]){
    Car myCar = new Car();
    myCar.move();
  }
}
```

Η κλάση Random: Δημιουργεί μια γεννήτρια τυχαίων αριθμών που παράγει τυχαίους αριθμούς

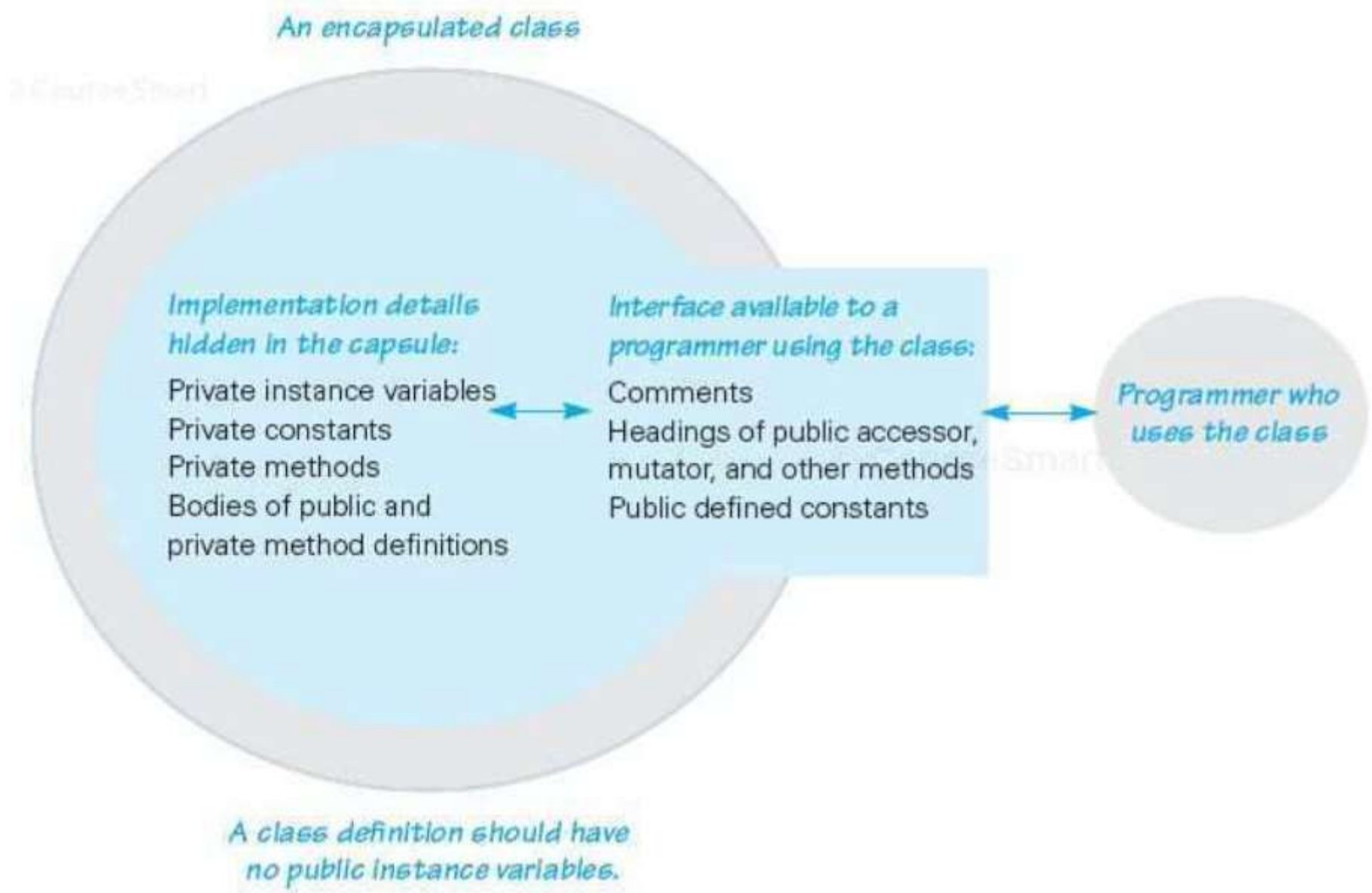
Μέθοδος nextInt(int x) της Random: Επιστρέφει ένα τυχαίο ακέραιο αριθμό στο διάστημα [0, x)

Public/Private

- Ότι είναι ορισμένο ως public σε μία κλάση είναι προσβάσιμο από οποιονδήποτε.
  - Μπορούμε να καλέσουμε τις μεθόδους ορίζοντας ένα αντικείμενο της κλάσης
- Ότι είναι ορισμένο ως private σε μία κλάση είναι προσβάσιμο μόνο από την ίδια κλάση.
- Ο τροποποιητής private μας επιτρέπει την απόκρυψη πληροφοριών (information hiding).
  - Ο χρήστης της κλάσης Car, δεν χρειάζεται να ξέρει πως υλοποιείται η μέθοδος `computeRandomSteps` που υπολογίζει τον τυχαίο αριθμό των βημάτων.
  - Αν αποφασίσουμε να αλλάξουμε κάτι στη μέθοδο αυτό θα γίνει ως μέρος του επανασχεδιασμού της κλάσης Car. Κανείς άλλος δεν θα πρέπει να επηρεαστεί από την αλλαγή στον κώδικα.
- Τα πεδία μιας κλάσης τα ορίζουμε πάντα private.

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.
- ADT (Abstract Data Type)
  - Ένας τύπος δεδομένων που ορίζεται χρησιμοποιώντας την αρχή της ενθυλάκωσης
    - Οι λίστες που χρησιμοποιήσατε στην Python είναι ένα παράδειγμα.
    - Δεδομένα και μέθοδοι.


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 p){
    position = p;
  }

  public int getPosition(){
    return position;
  }

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

class MovingCar7
{
  public static void main(String args[]){
    Car myCar = new Car();
    myCar.setPosition(10);
    myCar.move();
    System.out.println(myCar.getPosition());
  }
}
```

Υπάρχουν περιπτώσεις που μπορεί να θέλουμε η συνάρτηση set να επιστρέφει boolean (true αν η ανάθεση έγινε επιτυχώς, false αλλιώς)

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

        public boolean setPosition(int position){
                if (position < 0){
                        return false;
                }
                this.position = position;
                return true;
        }

        public int getPosition(){
                return position;
        }

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

class MovingCar9
{
        public static void main(String args[]){
                Car myCar = new Car();
                boolean check = myCar.setPosition(-1);
                if (!check){
                        System.out.println("position not set");
                }
        }
}
```

Η setPosition μπορεί να επιστρέφει τιμή
Το πιο συνηθισμένο είναι να επιστρέφει boolean αν έγινε σωστά η ανάθεση

Τοπικές μεταβλητές

- Οι τοπικές μεταβλητές (και οι παράμετροι) που ορίζουμε μέσα σε μία μέθοδο, έχουν προτεραιότητα σε σχέση με τα πεδία της μεθόδου
  - Δηλαδή αν έχουμε μια τοπική μεταβλητή με το ίδιο όνομα με ένα πεδίο μέσα σε μία μέθοδο, όταν χρησιμοποιούμε το όνομα αναφερόμαστε στην τοπική μεταβλητή και όχι στο πεδίο.
  - Αν θέλουμε να αναφερθούμε στο πεδίο μπορούμε να χρησιμοποιήσουμε την δεσμευμένη λέξη this.

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

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

  public int getPosition(){
    return position;
  }

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

class MovingCar7
{
  public static void main(String args[]){
    Car myCar = new Car();
    myCar.setPosition(10);
    myCar.move();
    System.out.println(myCar.getPosition());
  }
}
```

Το this.position αναφέρεται στο πεδίο του αντικειμένου.
Το position αναφέρεται στην παράμετρο της συνάρτησης

Το κρυφό πεδίο this προσδιορίζει το αντικείμενο που κάλεσε την μέθοδο

Έτσι μπορούμε να χρησιμοποιήσουμε το ίδιο όνομα μεταβλητής χωρίς να δημιουργείται σύγχυση

```
class LocalVariableTest
{
        private int var = 10;

        public void method1(){
                int var = 5;
                var ++;
        }

        public void method2(int var){
                var ++;
        }

        public void method3(){
                int var = 1;
                this.var = var;
        }

        public void printVar(){
                System.out.println("var = "+var);
        }

        public static void main(String[] args){
                LocalVariableTest x = new LocalVariableTest();
                x.method1(); x.printVar();
                x.method2(3); x.printVar();
                x.method3(); x.printVar();
        }
}
```

Ορισμός του πεδίου var

Ορισμός τοπικής μεταβλητής var.
Η χρήση της var μεσα στην μέθοδο αναφέρεται στην τοπική μεταβλητή

Ορισμός παραμέτρου var.
Η χρήση της var μεσα στην μέθοδο αναφέρεται στην τοπική μεταβλητή

Ορισμός τοπικής μεταβλητής var.
Η χρήση της var μεσα στην μέθοδο αναφέρεται στην τοπική μεταβλητή.
Το this.var αναφέρεται στο πεδίο της κλάσης

Τι θα τυπώσει?

var = 10

var = 10

var = 1

Παρένθεση: Μπορούμε να ορίσουμε main μέσα σε μία κλάση για να την τεστάρουμε

7. CONSTRUCTORS – ΥΠΕΡΦΟΡΤΩΣΗ – ΑΝΤΙΚΕΙΜΕΝΑ ΩΣ ΠΑΡΑΜΕΤΡΟΙ

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.

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 MovingCar7
{
  public static void main(String args[]){
    Car myCar = new Car();
    myCar.setPosition(10);
    myCar.move();
    System.out.println(myCar.getPosition());
  }
}
```

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

        public boolean setPosition(int position){
                if (position < 0){
                        return false;
                }
                this.position = position;
                return true;
        }

        public int getPosition(){
                return position;
        }

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

class MovingCar8
{
        public static void main(String args[]){
                Car myCar = new Car();
                boolean check = myCar.setPosition(-1);
                if (!check){
                        System.out.println("position not set");
                }
        }
}
```

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, τότε 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 HelloWorld2
{
    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

Μια συνομιλία

```
class Person
{
    private String name;

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

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

public class Conversation
{
    public static void main(String[] args){
        Person alice = new Person("Alice");
        Person bob = new Person("Bob");
        alice.speak("Hi Bob");
        bob.speak("Hi Alice");
    }
}
```

Παράδειγμα

```
class Car
{
        private int position;

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

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

        public void printPosition(){
                System.out.println("Car is at position "+position);
        }

}

class MovingCar9
{
        public static void main(String args[]){
                Car myCar1 = new Car(1);
                Car myCar2 = new Car(-1);
                myCar1.move(-1); myCar1.printPosition();
                myCar2.move(1); myCar2.printPosition();
        }
}
```

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 ;
   }

   public void printPosition(){
     System.out.println("Car is at position "+position);
   }
}

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

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 MovingCar11
{
        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 MovingCar12
{
        public static void main(String args[]){
                Car myCar1 = new Car(1); myCar1.move();
                Car myCar2= new Car(); myCar2.move(-1);
        }
}
```

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

        public Car(){}

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

        public void move(){
                position ++ ;
        }

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

}

class MovingCar12
{
        public static void main(String args[]){
                Car myCar1 = new Car(1); myCar1.move();
                Car myCar2= new Car(); myCar2.move(-1);
        }
}
```

Κενός κώδικας, χρειάζεται για να οριστεί ο “default” constructor

Γενικά είναι καλό να ορίζετε και ένα constructor χωρίς ορίσματα

Υπερφόρτωση – Προσοχή Ι

- Όταν ορίζουμε ένα constructor, ο default constructor παύει να υπάρχει. Πρέπει να τον ορίσουμε μόνοι μας.

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

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

        public void move(){
                position ++ ;
        }

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

}

class MovingCar12
{
        public static void main(String args[]){
                Car myCar1 = new Car(1);
                myCar1.move();
                Car myCar2= new Car();
                myCar2.move(-1);
        }
}
```

Θα χτυπήσει λάθος ότι δεν υπάρχει constructor χωρίς ορίσματα

Υπερφόρτωση – Προσοχή II

- Η υπερφόρτωση γίνεται μόνο ως προς τα ορίσματα, ΟΧΙ ως προς την επιστρεφόμενη τιμή.
- Η υπογραφή μίας μεθόδου είναι το όνομα της και η λίστα με τους τύπους των ορισμάτων της μεθόδου
  - Η Java μπορεί να ξεχωρίσει μεθόδους με διαφορετική υπογραφή.
  - Π.χ., move(), move(int) έχουν διαφορετική υπογραφή
- Όταν δημιουργούμε μια μέθοδο θα πρέπει να δημιουργούμε μία διαφορετική υπογραφή.

```
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	

Υπερφόρτωση – Προσοχή III

- 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, 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)