Αντικειμενοστρεφής Προγραμματισμός
Διάλεξη – 5 : ΠΕΡΙΣΣΟΤΕΡΑ ΓΙΑ ΤΙΣ CLASSES

Κων. Κόκκινος

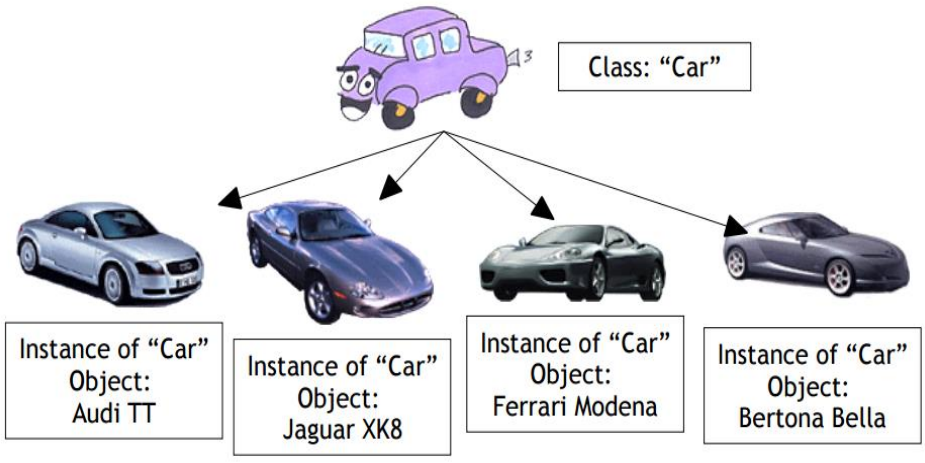
Αντικειμενοστραφής Προγραμματισμός-1/2

- Η αντικειμενοστραφής πλευρά της Java φαίνεται με την χρήση των αντικειμένων και των κλάσεων στις οποίες ανήκουν τα αντικείμενα.
- Όλα τα αντικείμενα που έχουν κοινά χαρακτηριστικά ανήκουν στην ίδια κλάση και κάθε ένα από αυτά λέγεται ότι είναι "στιγμιότυπο" (instance) της κλάσης.
- Για παράδειγμα, η κλάση "αυτοκίνητο" θεωρεί μια γενική εικόνα του αυτοκινήτου με κάποια χαρακτηριστικά που υπάρχουν σε όλα τα στιγμιότυπα.

Αντικειμενοστραφής Προγραμματισμός-2/2

- Συγκεκριμένα, η κλάση αυτοκίνητο πιθανώς να ορίζει ότι είναι τετράτροχο, έχει τιμόνι, ταχύτητες, πεντάλ και καθίσματα για τους οδηγούς αλλά δεν ορίζει με σαφήνεια το σχήμα ή τους μηχανισμούς όλων των εξαρτημάτων, ούτε τα χαρακτηριστικά της μηχανής (κυβικά, ίπποι, κύλινδροι, κλπ).
- Αυτά είναι χαρακτηριστικά που αφορούν το κάθε αντικείμενο ξεχωριστά (ή κάποια υποκατηγορία/υποκλάση της κλάσης "αυτοκίνητο").
- Για καλύτερη κατανόηση, δείτε το παρακάτω σχήμα.

Παράδειγμα κλάσης 1/5



Παράδειγμα κλάσης **2/5**

- Στη Java η κλάση δηλώνεται με τη λέξη class. Στο παράδειγμα του αυτοκινήτου, έστω ότι θέλουμε να δηλώσουμε μια κλάση με το όνομα Car:
- **class Car {**

  (δεδομένα/μεταβλητές)

  (συναρτήσεις/μέθοδοι)

  }

Παράδειγμα κλάσης ΙΔΙΟΤΗΤΕΣ **3/5**

- Τα χαρακτηριστικά του αντικειμένου είναι τα δεδομένα γύρω από τα οποία πρέπει να αναπτύξουμε το πρόγραμμά μας.
- Στο παράδειγμα με το αυτοκίνητο, τα δεδομένα αυτά είναι:
  - η γωνία στρέψης του τιμονιού,
  - η πίεση που ασκούμε στα πεντάλ,
  - η θέση του μοχλού ταχυτήτων και άλλα.
- Στον προγραμματισμό, αυτά τα δεδομένα θα πρέπει να μοντελοποιηθούν, δηλαδή να γίνει η αντιστοιχία τους σε μεταβλητές τις οποίες μπορούμε να επεξεργαστούμε στο πρόγραμμά μας.

Παράδειγμα κλάσης ΙΔΙΟΤΗΤΕΣ **4/5**

```
class Car
{
   // Γωνία στρέψης του τιμονιού
   float steering_angle;

   // Ποσοστό πατήματος του γκαζιού (0 = καθόλου, 100 = τερματισμένο!)
   float gas_pedal;

   // Ποσοστό πατήματος του φρένου (0 = καθόλου, 100 = τερματισμένο!)
   float break_pedal;

   // Ποσοστό πατήματος του συμπλέκτη (0 = καθόλου,
   // 100 = τερματισμένο!)
   float clutch;

   // θέση της τρέχουσας ταχύτητα (πιθανές τιμές: 0, 1,2,3,4,5,
   // 0 = νεκρό, -1 = όπισθεν)
   int gear;

   // μεταβλητές που καθορίζουν την επιτάχυνση, την ταχύτητα του
   // αυτοκινήτου και τις στροφές του κινητήρα
   float acceleration, speed, rpm;
}
```

Παράδειγμα κλάσης ΙΔΙΟΤΗΤΕΣ **5/5**

- Εδώ πρέπει να σημειωθεί ότι οι μεταβλητές της κλάσης έχουν διαφορετικές τιμές για κάθε αντικείμενο, και κάθε αντικείμενο έχει πρόσβαση μόνο στις δικές του μεταβλητές.
- Επίσης, ο τύπος δεδομένων που επιλέχθηκε για κάθε μεταβλητή εξαρτάται από το είδος της πληροφορίας που θα κρατάει αυτή η μεταβλητή. Για παράδειγμα, εφόσον η γωνία στρέψης του τιμονιού θα είναι σε μοίρες, είναι λογικό να χρησιμοποιήσουμε ένα τύπο που θα μπορεί να κρατήσει δεκαδικούς αριθμούς, όπως ο float.
- Αυτά, λοιπόν, είναι τα δεδομένα μας ομαδοποιημένα υπό την έννοια της κλάσης "Car". Αυτή τη στιγμή δεν είναι τίποτε παραπάνω από μια ομάδα μεταβλητών χωρίς να έχουμε ορίσει τον τρόπο επεξεργασίας τους. Γι' αυτό είναι απαραίτητη η ύπαρξη του κατάλληλου interface των δεδομένων για τον χρήστη. Το interface επιτυγχάνεται με την ύπαρξη των μεθόδων της κλάσης (member methods).

ΜΕΘΟΔΟΙ 1/3

- Οι μέθοδοι (methods) σε μια κλάση, δεν είναι τίποτε παραπάνω από συναρτήσεις (υπορουτίνες) που προσφέρουν πρόσβαση στα δεδομένα του εκάστοτε αντικειμένου της κλάσης.
- Η αλληλεπίδραση του χρήστη με κάθε αντικείμενο γίνεται μέσω των μεθόδων της κλάσης, οι οποίες καθορίζουν και τον τρόπο χειρισμού των μεταβλητών του αντικειμένου.
- Στο παράδειγμά μας, οι μέθοδοι θα καθορίζουν με ποιον τρόπο θα στρίβουμε το τιμόνι, θα αυξάνουμε το πάτημα στο γκάζι ή στο φρένο, θα αλλάζουμε ταχύτητες αλλά και πώς θα μπορούμε να γνωρίζουμε την ταχύτητα του αυτοκινήτου, τις στροφές του κινητήρα δηλαδή πληροφορίες που δε μπορούμε να ελέγξουμε άμεσα.

ΜΕΘΟΔΟΙ 2/3

- Οι πιθανές μέθοδοι θα μπορούσαν να ήταν οι εξής:

```
// Αλλαγή της γωνία στρέψης του τιμονιού, <relative_angle> μοίρες
// σε σχέση με την τρέχουσα γωνία.
void turn_wheel(float relative_angle);

// Πάτημα πεντάλ γκαζιού
void press_gas_pedal(float amount);

// Πάτημα πεντάλ φρένου
void press_break_pedal(float amount);

// Πάτημα πεντάλ συμπλέκτη
void press_clutch_pedal(float amount);

// Αλλαγή της ταχύτητας. Επιστρέφει true αν η αλλαγή ήταν επιτυχής
// ή false αν ήταν ανεπιτυχής (π.χ. από 5 σε όπισθεν).
bool change_gear(int new_gear);

// προβολή της τρέχουσας ταχύτητας, επιτάχυνσης και στροφών του
// κινητήρα
float get_acceleration();
float get_speed();
float get_rpm();
```

ΜΕΘΟΔΟΙ 3/3

- Οι παραπάνω είναι απλώς οι δηλώσεις των μεθόδων, δηλαδή δεν περιέχουν καθόλου κώδικα. Για να είναι ολοκληρωμένος ο ορισμός μιας μεθόδου θα πρέπει να συνοδεύεται και από την υλοποίησή της (implementation). Η υλοποίηση μπορεί να γίνει ταυτόχρονα με τη δήλωση, για παράδειγμα η υλοποίηση της turn_wheel() θα μπορούσε να είναι:

```
void turn_wheel(float relative_angle)
{
   steering_angle += relative_angle;
   if (steering_angle <= -720.0)
       steering_angle = -720.0;
   if (steering_angle >= 720.0)
       steering_angle = 720.0;
}
```

Δημιουργία αντικειμένων με τη new 1/3

- Έχοντας ορίσει την κλάση μας, θα πρέπει να δημιουργήσουμε τα αντικείμενα -τα στιγμιότυπα (instances) της κλάσης. Για το σκοπό αυτό, χρησιμοποιούμε την εντολή new της Java.
- Η new δημιουργεί μια φυσική αναπαράσταση της κλάσης, ένα μοναδικό στιγμιότυπο, και επιστρέφει ένα δείκτη αναφοράς (reference) σε αυτό. Με αυτό το δείκτη αναφοράς μπορούμε να προσπελάσουμε το αντικείμενο με οποίον τρόπο θέλουμε (και εφόσον το επιτρέπει το ίδιο το αντικείμενο).
- Ο τρόπος χρήσης της new είναι ο εξής:

```
Car acar = new Car();
Car anothercar = new Car();
```

Δημιουργία αντικειμένων με τη new 2/3

- Οι μεταβλητές acar και anothercar είναι οι δείκτες αναφοράς στα αντίστοιχα αντικείμενα που δημιουργούνται με τη new.
- Η δημιουργία του αντικειμένου δεν είναι αναγκαίο να γίνει κατά την δήλωσή του. Το ίδιο αποτέλεσμα μπορούμε να έχουμε και με τις εντολές:

  Car acar;

  acar = new Car();
- Τα δεδομένα του κάθε αντικειμένου (οι μεταβλητές) αλλά και οι μέθοδοι της κλάσης που αντιστοιχούν στο αντικείμενο, μπορούν να προσπελλαστούν ως εξής:

Δημιουργία αντικειμένων με τη new 3/3

```
// Η γωνία στρέψης του τιμονιού του acar
acar.steering_angle

// Η γωνία στρέψης του τιμονιού του anothercar
anothercar.steering_angle

// Εντολή στο acar να στρίψει δεξιά 13.4 μοίρες.
acar.turn(13.4);

// Επιστρέφει την τρέχουσα ταχύτητα του acar
float speed = acar.get_speed();

// Εντολή στο anothercar να στρίψει αριστερά 32 μοίρες
anothercar.turn(-32.0);

// Εντολή στο anothercar να βάλει όπισθεν
bool result = anothercar.ghange_gear(-1);
```

- Όπως βλέπουμε ορισμένοι μέθοδοι δέχονται κάποιες παραμέτρους (εντός παρενθέσεων). Οι παράμετροι μπορεί να είναι μεταβλητές οποιουδήποτε αποδεκτού τύπου στη Java ή ακόμη και άλλα αντικείμενα. Αν έχουμε περισσότερες από μία παραμέτρους τις διαχωρίζουμε με κόμμα ','.

Κατασκευαστές (Constructors) 1/5

- Όταν δημιουργείται ένα αντικείμενο με την εντολή new, στην πραγματικότητα η Java, αφού δεσμεύσει την απαραίτητη μνήμη για το αντικείμενο, εκτελεί μια συγκεκριμένη μέθοδο της κλάσης, τον κατασκευαστή/δημιουργό (constructor).
- Ο κατασκευαστής πραγματοποιεί τις απαραίτητες ενέργειες για να καταστεί κάποιο αντικείμενο έτοιμο προς χρήση. Αυτές μπορεί να είναι κάτι απλό όπως η ρύθμιση κάποιων μεταβλητών με αρχικές τιμές, ή πιο περίπλοκο όπως η δημιουργία σύνδεσης με μια βάση δεδομένων, η αρχικοποίηση των πινάκων SQL, η δέσμευση κάποιων δικτυακών θυρών (sockets) για κάποιο server ή ακόμη και το άνοιγμα κάποιου παραθύρου για εμφάνιση γραφικής πληροφορίας.
- Ο κατασκευαστής έχει τη μορφή μιας μεθόδου με το όνομα της κλάσης και χωρίς τύπο (δηλαδή δεν δηλώνεται ο τύπος δεδομένων που επιστρέφει, το οποίο είναι διαφορετικό από το να δηλωθεί ως void).

Κατασκευαστές (Constructors) 2/5

- Για παράδειγμα, στην κλάση "Car", ένας πιθανός κατασκευαστής μπορεί να είναι:

```
Car()
{
   steering_wheel = 0.0;
   gas_pedal = 0.0;
   break_pedal = 0.0;
   float clutch = 0.0;
   int gear = 0;
   acceleration = 0.0;
   speed = 0.0;
   rpm = 0.0;
}
```

Δηλαδή, ο κατασκευαστής πραγματοποιεί την αρχικοποίηση (initialization) των μεταβλητών του αντικειμένου ώστε αυτό να είναι έτοιμο προς χρήση. Μπορούμε να έχουμε περισσότερους από έναν κατασκευαστές, οι οποίοι να δέχονται διαφορετικές παραμέτρους ο καθένας.

Κατασκευαστές (Constructors) 3/5

- Για παράδειγμα, αν υποθέσουμε ότι μπορούσαμε να ορίσουμε χαρακτηριστικά του κινητήρα (engine_cc: κυβικά, engine_hp: ίπποι) στη δημιουργία του αντικειμένου, θα μπορούσαμε να έχουμε έναν επιπλέον κατασκευαστή:

```
Car(int cc, int hp)
{
   engine_cc = cc;
   engine_hp = hp;
   // Ακολουθούν οι υπόλοιπες εντολές αρχικοποίησης του αντικειμένου
}
```

Η δημιουργία περισσότερων από έναν δημιουργούς καλείται constructor overloading και είναι υποπερίπτωση του χαρακτηριστικού της Java, method overloading (

Κατασκευαστές (Constructors) 4/5

- 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.

Κατασκευαστές (Constructors) 5/5

```
class Person
{
    private String name;

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

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

public class HelloWorld
{
    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

```
class Date
{
  private int day;
  private int month;
  private int year;
  private String[] monthNames =
        {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec"};

  public Date(int day, int month, int year)
  {
    if (day <= 0 || day > 31 || month <= 0 || month >12 ){
        return;
    }
    this.day = day;
    this.month = month;
    this.year = year;
  }

  public void printDate()
  {
    System.out.println(day + " " + monthNames[month-1] + " " +
year);
  }
}
class DateExample
{
  public static void main(String args[])
  {
    Date myDate = new Date(7,3,2013);
    myDate.printDate();
  }
}
```

Υπερφόρτωση - Προσοχή

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

Destructors- Garbage collection στη Java

- Όσοι γνωρίζουν C++, θα θυμούνται πιθανόν ότι παράλληλα με τους κατασκευαστές ενός αντικειμένου υπάρχουν και οι καταστροφείς του (destructors).
- Στη C++, ό,τι αντικείμενο δημιουργούμε πρέπει να το καταστρέφουμε (συνήθως με την εντολή delete) ελευθερώνοντας όλους τους πόρους που δεσμεύσαμε κατά την ύπαρξή του (κλείσιμο αρχείων, αποσύνδεση από βάσεις δεδομένων, τερματισμός threads, αποδέσμευση μνήμης, κλπ).
- Στη Java κάτι τέτοιο δεν είναι απαραίτητο, για την ακρίβεια δεν παρέχεται καν αυτή η δυνατότητα! Η Java παρέχει ένα δικό της μηχανισμό "περισυλλογής σκουπιδιών" (garbage collection) όπως λέγεται, με τον οποίο ελευθερώνει τα αντικείμενα που δεν χρησιμοποιούνται πλέον από το πρόγραμμα. Έτσι πλέον, δεν χρειάζεται να καταστρέψουμε πλέον αντικείμενα

Ο δείκτης αναφοράς **this 1/2**

- Μέσα σε μια μέθοδο, μπορούμε να χρησιμοποιήσουμε μια μεταβλητή της κλάσης απλώς με το όνομά της, αναφερόμενοι φυσικά στην τιμή που έχει η μεταβλητή για το συγκεκριμένο αντικείμενο.
- Τις περισσότερες φορές κάτι τέτοιο είναι αρκετό, υπάρχουν όμως και περιπτώσεις που χρειάζεται πιο ρητή αναφορά στο συγκεκριμένο αντικείμενο.
- Για το σκοπό αυτό μπορούμε να χρησιμοποιήσουμε τη μεταβλητή this που επιστρέφει πάντα ένα δείκτη αναφοράς στο τρέχον αντικείμενο (δηλαδή αυτό που καλεί την μέθοδο).
- Με το δείκτη αναφοράς this, η μέθοδος **turn_wheel()** που είδαμε παραπάνω μετασχηματίζεται ως εξής:

Ο δείκτης αναφοράς **this 2/2**

```
void turn_wheel(float relative_angle)
{
   steering_angle += relative_angle;
   if (steering_angle <= -720.0)
       steering_angle = -720.0;
   if (steering_angle >= 720.0)
       steering_angle = 720.0;
}

void turn_wheel(float relative_angle)
{
   this.steering_angle += relative_angle;
   if (this.steering_angle <= -720.0)
       this.steering_angle = -720.0;
   if (this.steering_angle >= 720.0)
       this.steering_angle = 720.0;
}
```

- Ο δείκτης αναφοράς **this** είναι ιδιαίτερα χρήσιμος ειδικά σε περιπτώσεις που μεταχειριζόμαστε περισσότερα από ένα όμοια αντικείμενα στην ίδια μέθοδο, για αποφυγή σύγχυσης.

Δυο ειδικές μέθοδοι

- Η Java «περιμένει» να δει τις εξής δύο μεθόδους για κάθε αντικείμενο
  - Τη μέθοδος **toString** η οποία για ένα αντικείμενο επιστρέφει μία string αναπαράσταση του αντικειμένου.
  - Τη μέθοδο **equals** η οποία ελέγχει για ισότητα δύο αντικειμένων
- Και οι δύο συναρτήσεις ορίζονται από τον προγραμματιστή
  - Το τι String θα επιστραφεί και τι σημαίνει δύο αντικείμενα να είναι ίσα μπορούν να οριστούν όπως μας βολεύει.

Παράδειγμα

- Στην κλάση Car θέλουμε να προσθεσουμε τις μεθόδους toString και equals
  - Η toString θα επιστρέφει ένα String με τη θέση του αυτοκινήτου
  - Η equals θα ελέγχει αν δύο οχήματα έχουν την ίδια θέση.

toString()

```
class Car {
  private int position = 0;
  public Car(int position){
    this.position = position;
  }
  public void move(int delta){
    position += delta ;
  }
  public string toString(){
    string s="";
    return s + position;
  }
}
class MovingCarToString
{
  public static void main(String args[]){
    Car myCar1 = new Car(1);
    Car myCar2 = new Car(0); myCar2.move(2);
    System.out.println("Car 1 is at " + myCar1 + " and car 2 is at " + myCar2);
  }
}
```

Χρησιμοποιούμε τις myCar1,myCar2 σαν String. Καλείται η μέθοδος toString() αυτόματα

Ισοδύναμο με το:

```
System.out.println("Car 1 is at " + myCar1.toString() + " and car 2 is at " + myCar2.toString());
```

toString()

```
class Car {
  private int position = 0;
  public Car(int position){
    this.position = position;
  }
  public void move(int delta){
    position += delta ;
  }
  public string toString(){
    return ""+position;
  }
}
class MovingCarToString
{
  public static void main(String args[]){
    Car myCar1 = new Car(1);
    Car myCar2 = new Car(0); myCar2.move(2);
    System.out.println("Car 1 is at " + myCar1 + " and car 2 is at " +
myCar2);
  }
}
```

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

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

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

  public boolean equals(Car other){
    if (this.position == other.position){
      return true;
    }
    return false;
  }
}

class MovingCarEquals
{
  public static void main(String args[]){
    Car myCar1 = new Car(2);
    Car myCar2 = new Car(0); myCar2.move(2);
    if (myCar1.equals(myCar2)){
       System.out.println("Collision!");
    }
  }
}
```

Η Java περιμένει αυτό το συντακτικό για τον ορισμό της equals

Ένα παράδειγμα αντικειμένου ως παράμετρος συνάρτησης

Αν και το πεδίο position είναι private μπορούμε να το προσπελάσουμε γιατί είμαστε μέσα στην κλάση Car.
Μία κλάση μπορεί να προσπελάσει τα ιδιωτικά μέλη όλων των αντικειμένων της κλάσης

Κλήση της equals στο πρόγραμμα

Το αντικείμενο scanner 1/2

- Υπάρχουν δύο κλασικοί τρόποι εισαγωγής δεδομένων (input) σε ένα πρόγραμμα Java: είτε με την χρήση εντολών από το command prompt, είτε με την ευκολία που προσφέρει το γραφικό περιβάλλον της Java με την βοήθεια των βιβλιοθηκών AWT και Swing.
- Η Scanner κλάση χρησιμοποιείται για να διαβάσει όλα τα είδη των primitive types (που ήδη έχουμε περιγράψει σε προηγούμενα μαθήματα) και Strings σαν αποδεκτές τιμές για την εισαγωγή πληροφοριών από τον χρήστη.
- Ο τρόπος με τον οποίο έχουμε την δυνατότητα να διαβάσουμε και να αναγνωρίσουμε το input είναι εύκολος αρκεί να γνωρίζουμε την σωστή λειτουργία και απαιτήσεις της Scanner κλάσης.
- ΠΑΡΑΔΕΙΓΜΑ

```
import java.util.Scanner;
Scanner sc = new Scanner(System.in);
int x = sc.nextInt();
```

Το αντικείμενο scanner 2/2

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

        Scanner in = new Scanner(System.in);

        String best_name = "";
        double best_price = 1;
        int best_score = 0;

        boolean more = true;
        while(more) {

                String next_name;
                double next_price;
                int next_score;

                System.out.println("Please enter the product name: ");
                next_name = in.nextLine();
                System.out.println("Please enter the product price: ");
                next_price = in.nextDouble();
                System.out.println("Please enter the product score: ");
                next_score = in.nextInt();
```

Ορατότητα (Visibility) 1/4

- Όπως ισχύει σε επίπεδο ορισμού κλάσης, έτσι και σε επίπεδο μελών με τον όρο ορατότητα αναφερόμαστε στη διαδικασία που ορίζει ποια μέλη της κλάσης θα είναι προσβάσιμα από άλλα σημεία του κώδικα και ποια όχι.
- Η Java υποστηρίζει τέσσερα διαφορετικά επίπεδα ορατότητας, το **public**, το **protected**, το ***default*** και το **private**.
- Χρησιμοποιώντας τη δεσμευμένη λέξη **public** μπροστά από κάποιο μέλος μιας κλάσης, του δίνουμε επίπεδο ορατότητας public που σημαίνει πως θα είναι ορατό από όλες τις κλάσεις και όλα τα πακέτα. Με την έκφραση «είναι ορατό» εννοούμε πως ο συγκεκριμένος κώδικας είναι προσβάσιμος (μπορεί να χρησιμοποιηθεί) από κάποιο άλλο σημείο. Το συγκεκριμένο επίπεδο ορατότητας είναι και το ασθενέστερο, δηλαδή προσφέρει τη χαλαρότερη 'προστασία'. Όπως θα δούμε αργότερα, σε κάθε κλάση δηλώνουμε με **public** προσδιοριστή ορατότητας τις μεθόδους της.

Ορατότητα (Visibility) 2/4

- Στον αντίποδα, ο προσδιοριστής **private** προσφέρει το υψηλότερο επίπεδο προστασίας. Τα μέλη μιας κλάσης που έχουν δηλωθεί ως **private** είναι ορατά μόνο από κώδικα που βρίσκεται στην ίδια την κλάση. Σύμφωνα με τους κανόνες σωστής πρακτικής, δηλώνουμε ως **private** τις μεταβλητές μέλη μιας κλάσης.
- Ένα ενδιάμεσο επίπεδο προστασίας είναι το protected. Το συγκεκριμένο επίπεδο επιτρέπει σε μέλη μιας κλάσης να είναι προσβάσιμα από την ίδια την κλάση καθώς και από όλες τις υποκλάσεις της, είτε βρίσκονται στο ίδιο είτε σε άλλο πακέτο. Τη συγκεκριμένη δυνατότητα θα την εξετάσουμε εκτενέστερα στην επόμενη ενότητα μιλώντας για την κληρονομικότητα.

Ορατότητα (Visibility) 3/4

- Τέλος, αν δε χρησιμοποιήσουμε κανέναν προσδιοριστή μπροστά από τη δήλωση ενός μέλους μιας κλάσης, η Java θα του αναθέσει αυτόματα το ***default*** επίπεδο ορατότητας. Το συγκεκριμένο επίπεδο μοιάζει με το **protected** μιας και το κάνει προσβάσιμο στην ίδια την κλάση και στις υποκλάσεις της που βρίσκονται στο ίδιο πακέτο με αυτήν. Η διαφορά τους είναι πως το *default* επίπεδο δεν επιτρέπει την πρόσβαση σε υποκλάσεις που βρίσκονται σε άλλο πακέτο.

Ορατότητα (Visibility) 4/4

- Στον παρακάτω πίνακα συνοψίζονται τα επίπεδα ορατότητας της Java και οι δυνατότητες πρόσβασης που παρέχουν

Ορατότητα	public	protected	*default*	private
Από την ίδια κλάση	Ναι	Ναι	Ναι	Ναι
Από άλλη κλάση στο ίδιο πακέτο	Ναι	Ναι	Ναι	Όχι
Από οποιαδήποτε μη υποκλάση σε άλλο πακέτο	Ναι	Όχι	Όχι	Όχι
Από υποκλάση στο ίδιο πακέτο	Ναι	Ναι	Ναι	Όχι
Από υποκλάση σε άλλο πακέτο	Ναι	Ναι	Όχι	Όχι

Getters/Setters 1/4

- Στην κατηγορία διαχειριστικών μεθόδων ανήκουν και οι getters/setters (ονομάζονται επίσης και accessors/mutators) που θα εξετάσουμε στην παρούσα υποενότητα. Πρόκειται για μεθόδους που παρέχουν εξίσου σημαντικές λειτουργίες, αφού μέσω αυτών μας παρέχεται πρόσβαση στις τιμές των μεταβλητών μελών της κλάσης. Θυμηθείτε πως τις μεταβλητές μέλη τις δηλώνουμε στις κλάσεις με προσδιοριστή ορατότητας **private** και άρα χρειαζόμαστε έναν τρόπο που θα μας επιτρέπει είτε να διαβάζουμε είτε να θέτουμε την τιμή των μεταβλητών μελών της κλάσης.
- Αυτό επιτυγχάνεται μέσω των μεθόδων getters/setters, οι οποίες κάνουν ακριβώς αυτό. Για κάθε μεταβλητή μέλος μιας κλάσης ορίζουμε ένα ζεύγος από μεθόδους, από τις οποίες η μία επιστρέφει την τιμή της μεταβλητής μέλους ενώ η άλλη την θέτει. Σύμφωνα με τους κανόνες σωστής πρακτικής, οι μέθοδοι αυτές ονομάζονται σύμφωνα με το ακόλουθο πρότυπο:

```
void setVariable(var_type a) // τυπική setter
var_type getVariable() // τυπική getter
```

Getters/Setters 2/4

- Για παράδειγμα, αν είχαμε μια μεταβλητή μέλος με το όνομα **radious,** τύπου **int,** θα ορίζαμε τις **getters/setters** ως εξής:

```
void setRadius(int rad){
    radius = rad;
}

int getRadius(){
    return radius;
}
```

Getters/Setters 3/4

- Οι μέθοδοι αυτές παρέχουν το μοναδικό τρόπο πρόσβασης των μεταβλητών μελών του αντικειμένου, δεδομένου του ότι δηλώνονται πάντοτε με προσδιοριστή ορατότητας **private**. Αντίστοιχα, οι getters/setters δηλώνονται πάντοτε public, ώστε να μπορεί ο χρήστης της κλάσης να τις χρησιμοποιήσει και να αποκτήσει πρόσβαση στις μεταβλητές μέλη.
- Όπως οι constructors έτσι και οι setters, δεδομένου πως θέτουν την τιμή μιας μεταβλητής μέλους περιέχουν συνήθως κώδικα ελέγχου εγκυρότητας των τιμών πριν τις αναθέσουν στις μεταβλητές μέλη. Λόγω του ότι οι συγκεκριμένες μέθοδοι είναι απαραίτητες σε κάθε κλάση που έχει γραφτεί σύμφωνα με τις αρχές του σωστού αντικειμενοστρεφούς προγραμματισμού καθώς και της ύπαρξης κοινού προτύπου ονομασίας τους, τα σύγχρονα IDEs όπως το Eclipse και το NetBeans παρέχουν τη δυνατότητα αυτόματης δημιουργίας τους (σε απλή βέβαια μορφή που δεν περιέχει ελέγχους εγκυρότητας

Getters/Setters 4/4

```
public class Point {
    // member variables
    private int x; // x coord
    private int y; // y coord
    // methods
    // default constructor
    public Point() { }
    // initialization constructor with ints
    public Point(int p_x, int p_y){
        x = p_x;
        y = p_y;
    }
    // constructor that creates a Point
    // from another Point (copy)
    public Point(Point p){
        x = p.getX();
        y = p.getY();
    }
```

```
// getters/setters
public void setX(int p_x){
    x = p_x;
}
public int getX(){
    return x;
}
public void setY(int p_y){
    y = p_y;
}
public int getY(){
    return y;
}
}
```