

ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κλάσεις και Αντικείμενα
Μέθοδοι

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);  
    }  
}
```

Ορισμός κλάσης

Ορισμός (και αρχικοποίηση) πεδίου

Ορισμός μεθόδου

Χρήση πεδίου

```
class MovingCar
```

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

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 += delta;  
}
```

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;
```

Μέθοδος με πολλές παραμέτρους

```
    public void moveManySteps(int steps, String direction)
```

```
    {
```

```
        if (direction.equals("right") { position += steps;}
```

```
        if (direction.equals("left") { position -= steps;}
```

```
    }
```

```
}
```

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

```
class MovingCar3
```

```
{
```

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

```
        Car myCar = new Car();
```

```
        myCar.moveManySteps(10, "left");
```

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.

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

Π.χ. αυτή η μέθοδος επιστρέφει τιμή boolean

Μια μέθοδος μπορεί να επιστρέφει και ένα αντικείμενο μιας κλάσης

```
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;
    }
}
```

Επιστρέφουμε μια τιμή μέσα στον κώδικα χρησιμοποιώντας την εντολή **return**.

Η εντολή return

- Η εντολή **return** χρησιμοποιείται για να επιστρέψει μια τιμή μια μέθοδος.
- ΣΥΝΤΑΚΤΙΚΟ:
 - **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)[‘Ει-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;
```

Ορισμός του πεδίου var

```
public void method1() {
```

```
int var = 5;
```

```
var ++;
```

```
}
```

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

```
public void method2(int var) {
```

```
var ++;
```

```
}
```

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

```
public void method3() {
```

```
int var = 1;
```

```
this.var = var;
```

```
}
```

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

```
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();
```

```
}
```

```
}
```

Τι θα τυπώσει?

var = 10

var = 10

var = 1

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