

ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Δημιουργία Κλάσεων και Αντικειμένων
Τοπικές μεταβλητές
Το αντικείμενο this
Accessor, Mutator μέθοδοι

Ορισμός κλάσης και αντικειμένων

- Μία κλάση **K** ορίζεται από
 - Κάποιες **μεταβλητές** τις οποίες ονομάζουμε **πεδία**
 - Κάποιες **συναρτήσεις** που τις ονομάζουμε **μεθόδους**.
 - Οι μέθοδοι «**βλέπουν**» τα πεδία της κλάσης
- Ένα **αντικείμενο** ορίζεται ως μια **μεταβλητή τύπου K**
 - Το αντικείμενο έχει συγκεκριμένες **τιμές** στα πεδία.
 - Στο πρόγραμμα έχουμε (συνήθως) **πρόσβαση** μόνο τις **μεθόδους**.
 - Μέσω των μεθόδων έχουμε πρόσβαση στα πεδία
 - Αν υπάρχουν κάποια **πεδία** στα οποία έχουμε πρόσβαση αυτά τα λέμε **properties**.

μέλη
της
κλάσης

Κλάσεις και αντικείμενα

- Ορισμός κλάσης:

```
class <Όνομα Κλάσης>
{
    <Ορισμός πεδίων κλάσης>

    <Ορισμός μεθόδων κλάσης>
}
```

- Ορισμός αντικειμένου:

```
<Όνομα Κλάσης> myObject = new <Όνομα Κλάσης>();
```

- Ο ορισμός του αντικειμένου γίνεται συνήθως μέσα στη **main** ή μέσα στη μέθοδο μίας **άλλης κλάσης** που χρησιμοποιεί το αντικείμενο

Light

```
class Light
```

```
{
```

```
    private boolean lightIsOn = false;
```

```
    public void flipSwitch()
```

```
    {
```

```
        lightIsOn = !lightIsOn;
```

```
    }
```

```
}
```

```
class HouseWithLights
```

```
{
```

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

```
    {
```

```
        Light bedroomLight = new Light();
```

```
        bedroomLight.flipSwitch();
```

```
    }
```

```
}
```

Ορισμός κλάσης

Ορισμός
(και αρχικοποίηση) πεδίου

Ορισμός μεθόδου τύπου
void χωρίς παραμέτρους

Χρήση πεδίου

Ορισμός αντικειμένου

Κλήση μεθόδου

Παράδειγμα

- Δημιουργήστε μια κλάση η οποία μοντελοποιεί ένα χρονόμετρο που κρατάει λεπτά και δευτερόλεπτα. Το χρονόμετρο δεν μπορεί να μετρήσει πάνω από μία ώρα, μετά μηδενίζει. Υπάρχει μια μέθοδος `tick` η οποία προχωράει το χρονόμετρο κατά ένα δευτερόλεπτο. Θέλουμε επίσης και μια μέθοδο `reset` και μία που μας τυπώνει την κατάσταση του ρολογιού.

```
class Chronometer
{
    private int seconds = 0;
    private int minutes = 0;

    public void tick()
    {
        if (seconds < 59){
            seconds ++;
        }else if (minutes < 59){
            seconds = 0;
            minutes ++;
        }else{
            seconds = 0;
            minutes = 0;
        }
    }

    public void reset()
    {
        seconds = 0;
        minutes = 0;
    }

    public void printTimePassed(){
        System.out.println(minutes + " minutes and " + seconds + " seconds");
    }
}

class ChronoTest
{
    public static void main(String[] args)
    {
        Chronometer timer = new Chronometer();
        for (int i = 0; i < 100; i ++){
            timer.tick();
        }
        timer.printTimePassed();
    }
}
```

Παράδειγμα

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

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**

void: δεν επιστρέφει τιμή

Δεν παίρνει ορίσματα

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

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 **πaráμετρος** 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 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**.

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

```
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;}
```

```
    }
```

```
}
```

```
class MovingCar3
```

```
{
```

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

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

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

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

Κλήση της μεθόδου

```
    }
```

```
}
```

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.

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 **παράμετρο steps** στην μέθοδο. Το πέρασμα παραμέτρων γίνεται δια τιμής και άρα η τιμή της μεταβλητής του **ορίσματος δεν** μεταβάλλεται

Τυπώνει --: -10

Τοπικές μεταβλητές

- Είδαμε πρώτη φορά τις **τοπικές μεταβλητές** όταν μιλήσαμε για μεταβλητές που ορίζονται μέσα σε ένα λογικό block.
 - Παρόμοια είναι και για τις μεταβλητές μιας **μεθόδου**.
- Τοπικές μεταβλητές μιας μεθόδου είναι οι μεταβλητές που ορίζονται **μέσα** στον κώδικα της μεθόδου
 - Περιλαμβάνουν και τις μεταβλητές που κρατάνε τις **παραμέτρους** της μεθόδου
- Οι μεταβλητές αυτές έχουν **εμβέλεια** μόνο **μέσα στην μέθοδο**
 - **Εξαφανίζονται** όταν **βγούμε** από τη μέθοδο.
- Αντιθέτως τα **πεδία** της κλάσης διατηρούνται όσο υπάρχει το **αντικείμενο**, και έχουν εμβέλεια σε **όλη** την κλάση

```
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

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

Η εντολή return

- Η εντολή **return** χρησιμοποιείται για να επιστρέψει μια τιμή μια μέθοδος.
- ΣΥΝΤΑΚΤΙΚΟ:
 - **return** <έκφραση>
- Ο **τύπος** της **έκφρασης** στην εντολή **return** θα πρέπει να είναι ίδιος (ή συμβατός) με τον **τύπο** της **μεθόδου**.
- **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.

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 **τύπο** 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;
    }
    position += steps;
    return true;
}
```

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

```
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();
```

```
        if (myCar.moveManySteps(steps) ) {
```

```
            myCar.printPosition();
```

```
        }
```

```
        else { System.out.println("Car could not move");}
```

```
    }
```

```
}
```

Κλήση της μεθόδου και χρήση του αποτελέσματος απευθείας μέσα στην συνθήκη. Δεν χρειάζεται να το αποθηκεύσουμε.

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`)

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

- 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(<τιμή>)

```
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 p){
        if (p < 0){
            return false;
        }
        position = p;
        return true;
    }
}
```

```
    public int getPosition(){
        return position;
    }

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

```
class MovingCar7b
{
    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**.

Το αντικείμενο this

- Με την δεσμευμένη λέξη **this** ένα αντικείμενο αναφέρεται στον εαυτό του.

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

    public void setPositionToTen(){
        this.position = 10;
    }
}

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

Το **this.position** αναφέρεται στο εσωτερικό πεδίο position του αντικείμενου που θα καλέσει την μέθοδο.

Όταν δημιουργηθεί το αντικείμενο myCar και καλέσει την μέθοδο setPositionToTen το **this** αναφέρεται πλέον στο αντικείμενο που κάλεσε την μέθοδο, δηλαδή το myCar

```

class Car
{
    private int position = 0;

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

    public int getPosition() {
        return position;
    }

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

class MovingCar8
{
    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 παράμετρο

```
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);
```

```
}
```

Δεν υπάρχει τοπική μεταβλητή,
χρήση του πεδίου var

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

```
LocalVariableTest x = new LocalVariableTest();
```

```
int var = 3;
```

```
x.method1(); x.printVar();
```

```
x.method2(var); x.printVar();
```

```
x.method3(); x.printVar();
```

```
System.out.println(var);
```

```
}
```

```
}
```

Τι θα τυπώσει?

var = 10

var = 10

var = 1

var = 3

Ορισμός και χρήση
τοπικής μεταβλητής
var στην main.

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

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.
 - Ο default constructor απλά εκτελεί τις αρχικοποιήσεις.
- Αν ορίσουμε constructor, τότε καλείται ο constructor που **ορίσαμε**.

ΣΥΝΤΑΚΤΙΚΟ

- Ο constructor είναι μια μέθοδος:
 - Που έχει το όνομα της κλάσης
 - Ορίζεται πάντα **public**
 - Δεν έχει **τύπο**

```
class <Όνομα κλάσης>
{
    <Ορισμός Πεδίων>
    public <Όνομα κλάσης> ( [ορίσματα] )
    {
        [κώδικας] ;
    }
}
```

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