Προχωρημένος Προγραμματισμός

Δημιουργία Κλάσεων και Αντικειμένων

ΕΛΕΥΘΕΡΙΟΣ ΚΟΣΜΑΣ

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ 2022-2023 | ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Περίληψη

Σήμερα ...

- θα συζητήσουμε τον τρόπο ορισμού και δημιουργίας δικών μας κλάσεων και αντικειμένων
- θα μελετήσουμε τον τρόπο ορισμού των πεδίων και των μεθόδων των κλάσεων
- θα παρουσιάσουμε τον τρόπο ορισμού των παραμέτρων των μεθόδων και την έννοια το περάσματος δια τιμής
- θα μιλήσουμε για την κλήση μεθόδων των κλάσεων και την επιστροφή τιμών από αυτές
- θα παρουσιάσουμε την έννοια της ενθυλάκωσης, μία από τις βασικές έννοιες του αντικειμενοστραφούς προγραμματισμού

Κλάση & Αντικείμενο
Σύνοψη

- Κλάση: μια αφηρημένη περιγραφή αντικειμένων με κοινά χαρακτηριστικά και κοινή συμπεριφορά
  - ένα καλούπι που παράγει αντικείμενα

- αντικείμενο: ένα στιγμιότυπο μίας κλάσης

- η κλάση ορίζει τον τύπο του αντικειμένου
  - τα χαρακτηριστικά του αντικειμένου
  - τις ενέργειες που μπορεί να επιτελέσει

Κλάση & Αντικείμενο

Πρακτικά στον κώδικα

μία κλάση Κ ορίζεται από:

- κάποιες μεταβλητές τις οποίες ονομάζουμε πεδία
- κάποιες συναρτήσεις που τις ονομάζουμε μεθόδους
  - οι μέθοδοι «βλέπουν» τα πεδία της κλάσης

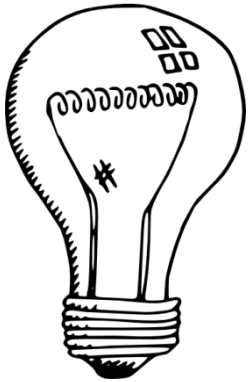
ένα αντικείμενο ορίζεται ως μια μεταβλητή τύπου Κ

- το αντικείμενο έχει συγκεκριμένες τιμές στα πεδία
- στο πρόγραμμα έχουμε (συνήθως) πρόσβαση μόνο στις μεθόδους
  - μέσω των μεθόδων έχουμε πρόσβαση στα πεδία
- αν υπάρχουν κάποια πεδία στα οποία έχουμε πρόσβαση αυτά τα λέμε ιδιότητες (properties)

Κλάση & Αντικείμενο

Παράδειγμα: Δημιουργώντας φως - Περιγραφή

- Θα φτιάξουμε μια κλάση που θα χειρίζεται ένα διακόπτη φωτός
  - το φως είναι είτε ανοιχτό είτε κλειστό και
  - μπορούμε να ανοιγοκλείνουμε το φως



Όνομα κλάσης
Πεδία κλάσης
Μέθοδοι κλάσης

`Light`
`boolean lightIsOn`
`flipSwitch()`

Κλάση & Αντικείμενο

- 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

Κλάση & Αντικείμενο

Ορισμός μελών κλάσης

- ορισμός πεδίων:
  - τα πεδία είναι μεταβλητές, ορίζονται όπως οι υπόλοιπες μεταβλητές, με την εξής διαφορά:
    - τα πεδία πρέπει να τα προσδιορίσουμε ως `private` ή `public`
    - τα πεδία τα ορίζουμε (σχεδόν) πάντα ως `private`
      - τις ιδιότητες τις ορίζουμε ως `public`

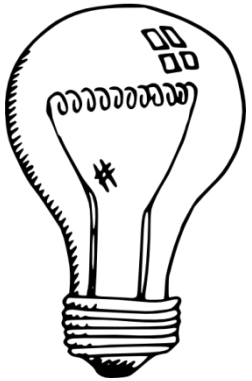
```
[private/public] <τύπος> <όνομα μεταβλητής> [ = τιμή];
```

- ορισμός μεθόδων:
  - οι μέθοδοι είναι συναρτήσεις και έχουν τον εξής ορισμό:

```
[private/public] <τύπος> <όνομα συνάρτησης> ([παράμετροι])
{
        <κώδικας συνάρτησης>
}
```

Κλάση & Αντικείμενο

Παράδειγμα: Δημιουργώντας φως - Υλοποίηση



```
1.  // ορισμός κλάσης και χρήση αντικειμένων Light
2.
3.  class Light                                    // ορισμός κλάσης
4.  {
5.     private boolean lightIsOn = false;          // ορισμός και αρχικοποίηση πεδίου
6.
7.     public void flipSwitch()                    // ορισμός μεθόδου
8.     {
9.        lightIsOn = !lightIsOn;                  // χρήση πεδίου
10.    }
11. }
12.
13. class HouseWithLights
14. {
15.    public static void main(String args[])
16.    {
17.       Light bedroomLight = new Light();        // ορισμός αντικειμένου
18.       bedroomLight.flipSwitch();               // κλήση μεθόδου
19.    }
20. }
```

Κλάση & Αντικείμενο

Ορισμός public/private πεδίων και μεθόδων

- οτιδήποτε είναι ορισμένο ως `public` σε μία κλάση **A** είναι προσβάσιμο από μία άλλη κλάση που ορίζει ένα αντικείμενο τύπου **A**
  - π.χ. η μέθοδος `flipSwitch()` είναι προσβάσιμη από την κλάση `HouseWithLights` μέσω του αντικειμένου `bedroomLight`

- οτιδήποτε είναι ορισμένο ως `private` σε μία κλάση δεν είναι προσβάσιμο από μία άλλη κλάση
  - π.χ. το πεδίο `lightIsOn` δεν είναι προσβάσιμο από την κλάση `HouseWithLights` μέσω του αντικειμένου `bedroomLight`

Κλάση & Αντικείμενο

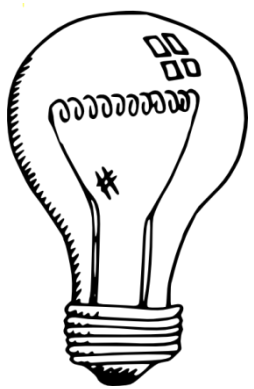
Ορισμός public/private πεδίων και μεθόδων II

- μπορούμε να έχουμε `public` και `private` πεδία και μεθόδους
  - κανόνας: τα πεδία τα ορίζουμε (σχεδόν) ΠΑΝΤΑ `private`
  - οι μέθοδοι που χρειάζονται να καλούνται από αντικείμενα είναι `public`
  - αυτές που είναι βοηθητικές είναι `private`

- τα πεδία και οι μέθοδοι μίας κλάσης, ανεξάρτητα αν είναι `public` ή `private`, είναι προσβάσιμα από όλες τις μεθόδους και τα αντικείμενα της ίδιας κλάσης
  - π.χ., το πεδίο `lightIsOn` είναι προσβάσιμο παντού μέσα στην κλάση `Light`, και σε οποιοδήποτε άλλο αντικείμενο τύπου `Light`
  - για να ξεχωρίζουν τα πεδία από άλλες μεταβλητές, κάποιοι συνηθίζουν να βάζουν '_' στην αρχή του ονόματος των πεδίων
    - π.χ. `_lightIsOn`

Κλάση & Αντικείμενο

Παράδειγμα: Δημιουργώντας φως - Υλοποίηση II



```
1.  // ορισμός κλάσης και χρήση αντικειμένων Light
2.
3.  class Light
4.  {
5.     private boolean lightIsOn = false;
6.
7.     public void flipSwitch() {
8.        lightIsOn = !lightIsOn;
9.     }
10.
11.    public void printState() {
12.       if (lightIsOn) {
13.          System.out.println("The light is on");
14.       } else {
15.          System.out.println("The light is off");
16.       }
17.    }
18. }
```

Η **κατάσταση** ενός αντικειμένου προσδιορίζεται από τις **τιμές** που έχουν τα πεδία της κλάσης

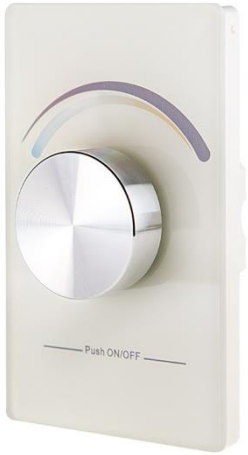
Η μόνη **πρόσβαση** που έχουμε στην κατάσταση του αντικειμένου είναι **μέσω** των **μεθόδων** της κλάσης.

```
19. class HouseWithLights
20. {
21.    public static void main(String args[])
22.    {
23.       Light bedroomLight = new Light();
24.       bedroomLight.flipSwitch();
25.       bedroomLight.printState();
26.       Light kitchenLight = new Light();
27.       kitchenLight.flipSwitch();
28.       kitchenLight.printState();
29.    }
30. }
```

Κλάση & Αντικείμενο

Παράδειγμα: Dimmer - Περιγραφή

- Θα φτιάξουμε μια κλάση που θα χειρίζεται ένα διακόπτη φωτός
  - το φως είναι είτε ανοιχτό είτε κλειστό και
  - μπορούμε να ανοιγοκλείνουμε το φως
  - θέλουμε ο διακόπτης μας να μας δίνει τη δυνατότητα να αυξομειώνουμε την ένταση
    - ? τι επιπλέον πεδία πρέπει να προσθέσουμε?
    - ? Τι επιπλέον μεθόδους χρειαζόμαστε?




```
Dimmer
boolean lightIsOn
int intensity
flipSwitch()
flipSwitch()
dim()
brighten()
```

Κλάση & Αντικείμενο

Παράδειγμα: Dimmer - Υλοποίηση

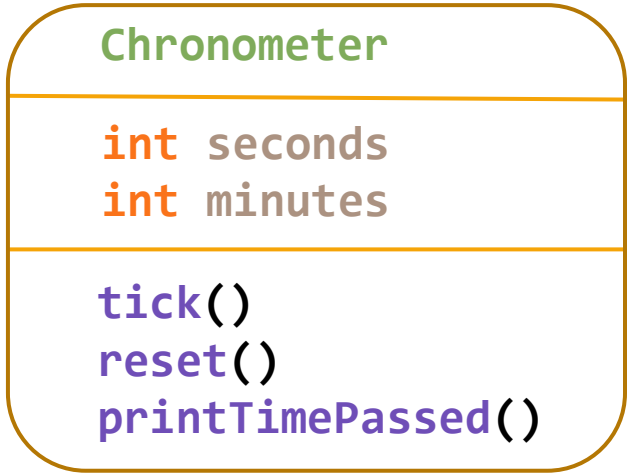
```
1.  class DimmerLight
2.  {
3.      private boolean lightIsOn = false;
4.      private int intensity = 100;
5.
6.      public void flipSwitch() {
7.          lightIsOn = !lightIsOn;
8.      }
9.
10.     public void dim() {
11.         if (intensity > 0)
12.             intensity--;
13.     }
14.
15.     public void brighten() {
16.         if (intensity < 100)
17.             intensity++;
18.     }
19.
20.     public void printState() {
21.         if (lightIsOn) {
22.             System.out.println("The light is on with intensity " + intensity);
23.         } else {
24.             System.out.println("The light is off");
25.         }
26.     }
27. }
```

```
28. class HouseWithDimmerLights
29. {
30.     public static void main(String args[])
31.     {
32.         DimmerLight bedroomLight = new DimmerLight();
33.         bedroomLight.flipSwitch();
34.         bedroomLight.dim();
35.         bedroomLight.printState();
36.     }
37. }
```

Κλάση & Αντικείμενο

Παράδειγμα: Χρονόμετρο - Περιγραφή

- θέλουμε να δημιουργήσουμε μία κλάση η οποία μοντελοποιεί ένα χρονόμετρο που κρατάει λεπτά και δευτερόλεπτα
  - το χρονόμετρο μπορεί να μετρήσει το πολύ μία ώρα
    - μετά μηδενίζει
  - η μέθοδος `tick` προχωράει το χρονόμετρο κατά ένα δευτερόλεπτο
  - η μέθοδος `reset` μηδενίζει το χρονόμετρο
  - η μέθοδος `printTimePassed` τυπώνει την κατάσταση του χρονομέτρου



Κλάση & Αντικείμενο

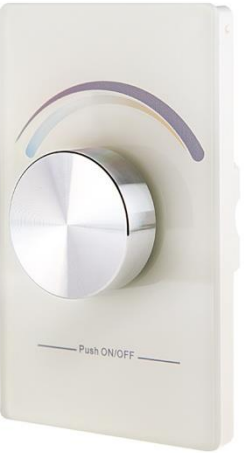
Παράδειγμα: Χρονόμετρο - Υλοποίηση

```
1.  class Chronometer
2.  {
3.      private int seconds = 0;
4.      private int minutes = 0;
5.
6.      public void tick() {
7.          if (seconds < 59) {
8.              seconds++;
9.          } else if (minutes < 59) {
10.             seconds = 0;
11.             minutes++;
12.         } else {
13.             seconds = 0;
14.             minutes = 0;
15.         }
16.     }
17.
18.     public void reset() {
19.             seconds = 0;
20.             minutes = 0;
21.     }
22.
23.     public void printTimePassed() {
24.        System.out.println(minutes + "minutes and " + seconds + " seconds");
25.     }
26. }
```

```
27. class ChronometerTest
28. {
29.     public static void main(String args[])
30.     {
31.         Chronometer timer = new Chronometer();
32.         for (int i = 0; i < 100; i++) {
33.             timer.tick();
34.         }
35.         timer.printTimePassed();
36.     }
37. }
```

Κλάση & Αντικείμενο

Παράδειγμα: Dimmer II - Περιγραφή



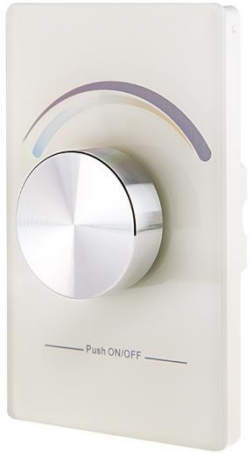
- Θα φτιάξουμε μια κλάση που θα χειρίζεται ένα διακόπτη φωτός
  - το φως είναι είτε ανοιχτό είτε κλειστό και
  - μπορούμε να ανοιγοκλείνουμε το φως
  - θέλουμε ο διακόπτης μας να μας δίνει τη δυνατότητα να αυξομειώνουμε την ένταση
  - κάθε φορά που αυξάνουμε ή μειώνουμε την ένταση θέλουμε να μας λέει και την κατανάλωση
    - κατανάλωση = ένταση * 0.1 λεπτά/ώρα

```
Dimmer
---
boolean lightIsOn
int intensity
---
flipSwitch()
dim()
brighten()
```

Κλάση & Αντικείμενο

Παράδειγμα: Dimmer II - Υλοποίηση

```
1.  class DimmerLight2
2.  {
3.      private boolean lightIsOn = false;
4.      private int intensity = 100;
5.
6.      public void flipSwitch() {
7.          lightIsOn = !lightIsOn;
8.      }
9.
10.     public void dim() {
11.         if (intensity > 0)
12.             intensity--;
13.         double consumption = intensity * 0.1;
14.         System.out.println("Consumption = " + consumption);
15.     }
16.
17.     public void brighten() {
18.         if (intensity < 100)
19.             intensity++;
20.         double consumption = intensity * 0.1;
21.         System.out.println("Consumption = " + consumption);
22.     }
23.
24.     public void printState() {
25.         ...
26.     }
27. }
```



Οι μεταβλητές consumption είναι **τοπικές** μεταβλητές

υπάρχουν **μόνο** μέσα στις μεθόδους `dim` και `brighten` και όταν **τελειώσει** η κλήση τους **εξαφανίζονται**

Τοπικές Μεταβλητές

- είδαμε πρώτη φορά τις τοπικές μεταβλητές όταν μιλήσαμε για μεταβλητές που ορίζονται μέσα σε ένα λογικό σύνολο (block) εντολών
  - παρόμοια είναι και για τις μεταβλητές μιας μεθόδου

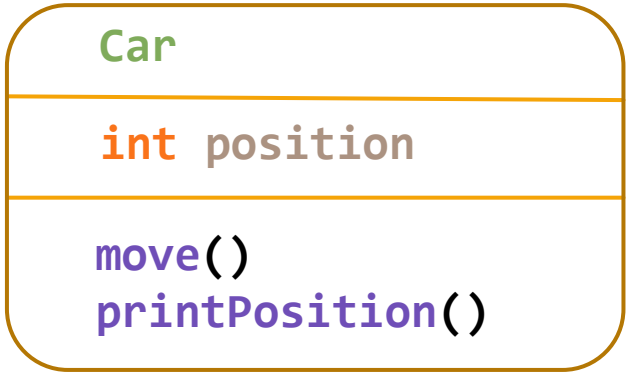
- τοπικές μεταβλητές μιας μεθόδου
  - είναι οι μεταβλητές που ορίζονται μέσα στον κώδικα της μεθόδου
    - περιλαμβάνουν και τις μεταβλητές που κρατάνε τις παραμέτρους της μεθόδου
  - έχουν εμβέλεια μόνο μέσα στην μέθοδο
  - εξαφανίζονται όταν βγούμε από τη μέθοδο

- αντιθέτως, τα πεδία της κλάσης
  - έχουν εμβέλεια σε όλη την κλάση
  - διατηρούνται όσο υπάρχει το αντικείμενο

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο - Περιγραφή

- θέλουμε να αναπτύξουμε ένα πρόγραμμα που προσομειώνει την κίνηση ενός αυτοκινήτου, το οποίο κινείται και τυπώνει τη θέση του



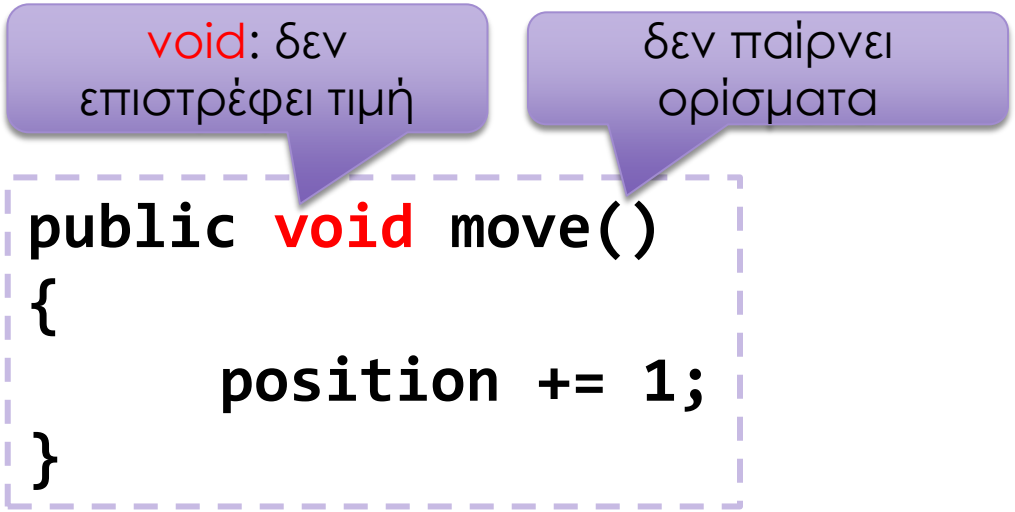
Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο - Υλοποίηση

```
1.  class Car
2.  {
3.     private int position = 0;
4.
5.     public void move() {
6.        position++;
7.     }
8.
9.     public void printPosition() {
10.       System.out.println("Car at position: " + position);
11.    }
12. }
13.
27. class MovingCar
28. {
29.    public static void main(String args[])
30.    {
31.       Car myCar = new Car();
32.       myCar.move();
33.       myCar.printPosition();
34.    }
35. }
```

Μέθοδοι

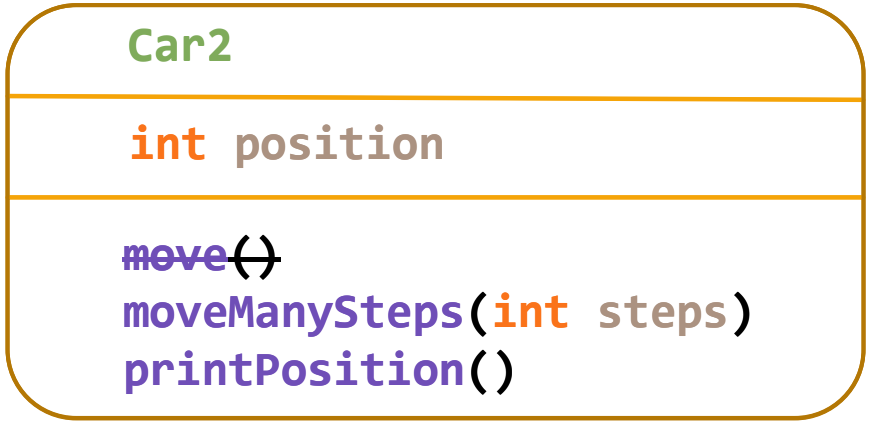
- οι μέθοδοι που έχουμε δει μέχρι τώρα είναι πολύ απλές
  - δεν έχουν παραμέτρους (δεν παίρνουν ορίσματα)
  - δεν επιστρέφουν τιμή



Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο ΙΙ - Περιγραφή

- θέλουμε να αναπτύξουμε ένα πρόγραμμα που προσομειώνει την κίνηση ενός αυτοκινήτου, το οποίο κινείται και τυπώνει τη θέση του
- επιπρόσθετα της κίνησης κατά μία θέση, θέλουμε να μπορούμε να κινούμε το όχημα όσες θέσεις θέλουμε



Παράμετροι

- οι μέθοδοι μπορούν να έχουν παραμέτρους
  - μας επιτρέπουν να περάσουμε τιμές στη μέθοδο μας

ορισμός παραμέτρου

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

- 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 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
  - π.χ. είναι σαν να κάνουμε ανάθεση `steps = x` ή `steps = 10`

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο ΙΙ - Υλοποίηση

```
1.  class Car2
2.  {
3.      private int position = 0;
4.
5.      public void moveManySteps(int steps) {
6.          position += steps;
7.      }
8.
9.      public void printPosition() {
10.         System.out.println("Car at position: " + position);
11.     }
12. }
13.
27. class MovingCar2
28. {
29.     public static void main(String args[])
30.     {
31.         Car2 myCar = new Car2();
32.         int x = 10;
33.         myCar.moveManySteps(x);
34.         myCar.moveManySteps(10);
35.         myCar.moveManySteps(2*x+10);
36.     }
37. }
```

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 στη μέθοδο.
Ως 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 στη 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

π.χ.

- η κλήση:
  - όπου η μεταβλητή `x` έχει την τιμή `10`
- ισοδυναμεί με τον κώδικά:

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

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

αποτιμάται η τιμή της έκφρασης και εκχωρείται

- το πέρασμα μεταβλητών με αυτό τον τρόπο λέγεται πέρασμα δια τιμής (pass by value)
- η μέθοδος έχει πρόσβαση μόνο στην τιμή (και όχι στη μεταβλητή)

Πέρασμα παραμέτρων

Δια τιμής (pass by value)

- η μέθοδος έχει πρόσβαση μόνο στην τιμή της παραμέτρου
  - και όχι στην μεταβλητή που χρησιμοποιήσαμε στο όρισμα
  - ☞ επομένως, δε μπορούμε να αλλάξουμε την τιμή της μεταβλητής που χρησιμοποιήσαμε στο όρισμα

- σε όλες τις γλώσσες προγραμματισμού πλέον το πέρασμα παραμέτρων γίνεται δια τιμής

- αν η παράμετρος είναι ένα αντικείμενο, τότε:
  - ☞ η τιμή της μεταβλητής που έχουμε σαν παράμετρο είναι διεύθυνση μνήμης
  - ✘ δε μπορούμε να αλλάξουμε τη διεύθυνση μνήμης *(γιατί;)*
  - ✓ αλλά, μπορούμε να αλλάξουμε τα περιεχόμενα της

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο III - Περιγραφή

- θέλουμε να αναπτύξουμε ένα πρόγραμμα που προσομειώνει την κίνηση ενός αυτοκινήτου, το οποίο κινείται και τυπώνει τη θέση του
- επιπρόσθετα της κίνησης κατά μία θέση, θέλουμε να μπορούμε να κινούμε το όχημα
  - όσες θέσεις θέλουμε
  - είτε προς τα μπροστά (+) είτε προς τα πίσω (-)

```
Car3
int position
moveManySteps(int steps, String direction)
printPosition()
```

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο III - Υλοποίηση

μέθοδος με πολλές παραμέτρους

```
1.    class Car3
2.    {
3.       private int position = 0;
4.
5.       public void moveManySteps(int steps, String direction) {
6.          if (direction.equals("forward")) {
7.             position += steps;
8.          }
9.          else if (direction.equals("backward")) {
10.            position -= steps;
11.         }
12.      }
13.
14.      public void printPosition() {
15.         System.out.println("Car at position: " + position);
16.      }
17.   }
18.
27.   class MovingCar3
28.   {
29.      public static void main(String args[])
30.      {
31.         Car3 myCar = new Car3();
32.         myCar.moveManySteps(10, "backward");
33.      }
34.   }
```

κλήση της μεθόδου

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

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 τιμή μια μέθοδος

- συντακτικό:

```
return <έκφραση>
```

- κάθε μονοπάτι εκτέλεσης του κώδικα της μεθόδου 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 για να απλοποιήσουμε τον κώδικα

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο IV - Περιγραφή

- το αυτοκίνητο μας δε μπορεί να μετακινηθεί έξω από το διάστημα [-10,10]
  - θέλουμε η `moveManySteps` να μας επιστρέφει μια λογική τιμή αν η μετακίνηση έγινε η όχι

```
Car4
int position
boolean moveManySteps(int steps)
void printPosition()
```

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο IV - Υλοποίηση

```
1.  class Car4
2.  {
3.     private int position = 0;
4.
5.     public boolean moveManySteps(int steps)
6.     {
7.        if (position + steps < -10 || position + steps > 10) {
8.           return false;
9.        } else {
10.          position += steps;
11.          return true;
12.       }
13.    }
14. }
```

επιστρέφουμε μια τιμή μέσα στον κώδικα χρησιμοποιώντας την εντολή **return**

- όταν ορίζουμε μια μέθοδο που επιστρέφει τιμή θα πρέπει να ορίσουμε τον τύπο της τιμής που επιστρέφει
  - π.χ. η μέθοδος `moveManySteps` επιστρέφει τιμή `boolean`
- μία μέθοδος μπορεί να επιστρέφει και ένα αντικείμενο μιας κλάσης

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο IV - Υλοποίηση II

```
1.  class Car4
2.  {
3.     private int position = 0;
4.
5.     public boolean moveManySteps(int steps)
6.     {
7.        if (position + steps < -10 || position + steps > 10) {
8.           return false;
9.        }
10.       position += steps;
11.       return true;
12.    }
13. }
```

Αν μπούμε μέσα στο `if` η `return` θα **σταματήσει** την εκτέλεση του κώδικα και θα μας **βγάλει** από τη μέθοδο (επιστρέφοντας την τιμή `false`)

→ άρα, **δε** χρειάζεται πλέον το `else`

Μέθοδος
Τύπος

- μία μέθοδος που επιστρέφει τιμή ορίζεται με συγκεκριμένο τύπο
  - `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, τότε η έκφραση στο `return` πρέπει να επιστρέφει μία τιμή τύπου (συμβατού με το) T
  - π.χ.

```
public double division(int x, int y) {
    return x / (double) y;
}
```

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο IV - Υλοποίηση II

```
1.  import java.util.Scanner;
2.
3.  class Car4 {
4.      private int position = 0;
5.
6.      public boolean moveManySteps(int steps) {
7.          if (position + steps < -10 || position + steps > 10) {
8.              return false;
9.          }
10.         position += steps;
11.         return true;
12.     }
13.
14.     public void printPosition() {
15.         System.out.println("Car at position: " + position);
16.     }
17. }
18.
19. class MovingCar4 {
17.     public static void main(String args[]) {
18.         Scanner input = new Scanner(System.in);
19.         Car4 myCar = new Car4();
20.         int steps = input.nextInt();
21.         boolean carMoved = myCar.moveManySteps(steps);
22.         if (carMoved) {myCar.printPosition();}
23.         else {System.out.println("Car could not be moved");}
24.     }
25. }
```

κλήση της μεθόδου

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο IV - Υλοποίηση II-II

```
1.  import java.util.Scanner;
2.
3.  class Car4 {
4.      private int position = 0;
5.
6.      public boolean moveManySteps(int steps) {
7.          if (position + steps < -10 || position + steps > 10) {
8.              return false;
9.          }
10.         position += steps;
11.         return true;
12.     }
13.
14.     public void printPosition() {
15.         System.out.println("Car at position: " + position);
16.     }
17. }
18.
19. class MovingCar4b {
17.     public static void main(String args[]) {
18.         Scanner input = new Scanner(System.in);
19.         Car4 myCar = new Car4();
20.         int steps = input.nextInt();
21.         if (myCar.moveManySteps(steps)) {myCar.printPosition();}
22.         else {System.out.println("Car could not be moved");}
23.     }
24. }
```

κλήση της μεθόδου και χρήση του αποτελέσματος **απευθείας** μέσα στη συνθήκη → **δε** χρειάζεται να το αποθηκεύσουμε

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο IV - Υλοποίηση II-III

```
1.  import java.util.Scanner;
2.
3.  class Car4 {
4.      private int position = 0;
5.
6.      public boolean moveManySteps(int steps) {
7.          if (position + steps < -10 || position + steps > 10) {
8.              return false;
9.          }
10.         position += steps;
11.         return true;
12.     }
13.
14.     public void printPosition() {
15.         System.out.println("Car at position: " + position);
16.     }
17. }
18.
19. class MovingCar4c {
17.     public static void main(String args[]) {
18.         Scanner input = new Scanner(System.in);
19.         Car4 myCar = new Car4();
20.         int steps = input.nextInt();
21.         myCar.moveManySteps(steps));
22.         myCar.printPosition();
23.     }
24. }
```

η **moveManySteps** επιστρέφει τιμή, αλλά η κλήση της την **αγνοεί**

η **printPosition** θα επιστρέψει **0** αν δεν κινήθηκε το όχημα

Η εντολή return II

- μπορούμε να καλέσουμε τη `return` και σε μία `void` μέθοδο (που δεν επιστρέφει κάποια τιμή)
  - η κλήση γίνεται χωρίς κάποια επιστρεφόμενη τιμή: **`return;`**
  - σταματά την εκτέλεση της μεθόδου

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

Η εντολή return III

- μπορούμε να καλέσουμε τη `return` και σε μία `void` μέθοδο (που δεν επιστρέφει κάποια τιμή)
  - η κλήση γίνεται χωρίς κάποια επιστρεφόμενη τιμή: **`return;`**
  - σταματά 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;}
}
```

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο V - Περιγραφή

- θέλουμε να μπορούμε να κινούμε το όχημα όσες θέσεις θέλουμε είτε προς τα εμπρός (+) είτε προς τα πίσω (-), και να παρουσιάζεται όλη η πορεία κίνησης (και όχι μόνο η τελική θέση)
  - θα τυπώνουμε τη νέα θέση μετά από κάθε βήμα

Υλοποίηση:

- θα ορίσουμε μια βοηθητική μεταβλητή `delta` την οποία θα προσθέτουμε στο `position` σε κάθε βήμα
- η προεπιλεγμένη τιμή της `delta` θα είναι 1
- αν η παράμετρος `steps` είναι αρνητική θα την μετατρέπουμε σε θετική και θα θέτουμε: `delta = -1`

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο V - Υλοποίηση

```
1.  class Car5 {
2.     private int position = 0;
3.
4.     public void moveManySteps(int steps) {
5.        int delta = 1;
6.        if (steps < 0) { steps = -steps; delta = -1;}
7.        for (int i = 0; i < steps; i++) {
8.           position += delta;
9.           System.out.println("Car at position: " + position);
10.       }
11.    }
12.
13.    public void printPosition() {
14.       System.out.println("Car at position: " + position);
15.    }
16. }
17.
27. class MovingCar5 {
28.    public static void main(String args[]) {
29.       Car5 myCar = new Car5();
30.       int steps = -10;
31.       myCar.moveManySteps(steps);
32.       System.out.println("--: " + steps);          // Τυπώνει: --: -10
33.    }
34. }
```

delta
- τοπική μεταβλητή της μεθόδου
- ορίζεται μέσα στη μέθοδο
- υπάρχει μόνο μέσα στην μέθοδο
- στο τέλος της μεθόδου χάνεται

steps
- παράμετρος της μεθόδου
- τοπική μεταβλητή της μεθόδου
- στο τέλος της μεθόδου χάνεται

το πέρασμα παραμέτρων γίνεται **δια τιμής** → επομένως, η τιμή της μεταβλητής του ορίσματος δεν μεταβάλλεται

Η μεταβλητή **steps** στη **main** είναι διαφορετική από την παράμετρο **steps**

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο V - Υλοποίηση II

```
1.  class Car5 {
2.     private int position = 0;
3.
4.     public void moveManySteps(int steps) {
5.        int delta = 1;
6.        if (steps < 0) { steps = -steps; delta = -1;}
7.        for (int i = 0; i < steps; i++) {
8.           position += delta;
9.           printPosition();
10.       }
11.    }
12.
13.    public void printPosition() {
14.       System.out.println("Car at position: " + position);
15.    }
16. }
17.
27. class MovingCar5 {
28.    public static void main(String args[]) {
29.       Car5 myCar = new Car5();
30.       int steps = -10;
31.       myCar.moveManySteps(steps);
32.       System.out.println("--: " + steps);
33.    }
34. }
```

μπορούμε να κάνουμε την εκτύπωση καλώντας την **printPosition()**

✍ κάθε **μέθοδο** που ορίζουμε μέσα σε μία κλάση μπορούμε να τη χρησιμοποιήσουμε και μέσα στην **ίδια** κλάση

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο VI - Περιγραφή

- 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: δε χρειάζεται να φαίνεται έξω από την κλάση

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

    return randomSteps;
}

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

κλήση της συνάρτησης και χρήση της επιστρεφόμενης τιμής

Κλάση & Αντικείμενο

Παράδειγμα: Αυτοκίνητο VI - Υλοποίηση

```
1.  import java.util.Random;
2.
3.  class Car6 {
4.      private int MAX_VALUE = 3;
5.      private int position = 0;
6.      private Random randomGenerator = new Random();
7.
8.      private int computeRandomSteps()
9.      {
10.         int randomSteps = randomGenerator.nextInt(2*MAX_VALUE + 1) - MAX_VALUE;
11.         return randomSteps;
12.     }
13.
14.     public void move()
15.     {
16.         int steps = computeRandomSteps();
17.         moveManySteps(steps);
18.     }
19.
20.     public int moveManySteps(int steps) {...}
21.
22.     public void printPosition() {
23.        System.out.println("Car at position: " + position);
24.     }
25. }
```

κλάση **Random**: μια γεννήτρια τυχαίων αριθμών που παράγει τυχαίους αριθμούς

μέθοδος **nextInt(int x)** της **Random**: Επιστρέφει έναν τυχαίο ακέραιο αριθμό στο διάστημα **[0, x)**

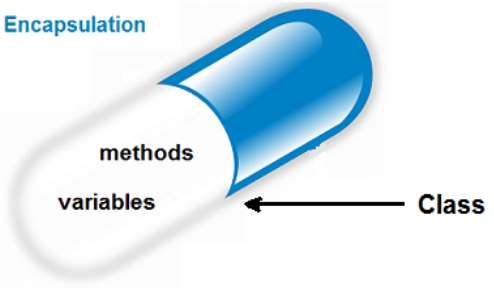
```
26.  class MovingCar6 {
27.      public static void main(String args[]) {
28.          Car6 myCar = new Car6();
29.          myCar.move();
30.      }
31.  }
```

Public/Private

- ό,τι είναι ορισμένο ως `public` σε μία κλάση είναι προσβάσιμο από οποιονδήποτε
  - μπορούμε να καλέσουμε τις μεθόδους (ή να προσπελάσουμε τα πεδία) ορίζοντας ένα αντικείμενο της κλάσης
- ό,τι είναι ορισμένο ως `private` σε μία κλάση είναι προσβάσιμο μόνο από την ίδια κλάση
- ο προσδιοριστής `private` επιτρέπει την απόκρυψη πληροφοριών (information hiding)

  π.χ.
  - ο χρήστης της κλάσης `Car6`, δε 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 `Car6` → κανείς άλλος δε θα πρέπει να επηρεαστεί από την εν λόγω αλλαγή στον κώδικα
- τα πεδία μιας κλάσης τα ορίζουμε πάντα `private`

Ενθυλάκωση (Encapsulation)

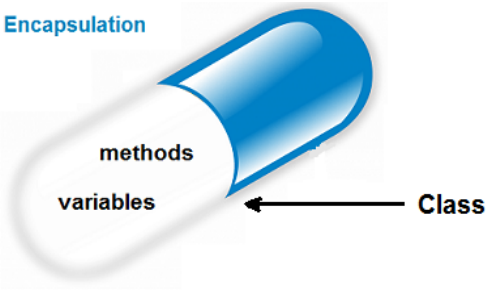


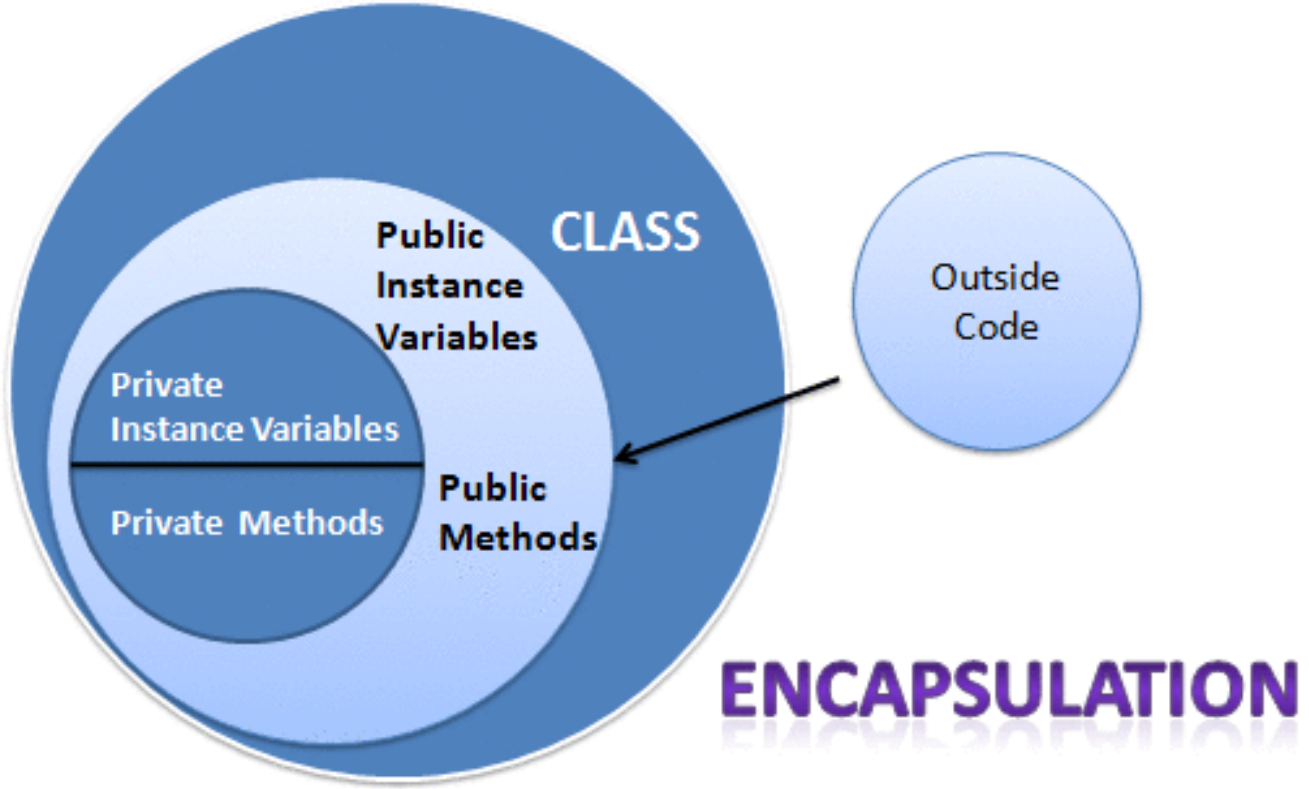
- η ομαδοποίηση δεδομένων και μεθόδων σε μία οντότητα (κλάση και αντικείμενα της κλάσης) ώστε να είναι εύχρηστη
  - μέσω μίας καλά ορισμένης διεπαφής (interface)
  - οι λεπτομέρειες υλοποίησης είναι κρυμμένες από τον χρήστη

- **API** (Application Programming Interface)
  - μία 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 (δε χρειάζεται να ξέρετε την υλοποίηση των μεθόδων της)

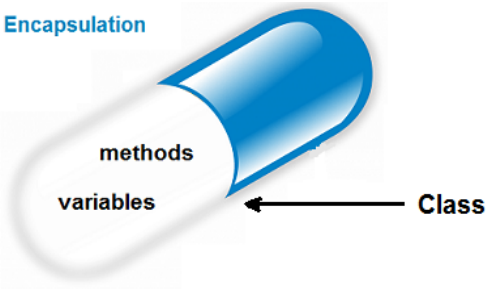
- **ADT** (Abstract Data Type)
  - ένας τύπος δεδομένων που ορίζεται χρησιμοποιώντας την αρχή της ενθυλάκωσης
  - δεδομένα και μέθοδοι

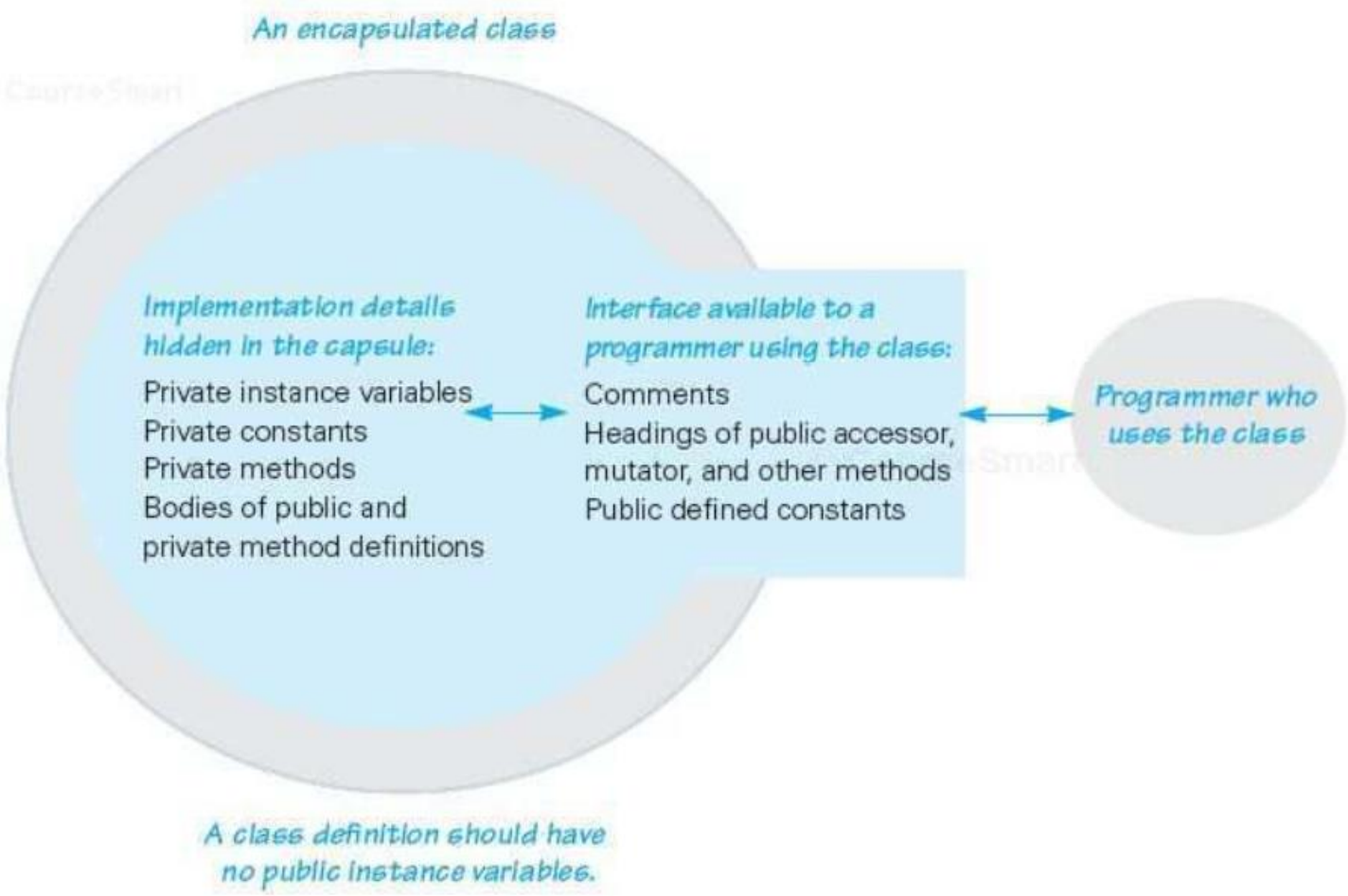
Ενθυλάκωση (Encapsulation) II





Ενθυλάκωση (Encapsulation) III





Ενθυλάκωση

Μέθοδοι ανάγνωσης και τροποποίησης (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
  - ? δεδομένου ότι το πεδίο είναι `private`, πώς θα το κάνουμε αυτό;

☞ ορίζουμε ειδικές μεθόδους

- μέθοδος ανάγνωσης (accessor method) για διάβασμα
- μέθοδος τροποποίησης (mutator method) για γράψιμο

- σύμβαση: στη Java η ονοματολογία των μεθόδων αυτών γίνεται με συγκεκριμένο τρόπο:
  - για την ανάγνωση: `get<όνομα πεδίου>()` (π.χ. `getPosition()`)
  - για την τροποποίηση: `set<όνομα πεδίου>()` (π.χ. `setPosition()`)

Ενθυλάκωση

Μέθοδοι ανάγνωσης και τροποποίησης - 1ο Παράδειγμα

```
1.  class Car7
2.  {
3.     private int position = 0;
4.
5.     public int getPosition() {
6.        return position;
7.     }
8.
9.     public void setPosition(int p) {
10.       position = p;
11.    }
12.
13.    public void move() {
14.       position++;
15.    }
16. }
```

υπάρχουν περιπτώσεις που μπορεί να θέλουμε η συνάρτηση **setPosition** να επιστρέφει **boolean** (**true** αν η ανάθεση έγινε επιτυχώς, **false** αλλιώς)

```
17. class MovingCar7
18. {
19.    public static void main(String args[])
20.    {
21.       Car7 myCar = new Car7();
22.       myCar.setPosition(10);
23.       myCar.move();
24.       System.out.println("Car at position: " + myCar.getPosition());
25.    }
26. }
```

Ενθυλάκωση

Μέθοδοι ανάγνωσης και τροποποίησης - 2ο Παράδειγμα

```
1.    class Car8 {
2.       private int position = 0;
3.
4.       public int getPosition() {
5.           return position;
6.       }
7.
8.       public boolean setPosition(int p) {
9.          if (position < 0) {
10.            return false;
11.         }
12.         position = p;
13.         return true;
14.      }
15.
16.      public void move() {
17.         position++;
18.      }
19.   }
18.
19.  class MovingCar8 {
20.      public static void main(String args[]) {
21.         Car8 myCar = new Car8();
22.         boolean check = myCar.setPosition(10);
23.         if (!check) {
24.            System.out.println("position not set");
25.         }
26.      }
27.   }
```

η **setPosition** μπορεί να επιστρέφει τιμή. Το πιο συνηθισμένο είναι να επιστρέφει **boolean** (για το εάν έγινε σωστά ή όχι η ανάθεση)

Τοπικές μεταβλητές

- οι τοπικές μεταβλητές (και οι παράμετροι) που ορίζουμε μέσα σε μία μέθοδο, έχουν προτεραιότητα σε σχέση με τα πεδία του αντικειμένου της κλάσης

  οπότε,

  - εάν έχουμε μια τοπική μεταβλητή με το ίδιο όνομα με ένα πεδίο μέσα σε μία μέθοδο
    - π.χ. `position`
  - τότε, όταν χρησιμοποιούμε το όνομα αναφερόμαστε στην τοπική μεταβλητή και όχι στο πεδίο

  ☞ αν θέλουμε να αναφερθούμε στο πεδίο μπορούμε να χρησιμοποιήσουμε τη δεσμευμένη λέξη `this`

Το αντικείμενο this

- με τη δεσμευμένη λέξη `this` ένα αντικείμενο αναφέρεται στον εαυτό του

```
1.   class Car
2.   {
3.      private int position = 0;
4.
5.      public void setPositionToTen() {
6.         this.position = 10;
7.      }
8.   }
9.
27.  class Example
28.  {
29.     public static void main(String args[])
30.     {
31.        Car myCar = new Car();
32.        myCar.setPositionToTen();
33.     }
34.  }
```

το **this.position** αναφέρεται στο εσωτερικό πεδίο **position** του αντικείμενουπου θα καλέσει τη μέθοδο

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

Ενθυλάκωση

Μέθοδοι ανάγνωσης και τροποποίησης - 2ο Παράδειγμα II

```
1.    class Car9 {
2.       private int position = 0;
3.
4.       public int getPosition() {
5.            return position;
6.       }
7.
8.       public boolean setPosition(int position) {
9.          if (position < 0) {
10.            return false;
11.         }
12.         this.position = position;
13.         return true;
14.      }
15.
16.      public void move() {
17.         position++;
18.      }
19.   }
18.
19.  class MovingCar9 {
20.      public static void main(String args[]) {
21.         Car9 myCar = new Car9();
22.         myCar.setPosition(10);
23.         myCar.move();
24.         System.out.println("Car at position: " + myCar.getPosition());
25.      }
26.   }
```

το κρυφό πεδίο **this** προσδιορίζει το αντικείμενο που κάλεσε τη μέθοδο

το **this.position** αναφέρεται στο αναφέρεται στο πεδίο του αντικειμένου

το **position** αναφέρεται στην παράμετρο της συνάρτησης

έτσι μπορούμε να χρησιμοποιήσουμε το **ίδιο** όνομα μεταβλητής **χωρίς** να δημιουργείται σύγχυση

Ενθυλάκωση

Μέθοδοι ανάγνωσης και τροποποίησης - 3ο Παράδειγμα

```
1.  class LocalVariableTest {
2.      private int var = 10;                       // ορισμός του πεδίου var
3.
4.      public void method1() {
5.          int var = 5;                            // ορισμός τοπικής μεταβλητής var
6.          var++;                                  // η χρήση της var μέσα στη μέθοδο αναφέρεται στην τοπική μεταβλητή
7.      }
8.
9.      public void method2(int var) {              // ορισμός παραμέτρου var
10.         var++;                                  // η χρήση της var μέσα στη μέθοδο αναφέρεται στην τοπική μεταβλητή
11.     }
12.
13.     public void method3() {
14.         int var = 1;                            // ορισμός τοπικής μεταβλητής var
15.         this.var = var;                         // η χρήση της var μέσα στη μέθοδο αναφέρεται στην τοπική μεταβλητή
16.     }                                           // το this.var αναφέρεται στο πεδίο της κλάσης
17.
18.     public void printVar() {
19.        System.out.println("var = " + var);
20.     }
21.
22.     public static void main(String args[]) {
18.         LocalVariableTest x = new LocalVariableTest();
19.         x.method1();        x.printVar();               // τυπώνει: var = 10
20.         x.method2(20);      x.printVar();               // τυπώνει: var = 10
21.         x.method3();        x.printVar();               // τυπώνει: var = 1
22.     }
23. }
```

Μπορούμε να ορίσουμε `main` μέσα σε μία κλάση για να τη δοκιμάσουμε

Σύνοψη

- Ορισμός κλάσεων και αντικειμένων
- Πεδία, μέθοδοι και παράμετροι
  - public / private
- Τοπικές μεταβλητές
- Πέρασμα παραμέτρων δια τιμής
- Μέθοδοι που επιστρέφουν τιμές
  - return
- Ενθυλάκωση
  - API - Application Programming Interface
  - ADT - Abstract Data Type
  - αντικείμενο this