ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Constructors

Υπερφόρτωση

Αντικείμενα ως παράμετροι

Ενθυλάκωση

- Η ομαδοποίηση λογισμικού και δεδομένων σε μία οντότητα (κλάση και αντικείμενα της κλάσης) ώστε να είναι εύχρηστη μέσω ενός καλά ορισμένου interface, ενώ οι λεπτομέρειες υλοποίησης είναι κρυμμένες από τον χρήστη.
- API (Application Programming Interface)[Έι-Πι-Άι]
  - Μια περιγραφή για το πώς χρησιμοποιείται η κλάση μέσω των public μεθόδων της.
    - Java docs είναι ένα παράδειγμα.
  - Το API είναι αρκετό για να χρησιμοποιήσετε μια κλάση, δεν χρειάζεται να ξέρετε την υλοποίηση των μεθόδων.

Accessor and Mutator methods

- Πολλές φορές χρειαζόμαστε να διαβάσουμε ή να αλλάξουμε ένα πεδίο ενός αντικειμένου
  - Π.χ., να διαβάσουμε τη θέση του οχήματος, ή να τοποθετήσουμε το όχημα σε μια συγκεκριμένη θέση.
  - Πως θα το κάνουμε αφού τα πεδία είναι private?
- Ορίζουμε ειδικές μεθόδους
  - Μέθοδος προσπέλασης (accessor method) για διάβασμα
  - Μέθοδος μεταλλαγής (mutator method) για γράψιμο
- Σύμβαση: Στη Java η ονοματολογία των μεθόδων αυτών γίνεται με συγκεκριμένο τρόπο:
  - get<ονομα μεταβλητης> για την πρόσβαση
    - getPosition
  - set<ονομα μεταβλητης> για την μετάλλαξη
    - setPosition

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

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

  public int getPosition(){
    return position;
  }

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

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

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

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

        public int getPosition(){
                return position;
        }

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

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

Constructors (Δημιουργοί)

- Όταν δημιουργούμε ένα αντικείμενο συχνά θέλουμε να μπορούμε να το αρχικοποιήσουμε με κάποιες τιμές
  - Ένα Person να αρχικοποιείται με ένα όνομα
  - Ένα Car να αρχικοποιείται με μία θέση
- Μπορούμε να το κάνουμε με μία συνάρτηση set αυτό, αλλά
  - Μπορεί να έχουμε πολλές μεταβλητές να αρχικοποιήσουμε
  - Θέλουμε η αρχικοποίηση να είναι μέρος της δημιουργίας του αντικειμένου
- Την αρχικοποίηση μπορούμε να την κάνουμε με ένα Constructor (Δημιουργό)

Constructors (Δημιουργοί)

- Ο Constructor είναι μια «μέθοδος» η οποία καλείται όταν δημιουργούμε το αντικείμενο χρησιμοποιώντας την new.
- Αν δεν έχουμε ορίσει Constructor καλείται ένας default Constructor χωρίς ορίσματα που δεν κάνει τίποτα.
- Αν ορίσουμε constructor, τότε καλείται ο constructor που ορίσαμε.

Παράδειγμα

```
class Person
{
    private String name;

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

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

public class HelloWorld2
{
    public static void main(String[] args){
        Person alice = new Person("Alice");
        alice.speak("Hello World");
    }
}
```

Constructor: μια μέθοδος με το ίδιο όνομα όπως και η κλάση και χωρίς τύπο (ούτε void)

Αρχικοποιεί την μεταβλητή name

Constructor: καλείται όταν δημιουργείται το αντικείμενο με την new και μόνο τότε

Μια συνομιλία

```
class Person
{
    private String name;

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

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

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

Παράδειγμα

```
class Car
{
        private int position;

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

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

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

}

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

Παράδειγμα

```
class Car
{
   private int position=0;
   private int ACCELERATOR = 2;

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

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

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

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

Η εκτέλεση αυτών των αρχικοποιήσεων γίνεται πριν εκτελεστούν οι εντολές στον constructor

Η τελική τιμή του position θα είναι αυτή που δίνεται σαν όρισμα

Υπερφόρτωση

- Είδαμε μια περίπτωση που είχαμε μια συνάρτηση move η οποία μετακινεί το όχημα κατά μία θέση, και μια συνάρτηση moveManySteps η οποία το μετακινεί όσες θέσεις ορίζει το όρισμα.
  - Το να θυμόμαστε δυο ονόματα είναι μπερδεμένο, θα ήταν καλύτερο να είχαμε μόνο ένα. Και στις δύο περιπτώσεις η λειτουργία που θέλουμε να κάνουμε είναι move
- Η Java μας δίνει αυτή τη δυνατότητα μέσω της διαδικασίας της υπερφόρτωσης (overloading)
  - Ορισμός πολλών μεθόδων με το ίδιο όνομα αλλά διαφορετικά ορίσματα, μέσα στην ίδια κλάση

```
class Car
{
        private int position;

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

        public void move(){
                position ++ ;
        }

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

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

Υπερφόρτωση Δημιουργών

- Είναι αρκετά συνηθισμένο να υπερφορτώνουμε τους δημιουργούς των κλάσεων.

```
class Car
{
        private int position;

        public Car(){
                this.position = 0;
        }

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

        public void move(){
                position ++ ;
        }

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

}

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

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

        public Car(){}

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

        public void move(){
                position ++ ;
        }

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

}

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

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

Γενικά είναι καλό να ορίζετε και ένα constructor χωρίς ορίσματα

Υπερφόρτωση – Προσοχή I

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

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

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

        public void move(){
                position ++ ;
        }

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

}

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

Θα χτυπήσει λάθος ότι δεν υπάρχει constructor χωρίς ορίσματα

Υπερφόρτωση – Προσοχή II

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

```
class SomeClass
{
  public int aMethod(int x, double y){
    System.out.println("int double");
    return 1;
  }

  public double aMethod(int x, double y){
    System.out.println("int double");
    return 1;
  }

  public int aMethod(double x, int y){
    System.out.println("double int");
    return 1;
  }

  public double aMethod(double x, int y){
    System.out.println("double int");
    return 1;
  }

}
```

A

B

C

D

Ποιοι συνδυασμοί είναι αποδεκτοί?

A	B	✗
A	C	✓
A	D	✓
B	C	✓
B	D	✓
C	D	✗

Υπερφόρτωση – Προσοχή III

- Λόγω της συμβατότητας μεταξύ τύπων μια κλήση μπορεί να ταιριάζει με διάφορες μεθόδους.
- Καλείται αυτή που ταιριάζει ακριβώς, ή αυτή που είναι πιο κοντά.
- Αν υπάρχει ασάφεια θα χτυπήσει ο compiler.

```
class SomeClass
{

  public int aMethod(int x, int y){
    System.out.println("int int");
    return 1;
  }

  public float aMethod(float x, float y){
    System.out.println("float float");
    return 1;
  }

  public double aMethod(double x, double y){
    System.out.println("double double");
    return 1;
  }
}

class OverloadingExample
{
  public static void main(String args[])
  {
    SomeClass anObject = new SomeClass();
    anObject.aMethod(1,1);
  }
}
```

Τι θα τυπώσει η κλήση της μεθόδου?

Τυπώνει “int int”
γιατί ταιριάζει ακριβώς με τις παραμέτρους που δώσαμε

```
class SomeClass
{
  /*
  public int aMethod(int x, int y){
    System.out.println("int int");
    return 1;
  }
  */

  public float aMethod(float x, float y){
    System.out.println("float float");
    return 1;
  }

  public double aMethod(double x, double y){
    System.out.println("double double");
    return 1;
  }
}

class OverloadingExample
{
  public static void main(String args[])
  {
    SomeClass anObject = new SomeClass();
    anObject.aMethod(1,1);
  }
}
```

Τι θα τυπώσει η κλήση της μεθόδου?

Τυπώνει “float float”
γιατί είναι πιο κοντά ακριβώς με
τις παραμέτρους που δώσαμε

Ασάφεια

```
class SomeClass
{
  public double aMethod(int x, double y){
    System.out.println("int double");
    return 1;
  }

  public int aMethod(double x, int y){
    System.out.println("double int");
    return 1;
  }
}

class OverloadingExample
{
  public static void main(String args[])
  {
    SomeClass anObject = new SomeClass();
    anObject.aMethod(1.0,1);
    anObject.aMethod(1,1);
  }
}
```

Τι θα τυπώσει η κλήση της μεθόδου σε κάθε περίπτωση?

Τυπώνει “double int”

Ο compiler μας πετάει λάθος γιατί η κλήση είναι ασαφής (ambiguous)

Αντικείμενα ως ορίσματα

- Μπορούμε να περνάμε αντικείμενα ως ορίσματα σε μία μέθοδο όπως οποιαδήποτε άλλη μεταβλητή
- Οποιαδήποτε κλάση μπορεί να χρησιμοποιηθεί ως παράμετρος.
- Όταν τα ορίσματα ανήκουν στην κλάση στην οποία ορίζεται η μέθοδος τότε η μέθοδος μπορεί να δει (και) τα ιδιωτικά (private)πεδία των αντικειμένων
- Αν τα ορίσματα είναι διαφορετικού τύπου τότε η μέθοδος μπορεί μόνο να καλέσει τις public μεθόδους.

Παράδειγμα

- Ορίστε μια μέθοδο που να μας επιστρέφει την απόσταση μεταξύ δύο οχημάτων.

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

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

  public int getPosition() { return position;}

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

class MovingCarDistance1
{
  public static void main(String args[]){
    Car myCar1 = new Car(1);
    Car myCar2 = new Car(0);
    myCar2.move(2);
    System.out.println("Distance of Car 1 from Car 2: " + computeDistance(myCar1,myCar2));
    System.out.println("Distance of Car 2 from Car 1: " + computeDistance(myCar2,myCar1));
  }

  private static int computeDistance(Car car1, Car car2){
    return car1.getPosition() - car2.getPosition();
  }
}
```

Μια μέθοδος ή ένα πεδίο που χρησιμοποιείται σε μία static μέθοδο πρέπει να είναι επίσης static

Η μέθοδος computeDistance παίρνει σαν όρισμα δύο αντικείμενα τύπου Car

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

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

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

  public int distanceFrom(Car other){
    return this.position - other.position;
  }
}

class MovingCarDistance2
{
  public static void main(String args[]){
    Car myCar1 = new Car(1);
    Car myCar2 = new Car(0); myCar2.move(2);
    System.out.println("Distance of Car 1 from Car 2: " + myCar1.distanceFrom(myCar2));
    System.out.println("Distance of Car 2 from Car 1: " + myCar2.distanceFrom(myCar1));
}
```

Συνήθως προτιμούμε όποια μέθοδος έχει σχέση με την κλάση να την ορίζουμε ως public μέθοδο της κλάσης. Έχουμε επιπλέον ευελιξία γιατί έχουμε πρόσβαση σε όλα τα πεδία της κλάσης

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