ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Υπερφόρτωση

Αντικείμενα σαν ορίσματα

ΥΠΕΡΦΟΡΤΩΣΗ

Η κλάση Car

- Μια κλάση που κρατάει την θέση ενός αυτοκινήτου.
  - Μέθοδος move(): μετακινεί το αυτοκίνητο κατά μία θέση προς τα δεξιά
  - Μέθοδος moveManySteps(int delta): μετακινεί delta θέσεις (αρνητικά ή θετικά).
- Και οι δύο μέθοδοι ουσιαστικά υλοποιούν το move απλά η μία παίρνει όρισμα και η άλλη όχι.
  - Θα ήταν καλύτερα να μπορούσαμε να χρησιμοποιήσουμε το ίδιο όνομα και για τις δύο συναρτήσεις.
- Η Java μας δίνει αυτή τη δυνατότητα μέσω της διαδικασίας της υπερφόρτωσης (overloading)

Υπερφόρτωση (Overloading)

- Η Java μας δίνει τη δυνατότητα να ορίσουμε την πολλές μεθόδους με το ίδιο όνομα μέσω της διαδικασίας της υπερφόρτωσης (overloading)
  - Ορισμός πολλών μεθόδων με το ίδιο όνομα αλλά διαφορετικά ορίσματα, μέσα στην ίδια κλάση.
- Για να μπορεί να γίνει σωστά η υπερφόρτωση θα πρέπει οι μέθοδοι να έχουν διαφορετική υπογραφή
- 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.

```
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);
        }
}
```

Οι μέθοδοι move() και move(int) έχουν διαφορετική υπογραφή

Μετακινεί το όχημα μια θέση μπροστά

Μετακινεί το όχημα μια θέση πίσω

Υπερφόρτωση Δημιουργών

- Είναι αρκετά συνηθισμένο να υπερφορτώνουμε τους δημιουργούς (constructors) των κλάσεων.

Υπερφόρτωση δημιουργών

```
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)

ΑΝΤΙΚΕΙΜΕΝΑ ΣΑΝ ΟΡΙΣΜΑΤΑ

Αντικείμενα ως ορίσματα

- Οποιαδήποτε κλάση μπορεί να χρησιμοποιηθεί ως παράμετρος και μπορούμε να περνάμε αντικείμενα ως ορίσματα σε μία μέθοδο όπως οποιαδήποτε άλλη μεταβλητή

Παράδειγμα

- Ορίστε μια (στατική) μέθοδο που να παίρνει σαν όρισμα δύο οχήματα και να μας επιστρέφει την απόσταση τους.

```
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

Αντικείμενα σαν ορίσματα

- Στον αντικειμενοστραφή προγραμματισμό συνήθως δεν ορίζουμε τέτοιου είδους μεθόδους. Η κλάση αναλαμβάνει να υλοποιεί μεθόδους που αφορούν τα αντικείμενα της
- Οπότε μέσα στην κλάση θα πρέπει να ορίσουμε μια μέθοδο που να μας δίνει την απόσταση. Πως θα το κάνουμε?
- Θα ορίσουμε μια public μέθοδο στην Car που θα παίρνει σαν όρισμα ένα άλλο αντικείμενο Car και θα μας επιστρέφει την απόσταση του από το αντικείμενο που κάλεσε την μέθοδο

Αντικείμενα ως ορίσματα

- Όταν τα ορίσματα είναι της ίδιας κλάσης με αυτή στην οποία ορίζεται η μέθοδος τότε η μέθοδος μπορεί να δει (έχει πρόσβαση) και στα ιδιωτικά (private) πεδία των αντικειμένων
- Αν τα ορίσματα είναι διαφορετικού τύπου τότε η μέθοδος μπορεί να καλέσει μόνο τις public μεθόδους.

Διάβασμα πεδίων

- Η προσπέλαση των πεδίων (για διάβασμα ή γράψιμο) γίνεται με τον ίδιο τρόπο όπως και η προσπέλαση των μεθόδων

```
<όνομα αντικειμένου>.<όνομα πεδίου>
```

Παράδειγμα

- Ορίστε μια μέθοδο της κλάσης Car που να παίρνει σαν όρισμα ένα άλλο αντικείμενο Car και να μας επιστρέφει την απόσταση μεταξύ δύο οχημάτων.

```
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

Στο σημείο αυτό διαβάζουμε τα πεδία position για το αντικείμενο this και other. 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

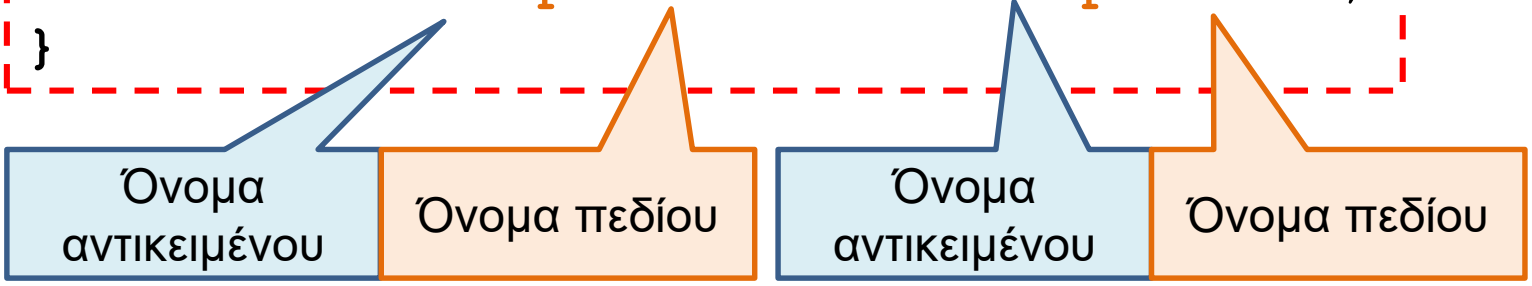
Διάβασμα πεδίων

- Η προσπέλαση των πεδίων (για διάβασμα ή γράψιμο) γίνεται με τον ίδιο τρόπο όπως και η προσπέλαση των μεθόδων

```
<όνομα αντικειμένου>.<όνομα πεδίου>
```

- Και το αντικείμενο this είναι μια τέτοια περίπτωση.

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



Παράδειγμα

- Ορίστε μια μέθοδο που θα παίρνει όρισμα ένα άλλο όχημα και θα βάζει το όχημα που είναι πιο πίσω στην ίδια θέση με το όχημα που είναι πιο μπροστά.

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

   public Car(){}

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

   public void catchUp(Car other){
      if (this.position > other.position){
         this.position = other.position;
      }else{
         other.position = this.position;
      }
   }

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

class MovingCar13{
   public static void main(String args[]){
      Car myCar1 = new Car(); myCar1.move(10);
      Car myCar2= new Car(); myCar2.move(20);
      myCar1.printPosition(); myCar2.printPosition();
      myCar1.catchUp(myCar2);
      myCar1.printPosition(); myCar2.printPosition();
   }
}
```

Μπορούμε όχι μόνο να διαβάσουμε αλλά και να αλλάξουμε την τιμή του πεδίου position στο αντικείμενο other.

Παράδειγμα

- Φτιάξετε μια κλάση που να χειρίζεται ένα λογαριασμό τράπεζας. Κρατάει το όνομα του ιδιοκτήτη και το ποσό.
- Δημιουργείστε και μία μέθοδο που συγχωνεύει δύο λογαριασμούς του ίδιου ατόμου.

```
class BankAccount
{
        private String name;
        private int amount;

        public BankAccount(String name, int amount){
                this.name = name;
                this.amount = amount;
        }

        public void merge(BankAccount other){
                if (this.name.equals(other.name)){
                        this.amount += other.amount;
                }
        }
}
```

Είναι σύνηθες το αποτέλεσμα μιας μεθόδου να αποθηκεύει το αποτέλεσμα της στο ίδιο αντικείμενο το οποίο κάλεσε την μέθοδο.

Π.χ. εδώ το αποτέλεσμα της συγχώνευσης αποθηκεύεται στον λογαριασμό που έκανε την κλήση.

Αντικείμενα σαν ορίσματα – Παράδειγμα Ι

- Θέλουμε να προσομοιώσουμε την κυκλοφορία σε ένα δρόμο.
  - Έχουμε ένα φανάρι που μπορεί να είναι πράσινο, ή κόκκινο. Αλλάζει σε κάθε βήμα
  - Έχουμε ένα όχημα που σε κάθε βήμα κινείται μία θέση, αν το φανάρι δεν είναι κόκκινο.

- Κλάσεις:
  - TrafficLight: κρατάει την κατάσταση του φαναριού και αλλάζει την κατάσταση του
  - Car: Τροποποίηση της move ώστε παίρνει όρισμα το φανάρι και να κινείται μόνο αν το φανάρι δεν είναι κόκκινο.
  - TrafficSimulation: κάνει την προσομοίωση.

```
class TrafficLight
{
  boolean isLightRed = false;

  public void change(){
    isLightRed = !isLightRed;
  }

  public boolean isRed(){
    return isLightRed;
  }

  public void printStatus(){
    if (isLightRed){
       System.out.println(
          "Traffic light is red");
    }else{
       System.out.println(
          "Traffic light is green");
    }
}
```

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

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

   public void move(TrafficLight light){
      if (!light.isRed()){
         position ++;
      }
   }
}
```

```
class TrafficSimulation
{
   public static void main(String[] args){
      TrafficLight light = new TrafficLight();
      Car myCar = new Car();
      for (int i = 0; i < 10; i ++){
          light.printStatus();
          myCar.printPosition();
          myCar.move(light);
          light.change();
      }
   }
}
```

Το όρισμα στην περίπτωση αυτή είναι από άλλη κλάση, άρα δεν μπορούμε να δούμε τα πεδία του, πρέπει να καλέσουμε τη μέθοδο isRed()