Αντικειμενοστρεφής Προγραμματισμός (Object Oriented Programming)

Πολυμορφισμός – Αφηρημένες Κλάσεις

Παναγιώτης Σφέτσος, PhD
http://aetos.it.teithe.gr/~sfetsos/
sfetsos@it.teithe.gr

Περιεχόμενα Μαθήματος

- **Απόκρυψη πεδίων** *(Hiding Fields)*
- **Στατική και Δυναμική Δέσμευση στη Java** *(Static and dynamic binding)*
- **Πολυμορφισμός** *(Polymorphism)*
- **Αφηρημένες κλάσεις** *(Abstract Classes)*

Απόκρυψη Πεδίων *(Hiding Fields) (1/2)*

- Στην υποκλάση ένα **πεδίο με το ίδιο όνομα** πεδίου της υπερκλάσης ***'κρύβει'*** το πεδίο της υπερκλάσης ακόμη και αν είναι διαφορετικού τύπου. Στην υποκλάση, για να έχουμε πρόσβαση στο πεδίο της υπερκλάσης χρησιμοποιούμε τη **super**.

- Η απόκρυψη πεδίων κάνει δύσκολη την αναγνωσιμότητα του κώδικα, για αυτό πρέπει να την αποφεύγουμε.

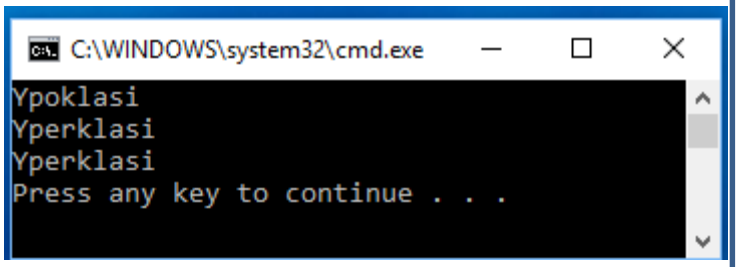
- Το πεδίο στην υποκλάση με το ίδιο όνομα της υποκλάσης είναι ένα **νέο πεδίο**, το **πεδίο της υπερκλάσης είναι κρυμμένο** και δεν μπορούμε να το υπερβούμε. **Δεν υπάρχει υπέρβαση πεδίων, όπως γίνεται με τις μεθόδους.**

- Πρόσβαση σε *κρυμμένα πεδία* έχουμε με:
  (α) χρήση ***αναφοράς υπερκλάσης*** *(δες το παράδειγμα)*
  (β) χρήση ***casting***: **System.out.println(((Super)c1).s);** *(δες στο παράδειγμα)*

Απόκρυψη Πεδίων *(Hiding Fields) (2/2)*

Παράδειγμα

```
class SuperClass
 { String f = "Yperklasi";}

class SubClass extends SuperClass
 {String f = "Ypoklasi";}

public class YpervasiPediou {
  public static void main(String[] args) {
   SubClass p1 = new SubClass();
   System.out.println(p1.f);
   SuperClass p2 = new SubClass(); //αναφορά υπερκλάσης
   System.out.println(p2.f);
   System.out.println(((SuperClass)p1).f); //casting
  }
}
```



Στατική και Δυναμική Δέσμευση στη Java
(Static and dynamic binding) *(1/4)*

Η **δέσμευση** *(binding)* αναφέρεται στον συσχετισμό μεταξύ της *κλήσης μιας μεθόδου* και του *ορισμού (κώδικα) της μεθόδου*.

Static ή Early binding: Η δέσμευση γίνεται κατά τη **μεταγλώττιση του προγράμματος**.

```
class Employee{
 ....
}
class Misthotos extends Employee{
  public void getSalary(){
     System.out.println("O misthotos plironetai");
}
public static void main( String args[]){
  Misthotos obj1 = new Misthotos(); //ορισμός αντικειμένου της υποκλάσης
  obj1.getSalary(); //ο compiler κατά τη μεταγλώττιση καλεί τη μέθοδο
 }                  //γιατί έχει ξεκάθαρη δέσμευση (static binding)
}                    // ...καμία υπέρβαση μεθόδου
```

Στατική και Δυναμική Δέσμευση στη Java
(Static and dynamic binding) (2/4)

Dynamic ή Late binding: Η δέσμευση γίνεται κατά τη **εκτέλεση του προγράμματος**. Το κάθε αντικείμενο έχει πληροφορία για την κλάση του και τον **ορισμό *(κώδικα)*** των μεθόδων του. Ο compiler δεν γνωρίζει τη κλήση/δέσμευση κατά την μεταγλώττιση, παράδειγμα η υπέρβαση των μεθόδων στην κληρονομικότητα *(δες στον πολυμορφισμό).*

```
class Employee{
 public void getSalary(){
  System.out.println("O Employee plironetai"); }

class Misthotos extends Employee{
  public void getSalary(){
     System.out.println("O misthotos plironetai");
}
public static void main( String args[]){
  Employee obj1 = new Misthotos();//αναφορά υπερκλάσης σε αντικειμένου της
                                  //υποκλάσης
  obj1.getSalary();//κατά τη μεταγλώττιση δεν γνωρίζει ποια μέθοδο να καλέσει,
}} //γιατί δεν έχει ξεκάθαρη δέσμευση(τελεστής new) (dynamic binding)
```

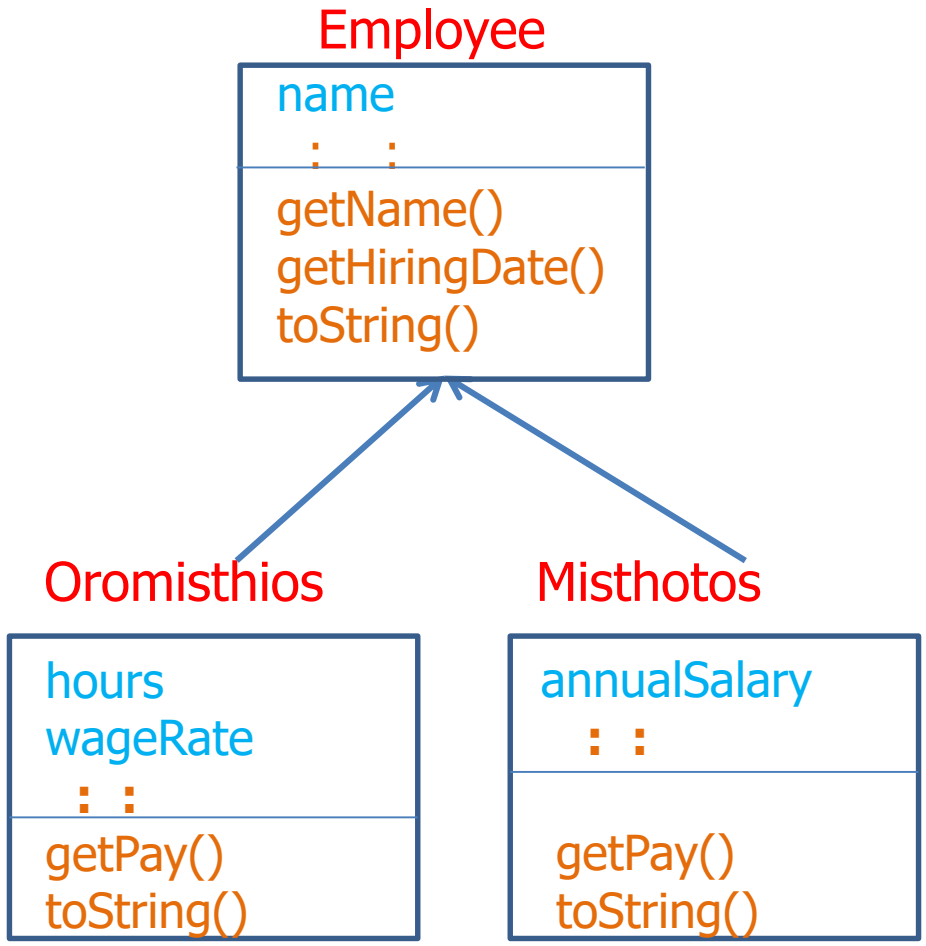
Αποτέλεσμα:
O misthotos plironetai

Πολυμορφισμός της Υποκλάσης
Upcasting και Late binding

Method Overriding (στην υποκλάση)

Στατική και Δυναμική Δέσμευση στη Java
(Static and dynamic binding) (3/4)

Dynamic ή Late binding: Ένα άλλο παράδειγμα αφορά την μέθοδο **toString()** στην κληρονομική ιεραρχία:




```
Employee e;
e = new Oromisthios();
System.out.println(e);
e = new Misthotos();
System.out.println(e);
```

Dynamic ή Late binding:
Ο κώδικας που εκτελείται για την **toString()** εξαρτάται από την κλάση του αντικειμένου την ώρα της κλήσης (Oromisthios ή Misthotos) και όχι την ώρα της δήλωσης *(Employee)*.

Πολυμορφισμός

Στατική και Δυναμική Δέσμευση στη Java
(Static and dynamic binding) (4/4)

Στατική vs Δυναμική Δέσμευση

- Η *Στατική δέσμευση* συμβαίνει κατά τον χρόνο της μεταγλώττισης, ενώ η *Δυναμική* κατά τον χρόνο της εκτέλεσης.

Στη Java:

- Η δέσμευση των **private**, **static** και **final** μεθόδων γίνεται κατά την μεταγλώττιση.
- Η δέσμευση των **υπερβατικών μεθόδων** *(overridden methods)* γίνεται κατά την εκτέλεση του προγράμματος.
- Η δέσμευση των **υπερφορτωμένων μεθόδων** *(overloaded methods)* γίνεται κατά την μεταγλώττιση.
- Στη Java εφαρμόζεται ο μηχανισμός του *Dynamic binding* για όλες τις μεθόδους *(σε αντίθεση με άλλες γλώσσες)*

Πολυμορφισμός *(Polymorphism)*

Χρησιμοποιούμε Πολυμορφισμό υποκλάσης για να εκτελέσουμε διαφορετικές μορφές της ίδιας μεθόδου.

Πολυμορφισμό έχουμε:

- Όταν ο αποστολέας ενός μηνύματος δεν χρειάζεται να γνωρίζει την κλάση του παραλήπτη *(στιγμιότυπου).*
- Όταν μια λειτουργία μπορεί να υλοποιηθεί με το ίδιο όνομα αλλά με διαφορετικό περιεχόμενο σε διαφορετικές κλάσεις *(τρόπο λειτουργίας)*.
- Στον πολυμορφισμό μια μεταβλητή της υπερκλάσης ***αναφέρεται σε ένα αντικείμενο της υποκλάσης***. Έτσι παρέχεται η δυνατότητα σε μια μεταβλητή αναφοράς αντικειμένου να **αλλάζει συμπεριφορά** *(εκτελεί διαφορετικές μεθόδους)* ανάλογα με το αντικείμενο στο οποίο αναφέρεται την κάθε φορά.

Πολυμορφισμός *(Polymorphism)*

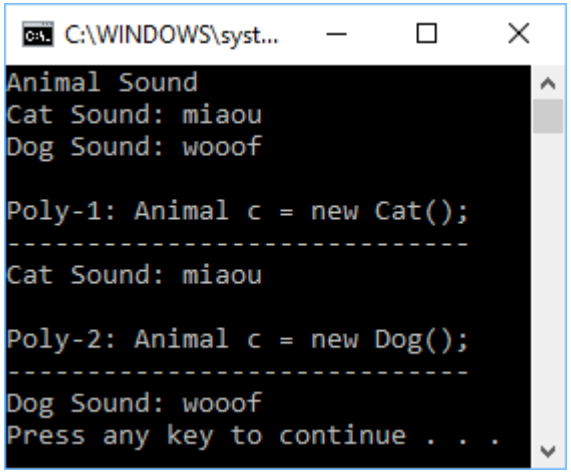
- Ο **πολυμορφισμός** αναφέρεται στη δυνατότητα να χειριζόμαστε αντικείμενα που ανήκουν στην ίδια ιεραρχία κλάσεων, σαν να ήταν αντικείμενα της υπερκλάσης. Για την επίτευξη πολυμορφικής συμπεριφοράς απαιτείται μία **ιεραρχία κλάσεων** και **υπερκαλυπτόμενες μέθοδοι**.

- Θα δούμε παραδείγματα:
  - Πολυμορφισμού υποκλάσης (υπέρβαση μεθόδων)
  - Run Time πολυμορφισμού
  - Πολυμορφισμού Υποκατάστασης
  - Στατικού πολυμορφισμού vs. Δυναμικού πολυμορφισμού

- Δες τα παραδείγματα *(προηγούμενες διαφάνειες (6), (7))* όπου έχουμε ***αναφορά υπερκλάσης σε αντικείμενο υποκλάσης.***

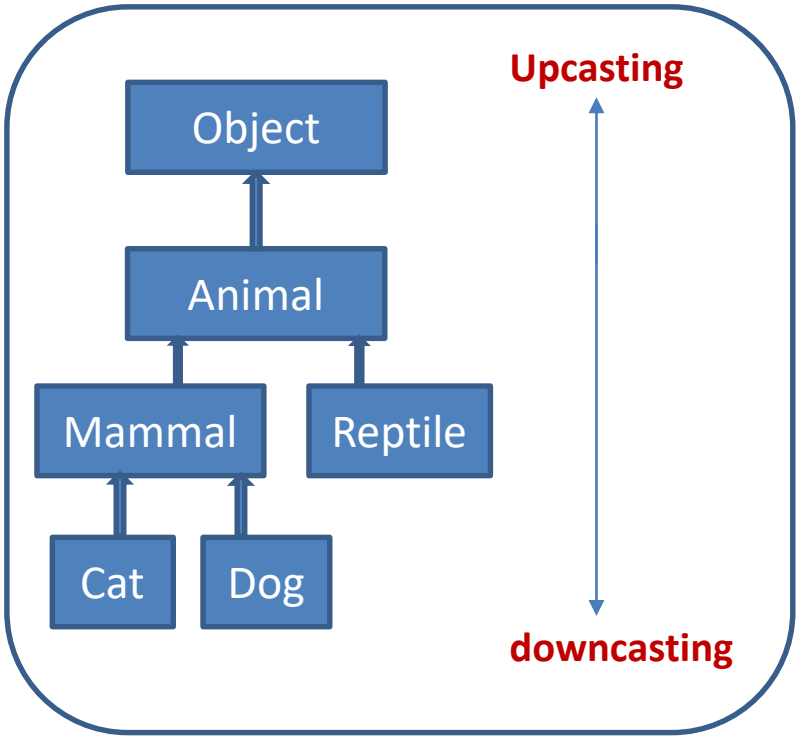
Τυπικό Παράδειγμα Πολυμορφισμού

```
class Animal
{
  void sound() {System.out.println("Animal Sound");}
}

class Cat extends Animal
 {
   void sound()  {// Overriding
     System.out.println("Cat Sound: miaou"); }
 }
```




```
class Dog extends Animal
 {
  void sound() {// Overriding
    System.out.println("Dog Sound: wooof");} }

class testPoly {
  public static void main(String[] args) {
    Animal animal = new Animal();
    animal.sound();
    Cat cat = new Cat();
    cat.sound();
    Dog dog = new Dog();
    dog.sound();
    System.out.println();
    System.out.println("Poly-1: Animal c = new Cat();");
    System.out.println("-----------------------------------");
    Animal c = new Cat(); //ότι η κλάση και όχι η αναφορά
    c.sound();                  // η cat sound()
    System.out.println();
    System.out.println("Poly-2: Animal c = new Dog();");
    System.out.println("-----------------------------------");
    Animal d = new Dog(); //ότι η κλάση και όχι η αναφορά
    d.sound();   //dog sound()
   } }
```

Πολυμορφισμός της Υποκλάσης - Upcasting – Downcasting και Late binding *(1/3)*



- **Casting:** Η ανάθεση του αντικειμένου ενός τύπου σε αναφορά άλλου τύπου.
- Επιτρέπονται *castings* μόνο μεταξύ αντικειμένων σε μια ιεραρχία κληρονομικότητας.
- **Upcasting:** Η ανάθεση ενός αντικειμένου της υποκλάσης σε αναφορά της υπερκλάσης.
  Δεν χρειάζεται κάποιο προσδιοριστικό, γιατί η υποκλάση είναι μια εξειδίκευση της υπερκλάσης.
  π.χ. **Animal c = new Cat();** //late binding
- Δεν αλλάζει το αντικείμενο Cat αλλά το χειρίζεσαι σαν οποιοδήποτε άλλο αντικείμενο της Animal. Κρύβεις τις ιδιότητες της Cat, μέχρι να το κάνεις ξανά downcast σε Cat.

```
Cat c = new Cat();
System.out.println(c);   //Cat@a90653
Mammal m = c;            // upcasting

System.out.println(m); //Cat@a90653
```

Το αντικείμενο Cat δεν άλλαξε μετά το upcasting, αλλά μετονομάστικε σε Mammal. Όμως δεν γίνεται cast σε Dog.

Οι ορισμοί (1) Mammal m=new Cat(); και (2) Mammal m=(Mammal)new Cat(); Είναι ισοδύναμοι. Ο (2) δεν απαιτείται.

Πολυμορφισμός της Υποκλάσης - Upcasting – Downcasting και Late binding *(2/3)*

- Το **downcasting** *(από την υπερκλάση στην υποκλάση)* δεν γίνεται αυτόματα, αλλά ρητά προγραμματιστικά *(τον τύπο μέσα σε αγκύλες).*

```
Cat c1 = new Cat();
Animal a = c1;          //αυτόματο upcasting σε Animal
Cat c2 = (Cat) a;       // downcasting σε αντικείμενο τύπου Cat
```

- Πριν από κάποιο downcasting μπορούμε να ελέγξουμε τον τύπο του αντικειμένου με την **instanceof**. Π.χ.

```
Cat c1 = new Cat();
Animal a = c1;                  //upcasting σε Animal
// έλεγχος αν το Animal είναι ένα αντικείμενο Cat
if(a instanceof Cat)
{
   System.out.println("Einai antikeimeno Cat");
   Cat c2 = (Cat)a;             //ασφαλές downcasting
}
```

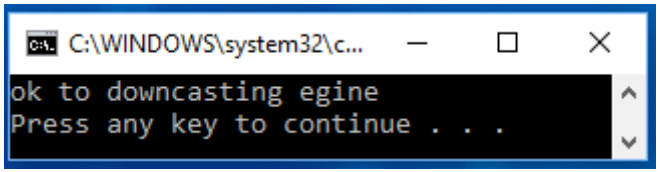
Πολυμορφισμός της Υποκλάσης - Upcasting – Downcasting και Late binding *(3/3)*

Upcasting μέσω της κλήσης μεθόδου

```
public static void walk(Animal a) {
        System.out.println("Animal walk "+a);
 }
:       :
Cat c = new Cat();
Dog d = new Dog();
walk(c);     // αυτόματο upcast σε Animal
walk(d);     // αυτόματο upcast σε Animal
```

Downcasting μέσω της κλήσης μεθόδου

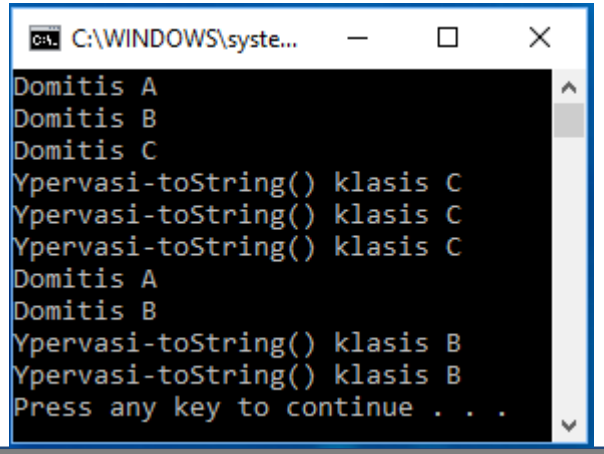
```
class Animal { }
class Dog extends Animal {
  static void method(Animal a) {
      Dog d=(Dog)a;  //downcasting
      System.out.println("ok to downcasting egine");}

  public static void main (String [] args) {
   Animal a=new Dog();
   Dog.method(a); }}
```



Παράδειγμα Upcasting – Downcasting

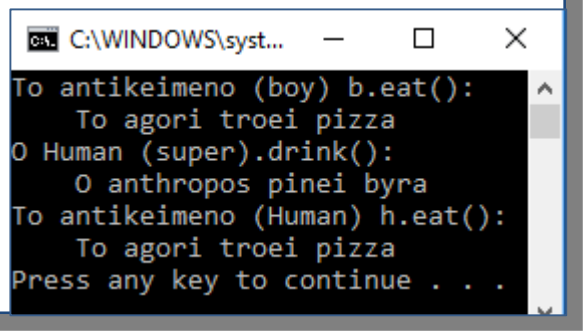
```
class A {
  public A() {
    System.out.println("Domitis A");}
  public String toString(){return "toString() klasis A";
}}

class B extends A {
  public B() {
    super();
    System.out.println("Domitis B");}
  @Override
  public String toString() {
    return "Ypervasi-toString() klasis B";}}

class C extends B {
  public C() {
    super();
    System.out.println("Domitis C");}
  @Override
  public String toString() {
    return "Ypervasi-toString() klasis C";}}
```

```
public class TestCasting {
  public static void main(String[] args) {
    A a1 = new C();   // ok, upcast
    System.out.println(a1); //ekteleitai toString() tis C-klasis
    B b1 = (B)a1;          // ok, downcast
    System.out.println(b1); // -->>--metonomasia tou
                            // antikeimenou se B1
    C c1 = (C)b1;          // ok, downcast
    System.out.println(c1); //-->>--metonomasia tou
                            //  antikeimenou se C1
    A a2 = new B();        // ok, upcast
    System.out.println(a2); //ekteleitai toString() tis B-klasis
    B b2 = (B)a2;          // ok, downcast
    System.out.println(b2); //-->>--metonomasia tou
                            //antikeimenou se b2
  }
}
```



Παράδειγμα 2ο - *RunTime Polymorphism*

```
class Human{
   public void eat() {System.out.println("    O anthropos troei pizza");}
   public void drink() {System.out.println("    O anthropos pinei byra");}}
class Boy1 extends Human{
   public void eat(){System.out.println("    To agori troei pizza");}
   public void drink(){System.out.println("    To agori pinei byra");}
 public void A_method() {
   Boy1 b = new Boy1();
   Human h=new Boy1();
   System.out.println("To antikeimeno (boy) b.eat():");
   b.eat();
   System.out.println("O Human (super).drink():");
   super.drink();
   System.out.println("To antikeimeno(Human) h.eat():");
   h.eat(); //υπέρβαση-overriding, Runtime polymorphism
 }
public static void main( String args[]) {
      Boy1 obj = new Boy1();
      obj.A_method(); }}
```



Παράδειγμα 3ο - *RunTime Polymorphism – Virtual methods (1/3)*

Πολυμορφισμός με υπέρβαση μεθόδων. Δες την κλήση και χρήση της μεθόδου **AddressCheck()**. Αυτές οι μέθοδοι λέγονται και **Virtual Methods**.

```
class Employee {
   private String name;
   private String address;
   private int AFM;
   public Employee(String name, String address, int afm) {
      System.out.println("Domitis employee");
      this.name = name;
      this.address = address;
      this.AFM = afm;}
   public void AddressCheck() {
      System.out.println("H Dieythinsi tou employee " + this.name + " einai:
                         " + this.address); }
   public String toString(){return name + " " + address + " " + AFM;}
   public String getName() {return name;}
   public String getAddress(){return address; }
   public void setAddress(String a){address = a;}
   public int getAFM() {return AFM;} }
```

Παράδειγμα 3ο - *RunTime Polymorphism – Virtual methods (2/3)*

```
class Misthotos extends Employee {
   private double Misthos;
   public Misthotos(String n, String a, int afm, double m) {
      super(n, a, afm);
      setMisthos(m);}

   public void AddressCheck() {
      System.out.println("Elegxos dieythinsis misthotou");
      System.out.println("H dieythinsi tou misthotou " + getName()
      + " me etisio mistho= " + Misthos);}

   public double getMisthos() {
      return Misthos; }

   public void setMisthos(double m) {
      if(m >= 0.0) {
         Misthos = m; }}

   public double MisthosAnaMina() {
     return Misthos/12;}
}
```

Παράδειγμα 3ο - *RunTime Polymorphism – Virtual methods (3/3)*

```
class VirtualMethodsDemo {
 public static void main(String [] args) {
  Misthotos s = new Misthotos("Nikos Nikas", "Tsimiski 32", 4, 12000.00);
  Employee e = new Misthotos("Paulos Pauloy", "Mitropoleos 24", 7, 20000.00);
  System.out.println("\nKlisi tis AddressCheck() me anafora ston misthoto");
  s.AddressCheck();
  System.out.println("O miniaios misthos toy misthotou "+ s.getName()+"
            einai= "+s.MisthosAnaMina());
  System.out.println("\nKlisi tis AddressCheck() me anafora ston employee");
  e.AddressCheck();
  System.out.println(); } }
```

Virtual Method invocation

```
C:\WINDOWS\system32\cmd.exe
Domitis employee
Domitis employee

Klisi tis AddressCheck() me anafora ston misthoto --
Elegxos dieythinsis misthotou
H dieythinsi tou misthotou Nikos Nikas me etisio mistho= 12000.0
O miniaios misthos toy misthotou Nikos Nikas einai= 1000.0

Klisi tis AddressCheck() me anafora ston employee--
Elegxos dieythinsis misthotou
H dieythinsi tou misthotou Paulos Pauloy me etisio mistho= 20000.0

Press any key to continue . . .
```

Κατά την μεταγλώττιση καλείται η *AddressCheck()* της Employee, ενώ κατά την εκτέλεση η JVM καλεί την *AddressCheck()* του Μισθωτού.

Γιατί δύο ίδια μηνύματα ;

Πολυμορφισμός Υποκατάστασης *(Substitutability)*

Είδαμε την αρχή της υποκατάστασης της Liskov *(ένα αντικείμενο της υποκλάσης μπορεί να κάνει ότι και το αντικείμενο της υπερκλάσης – άρα μπορούμε να **υποκατα-στήσουμε το αντικείμενο της υπερκλάσης με ένα της υποκλάσης)***. Αν μια αναφορά της υπερκλάσης αναφέρεται σε αντικείμενο της υποκλάσης, ***τότε εκτελούνται οι μόνο οι μέθοδοι της υπερκλάσης.***

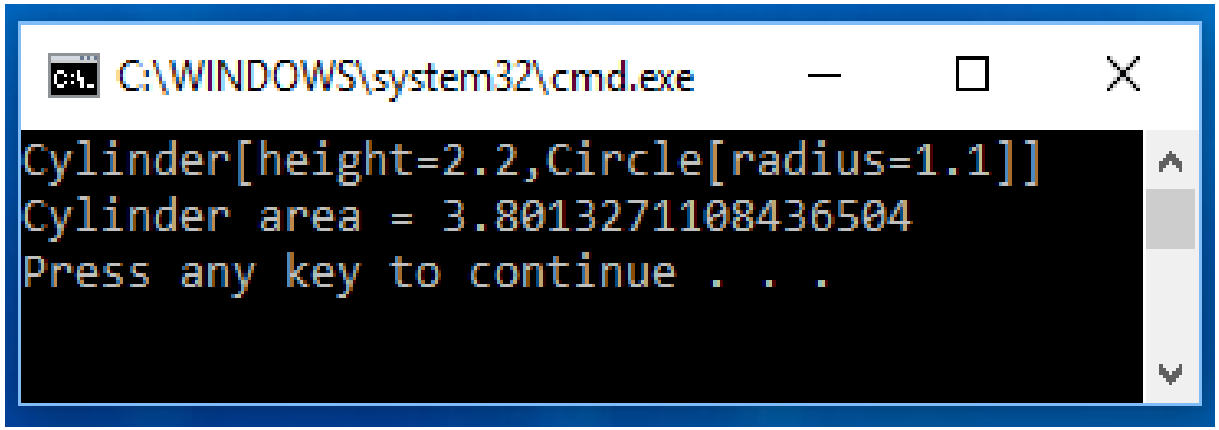
```
class Circle {
  private double radius;
  public Circle(){};
  public Circle(double radius) {this.radius = radius;}
  public double getRadius() {return this.radius;}
  public double getArea(){return radius * radius * Math.PI;}
  public String toString() {
     return "Circle[radius=" + radius + "]";   } }
```

Πολυμορφισμός Υποκατάστασης *(Substitutability)*

```
class Cylinder extends Circle {
   private double height;
   public Cylinder() {
    super();
    height = 1.0; }
   public Cylinder(double height) {
      super();
      this.height = height;}
   public Cylinder(double radius, double height) {
      super(radius);
      this.height = height;}
   public double getHeight() {return height;}
   public double getVolume() {return getArea()*height;}
   public String toString() {
    return "Cylinder[height=" + height + "," + super.toString() + "]";
   }
}
```

Πολυμορφισμός Υποκατάστασης *(Substitutability)*

```
class TestCylinder {
 public static void main (String[] args) {
   // Ypokatastasi antikeimenou tis yperklasis me antikeimeno tis ypoklasis
   Circle c1 = new Cylinder(1.1, 2.2);
   System.out.println(c1.toString());  // overridden
   System.out.println("Cylinder area = "+c1.getArea()); // overridden
 }
}
```



Πολυμορφισμό θα δούμε και στις Αφηρημένες κλάσεις και στις Διεπαφές/Διασυνδέσεις

Στατικός Πολυμορφισμός Vs Δυναμικός Πολυμορφισμός *(1/4)*

Στατικός Πολυμορφισμός: Υπερφόρτωση Μεθόδων *(Method overloading)*

Δυναμικός Πολυμορφισμός: Υπέρβαση Μεθόδων *(Method Overriding)*

```
class Employee
{
  public String FirstName;
  public String LastName;
  public String GetFullName()
  {return FirstName + " " + LastName; } }

class Monimos extends Employee
 {
    //Overriding method
    public String GetFullName()
    {return "Monimos: " + FirstName + " " + LastName; }}
```

Στατικός Πολυμορφισμός Vs Δυναμικός Πολυμορφισμός *(2/4)*

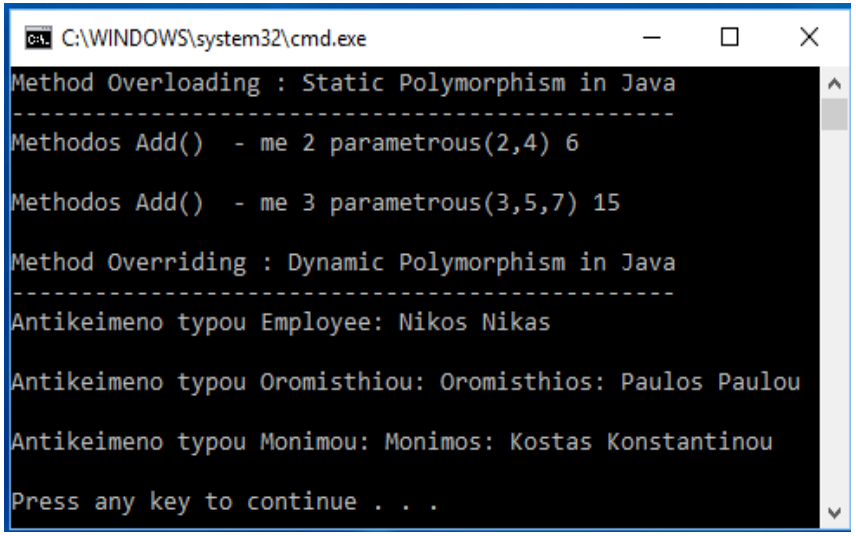
```
class Oromisthios extends Employee
{
 //overriding method
 public String GetFullName()
   {return "Oromisthios: " + FirstName + " " + LastName; }}

class PolyStaticVsDynamic {
 public static void main(String[] args) {
  String newLine = System.getProperty("line.separator");
  System.out.println("Method Overloading : Static Polymorphism in Java");
  System.out.println("-----------------------------------------");
  System.out.println("Methodos Add() - me 2 parametrous(2,4) " +
      Add(2,4) + newLine);
  System.out.println("Methodos Add() - me 3 parametrous(3,5,7) " +
     Add(3,5,7) + newLine);
```

Στατικός Πολυμορφισμός Vs Δυναμικός Πολυμορφισμός *(3/4)*

```
System.out.println("Method Overriding : Dynamic Polymorphism in Java");
System.out.println("-----------------------------------------");
Employee e = new Employee();
e.FirstName = "Nikos";
e.LastName = "Nikas";
System.out.println("Antikeimeno typou Employee: " + e.GetFullName() +
  newLine);
//anafora ston oromisthio
e = GetEmployee(2);
e.FirstName = "Paulos";
e.LastName = "Paulou";
System.out.println("Antikeimeno typou Oromisthiou: " + e.GetFullName()+
  newLine);
//point e to PermanentEmployee
e = GetEmployee(3);
e.FirstName = "Kostas";
e.LastName = "Konstantinou";
System.out.println("Antikeimeno typou Monimou: " + e.GetFullName() +
   newLine); }
```

Στατικός Πολυμορφισμός Vs Δυναμικός Πολυμορφισμός *(4/4)*

```
//Factory pattern: Epistrefei ton ypal. analoga me ton typo pou stelnoume
 private static Employee GetEmployee(int type)
  {
   switch(type)
    {
     case 1: return new Employee();
     case 2: return new Oromisthios();
     case 3: return new Monimos();
    }
     return new Employee();}

 //methods overloadding
 private static int Add(int x, int y)
   {return x + y;}
 private static int Add(int x, int y, int z)
   {return x + y + z;}
}
```



Άσκηση -1 *(Πολυμορφισμός)*

Να γίνει το πρόγραμμα που χειρίζεται την πώληση και ενοικίαση κατοικιών. Το πρόγραμμα θα ορίζει την κλάση **Katoikia**, με private μέλη: (1) ***Κωδικός,*** int *(1=πώληση, 2=ενοικίαση),* (2) ***Πλήθος Δωματίων****,* int, (3) ***Τετραγωνικά μέτρα***, double, (4) ***Διεύθυνση,*** String, και τις μεθόδους: (1) ***TyposSynalagis()****,* που θα επιστρέφει με μήνυμα, τον τύπο της συναλαγής, δηλ. “Polisi”, ”Enoikio”, ανάλογα με τον κωδικό, (2) ***Poso()***, που θα υπολογίζει το τελικό ποσό συναλλαγής, λαμβάνοντας υπόψιν τα παρακάτω: Αν είναι για ενοίκιο, τότε η τιμή προσαυξάνεται επί δύο (προκαταβολή ενός ενοικίου) συν (+) μισό ενοίκιο (μισή ***τιμή ενοικίου****)* σαν προμήθεια. Αν είναι πώληση τότε η ***τιμή πώλησης*** επιβαρύνεται με 1200 Ευρώ (προμήθεια). Οι κλάσεις “**Diamerisma**”, “**ExohikiKatoikia**” και “**Mesonette**” κληρονομούν/εξειδικεύουν την κλάση “Katoikia” και διαθέτουν επιπλέον τα παρακάτω χαρακτηριστικά:
Diamerisma: TimiD *(double)*, **ExohikiKatoikia:** TimiEx *(double)*, **Mesonette:** TimiM *(double)* και υλοποιούν με υπέρβαση τις μεθόδους: ***TyposSynalagis()*** και ***Poso()***, Στο κυρίως πρόγραμμα δημιουργείστε ένα πίνακα **Ν - κατοικιών**. Το Ν εισάγεται από το Πληκτρολόγιο. Σε κάθε θέση του πίνακα μπορεί να ανατίθεται αντικείμενο τύπου “Diamerisma” (60%-στον πίνακα), “ExohikiKatoikia” (20%-στον πίνακα), ή τύπου “Mesonette” (20%-στον πίνακα). Το πρόγραμμα εμφανίζει τα χαρακτηριστικά κάθε κατοικίας, καθώς και τα αποτελέσματα των μεθόδων και *TyposSynalagis()* και *Poso()*. Ορίστε δομητές και μεθόδους getter(), όπου χρειάζεστε. Όλα τα χαρακτηριστικά να είναι private. Υπερβείτε την μέθοδο ***toString()*** της Object.

Αφηρημένη Κλάση *(abstract class) (1/9)*

Αφηρημένοι Τύποι Δεδομένων *(Abstract Data Types – ADT):*

Αφηρημένες κλάσεις και Διεπαφές/Διασυνδέσεις

Αφηρημένη κλάση *(abstract class)* είναι μια κλάση που ***αντιπροσωπεύει μια ευρεία κατηγορία τύπων*** και ορίζεται μόνο για να ***κληρονομηθεί*** *(επεκταθεί)* ***από θυγατρικές υποκλάσεις*** *(π.χ. liquid, people, mammal, employee, κλπ.)*

Σύνταξη *(η δεσμευμένη λέξη abstract):*

(1) κλάση: *<visibility>* **abstract class** *<name>* **{ }**

(2) μέθοδος: *<visibility>* **abstract** *<type> <name> ([parameters]);*

Κανόνες και Συνθήκες:

- Δεν μπορούμε να δημιουργήσουμε αντικείμενα μιας αφηρημένης κλάσης *(τελεστής new)*, αλλά μπορούμε να ορίσουμε δομητές *(και να έχουμε πρόσβαση από τους δομητές των υποκλάσεων).*
- Μια αφηρημένη μέθοδος δεν μπορεί να υπάρχει σε μια μη αφηρημένη κλάση. Αν μια υποκλάση δεν υλοποιεί όλες τις αφηρημένες μεθόδους, τότε πρέπει να δηλωθεί abstract.

Αφηρημένη Κλάση *(abstract class) (2/9)*

- Η αφηρημένη κλάση ορίζει απλώς ένα ***"συμβόλαιο"*** το οποίο θα πρέπει να ακολουθούν οι υποκλάσεις της όσον αφορά τις υπογραφές των μεθόδων τους.
- Οι ***αφηρημένες μέθοδοί*** της είναι απλώς ένας *ορισμός της υπογραφής τους* και **εναπόκειται στις *υποκλάσεις να τις ορίσουν και υλοποιήσουν*** *(συμβόλαιο)*.
- Δηλαδή, η αφηρημένη μέθοδος δεν έχει σώμα και τελειώνει με το (;)
- Οι ***αφηρημένες μέθοδοι*** πρέπει να είναι **public** *(ή **protected**), όχι private*.
- Ο όρος **abstract** *δεν μπορεί να χρησιμοποιηθεί* σε **στατικές** μεθόδους ή δομητές.

Αφηρημένη Κλάση *(abstract class) (3/9)*

- Μία αφηρημένη κλάση μπορεί να έχει ***και μη αφηρημένες μεθόδους*** οι οποίες υλοποιούνται στην ίδια την κλάση *(αν και μπορούμε να τις υπερβούμε στις υποκλάσεις).* Αν δεν περιέχει καμία abstract μέθοδο λειτουργεί μεν σαν υπερκλάση, ωστόσο δεν μπορούμε να ορίσουμε αντικείμενα της *(τελεστής new).*

- Μια υποκλάση μπορεί να δηλωθεί abstract, ακόμη και αν η υπερκλάση είναι μη abstract.

- Μια abstract κλάση μπορεί να χρησιμοποιηθεί σαν **τύπος δεδομένων** π.χ. Array - αντικειμένων, τότε μπορούμε να χρησιμοποιήσουμε τον τελεστή new. π.χ. **People[ ] objs = new People[5];**
  Μετά γεμίζουμε τον πίνακα με αναφορές αντικειμένων, υποκλάσεων:
  π.χ. **People [0]= new Student();** ***(πολυμορφική συμπεριφορά)***

Αφηρημένη Κλάση *(abstract class)* *(4/9)*

Παράδειγμα 1ο :

Δημιουργούμε δύο Shape – αντικείμενα: ένα κύκλο και ένα ορθογώνιο που υλοποιούν τις δύο *abstract μεθόδους* getArea() και getPerimeter().

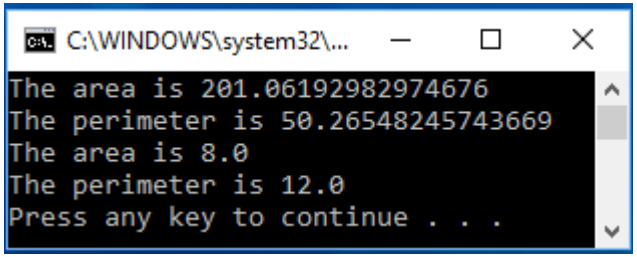
```
abstract class Shape {
  private String color;
  private boolean filled;

  /** Abstract method getArea */
  public abstract double getArea();

  /** Abstract method getPerimeter */
  public abstract double getPerimeter(); }

class Circle extends Shape {
  private double radius;
  public Circle(double r) {this.radius = r;}
  public void setRadius(double r) {this.radius = r;}
  public double getRadius() {return radius;}
```

```
@Override /** Return area */
public double getArea()
 {return radius * radius * Math.PI;}
public double getDiameter()
 {return 2 * radius;}
@Override /** Return perimeter */
public double getPerimeter()
 {return 2 * radius * Math.PI;}

public void printCircle() {
  System.out.println("The radius is
          + radius);} }
```

Αφηρημένη Κλάση *(abstract class)* *(5/9)*

```
class Rectangle extends Shape {
  private double width;
  private double height;
  public Rectangle(double width, double
height) {
    this.width = width;
    this.height = height;}
  public double getWidth()
    {return width;}
  public void setWidth(double width)
     {this.width = width;}
  public double getHeight()
     {return height;}
  public void setHeight(double height)
     {this.height = height;}
  @Override /** Return area */
  public double getArea()
    {return width * height;}
  @Override /** Return perimeter */
  public double getPerimeter()
    {return 2 * (width + height);}}
```

```
class TestShape {
  public static void main(String[] args) {
    Shape Object1 = new Circle(8);
    Shape Object2 = new Rectangle(2, 4);
    // Display circle
    displayObject(Object1);
    // Display rectangle
    displayObject(Object2);  }

  public static void displayObject(
        Shape object) {
    System.out.println("The area is " +
            object.getArea());
    System.out.println("The perimeter is «
        +   object.getPerimeter()); } }
```



Αφηρημένη Κλάση *(abstract class)* *(6/9)*

Παράδειγμα 2ο με **αφηρημένη κλάση** την Employee που περιέχει δύο *abstract μεθόδους* τις **getEmpType()** και **calcSalary(),** που θα υλοποιηθούν στις κλάσεις απογόνους.

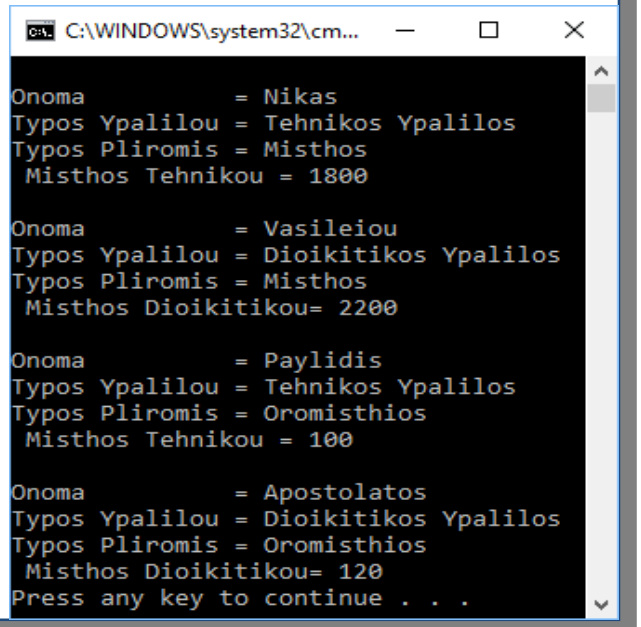
```
abstract class Employee {
  protected String name;
  protected int bonus;  //bonus
  protected int hours;  //hours
  protected int payType; //0=salary, 1=byhour

  Employee(String s, int b, int h, int p)
    {name=s; bonus=b; hours=h; payType=p;}
  public String getName() {return name;}
  public String getPayType() {
    String pType;
    if (payType==0) pType="Misthos";
    else Type="Oromisthios";
    return pType; }

   abstract String getEmpType();
   abstract void calcSalary();
}
```

```
class Administrator extends Employee {
  Administrator(String s, int b, int h, int p)
    {super(s,b,h,p);}
  public String getEmpType()
     {return "Dioikitikos Ypalilos";}
  public void calcSalary() {
    int s=0;
    if (payType==0) s=1200+bonus;
       //vasikos=1200 + bonus
    else s=(hours*12);  //12 Euro per hour
       System.out.println(" Misthos
             Dioikitikou= " + s); }}
```

Αφηρημένη Κλάση *(abstract class)* *(7/9)*

```
class Technical extends Employee {
 public Technical(String s, int b, int h, int p)
   {super(s,b,h,p);}

 public String getEmpType()
   {return "Tehnikos Ypalilos";}
 public void calcSalary() {
 int s=0;
 if (payType==0) s=800+bonus;
   //vasikos=800 + bonus
 else s=(hours*10); //10 Euro per hour
  System.out.println(" Misthos Tehnikou = "+s);
  }
}

class Company {
 Employee emp[]= new Employee[4];
 public void setEmployee(Employee e,int a)
    {emp[a]=e;}
 public void printAll() {
       for (int i=0;i<4;i++) {
         System.out.println();
         System.out.println("Onoma         = "+
                 emp[i].getName());
         System.out.println("Typos Ypalilou = "+
                 emp[i].getEmpType());
         System.out.println("Typos Pliromis = "+
                 emp[i].getPayType());
                 emp[i].calcSalary(); } } }

class TestEmployeeAbstract {
  public static void main(String[] args) {
    int a;
    Employee e1=new Technical("Nikas",1000,10,0);
    Employee e2=new Administrator("Vasileiou",1000,20,0);
    Employee e3=new Technical("Paylidis",10,10,1);
    Employee e4=new Administrator("Apostolatos",10,10,1);
    Company c=new Company();
    c.setEmployee(e1,0);
    c.setEmployee(e2,1);
    c.setEmployee(e3,2);
    c.setEmployee(e4,3);
    c.printAll(); } }
```

Οι Αφηρημένες Κλάσεις ωθούν σε αναγκαστική δημιουργία υποκλάσεων και Πολυμορφισμό



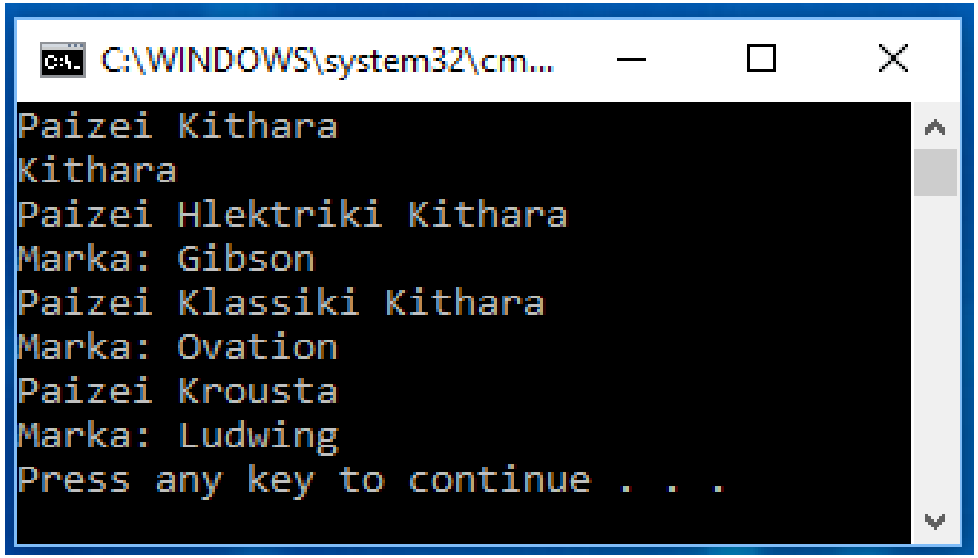
Αφηρημένη Κλάση *(abstract class)* *(8/9)*

Παράδειγμα 3ο με **αφηρημένη κλάση** την **MousikaOrgana** που περιέχει μια *abstract μέθοδο την* ***play()*** και μια μέθοδο που δεν είναι αφηρημένη την ***BrandName()*** **,** που θα υλοποιηθούν στις κλάσεις απογόνους.

```
abstract class MousikaOrgana {
   int i;
   public abstract void play();
   public String BrandName() {
      return "Mousiko Organo ";}}
class Kithara extends MousikaOrgana {
  public void play() {
     System.out.println ("Paizei Kithara"); }
  public String BrandName() {
     return "Kithara";}}
class HlektrikiKithara extends Kithara {
  public void play(){
    System.out.println("Paizei Hlektriki Kithara"); }
  public String BrandName() {
     return "Marka: Gibson";}}
class KlassikiKithara extends Kithara {
   public void play(){
     System.out.println("Paizei Klassiki Kithara");
    }
   public String BrandName() {
      return "Marka: Ovation";}}

class Krousta extends MousikaOrgana {
   public void play(){
     System.out.println("Paizei Krousta"); }
   public String BrandName(){
      return "Marka: Ludwing";}}

public class MusicOrgans{
   static void tune(MousikaOrgana mo) {
      mo.play() ;}
```

Αφηρημένη Κλάση *(abstract class)* *(9/9)*

```
public static void main(String[] args) {
   MousikaOrgana[] o = new MousikaOrgana[4];
   int i = 0;
   o[i++] = new Kithara();
   o[i++] = new HlektrikiKithara();
   o[i++] = new KlassikiKithara();
   o[i++] = new Krousta() ;
   for(i = 0; i < o.length; i++) {
      tune(o[i]);
      System.out.println( o[i].BrandName() );
     }
  }
}
```



Άλυτες Ασκήσεις – 1 και 2 *(Παραλλαγές του 1ου – Παραδείγματος)*

Να γραφεί πρόγραμμα που δημιουργεί την **αφηρημένη κλάση Shape**, η οποία θα περιέχει τα private μέλη - σημεία **X** και **Y** (τύπου double), ένα πλήρη δομητή, αντίστοιχες μεθόδους **gettter()**, την αφηρημένη μέθοδο **area()** που θα υλοποιηθεί στις υποκλάσεις και την αντίστοιχη μέθοδο **toString()**. Επίσης το πρόγραμμα θα δημιουργεί δύο υποκλάσεις που θα κληρονομούν (extends) την shape: τις **Rectangle** και **Circle**.

H Rectangle:

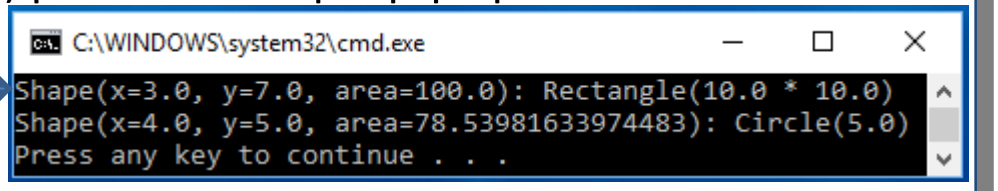
- θα περιέχει τα private μέλη height και width, ένα πλήρη δομητή, την υλοποίηση της area() *(height*width)* και την αντίστοιχη μέθοδο toString().

H Circle:

- θα περιέχει το private μέλος ακτίνα *(radius)*, ένα πλήρη δομητή, την υλοποίηση της area() *(Math.PI* radius * radius)* και την αντίστοιχη μέθοδο toString().

Στην κλάση **AbstractShape** (με την main()), θα δημιουργηθούν δύο αντικείμενα των τύπων Rectangle και Circle σε ένα πίνακα αντικειμένων και θα εκτελείται ο υπολογισμός και η εμφάνιση των αποτελεσμάτων (area()) με τα κατάλληλα μηνύματα.

Π.χ. αν δώσουμε για το obj-Rectangle τις τιμές –σημείων 3 και 7 και διαστάσεις 10,10 και για το obj-Circle τις τιμές-σημείων 4, 5 και radius=5, θα πάρουμε τα αποτελέσματα:



Άσκηση - 2

Ορίστε την αφηρημένη κλάση "**GeometricObject**" η οποία έχει τα χαρακτηριστικά: (1) ***color,*** String, (2) ***filled,***boolean, την μέθοδο ***isFilled()*** που επιστρέφει true ή false ανάλογα, και τις αφηρημένες μεθόδους ***getArea()*** και ***getPerimiter()***.

Οι κλάσεις "**Rectangle**", "**Circle**" και "**Cylinder**" κληρονομούν/εξειδικεύουν την κλάση "GeometricObject" και διαθέτουν επιπλέον τα παρακάτω χαρακτηριστικά και μεθόδους:

Rectangle : (1) ***width***, int, (2) ***height,*** int, και (α) ***getArea()*** (width*height) και (β) ***getPerimeter()*** (2*(width*height)).

Circle : (1) ***radius***, double και (α) ***getArea()*** (radius * radius * Math.PI;) και (β) ***getPerimeter()*** (2*(radius*Math.PI)).

Cylinder : (1) ***height,*** int, (2) ***radius,*** double και (α) ***getArea()*** (2*getArea()*getPerimiter()* height) και (β) ***Volume()*** (getArea()*height).

Στο κυρίως πρόγραμμα δημιουργείστε ένα πίνακα N σχημάτων. Το N εισάγεται από το πληκτρολόγιο. Σε κάθε θέση του πίνακα μπορεί να ανατίθεται αντικείμενο τύπου "Rectangle" (40%-στον πίνακα), "Circle" (30%-στον πίνακα), ή τύπου "Cylinder" (30%-στον πίνακα).

Το πρόγραμμα εμφανίζει τα χαρακτηριστικά κάθε σχήματος, καθώς και τα αποτελέσματα των μεθόδων isFilled(), getArea() και getPerimiter().