Αντικειμενοστρεφής Προγραμματισμός (Object Oriented Programming)

Διασυνδέσεις / Διεπαφές

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

Περιεχόμενα Μαθήματος

- **Διεπαφές/Διασυνδέσεις** *(Interfaces)*

Διεπαφές/Διασυνδέσεις *(Interfaces) (1/13)*

Η Διεπαφή περιέχει ***ένα σύνολο από αφηρημένες μεθόδους*** που ***ορίζουν μία συμπεριφορά,*** την οποία συμπεριφορά αποκτούν όσες κλάσεις υλοποιήσουν τη συγκεκριμένη Διεπαφή *(υλοποιώντας αυτές τις αφηρημένες μεθόδους).*

- Η διεπαφή είναι μια έννοια ***ειδικής κλάσης***, παρόμοιας με την αφηρημένη κλάση, που μπορεί να περιέχει μόνο **static σταθερές** και **αφηρημένες μεθόδους** *(μόνο τις υπογραφές των μεθόδων, χωρίς υλοποίηση/σώμα)***.**

- Οι διεπαφές μεταγλωττίζονται σε ξεχωριστά bytecode αρχεία *(.class)* όπως και οι κανονικές κλάσεις.

- Όπως και στις αφηρημένες κλάσεις ***δεν μπορούμε να ορίσουμε αντικείμενα της διεπαφής με τον τελεστή new.***

- Μια κλάση μπορεί να υλοποιεί πολλές διεπαφές.

Διεπαφές/Διασυνδέσεις *(Interfaces) (2/13)*

- **Δεν περιέχει** πεδία, δομητές και στατικές μεθόδους.
- Το σύνολο των μεθόδων πρέπει να υλοποιηθεί από άλλη κλάση/σεις.
- Όλες οι μεταβλητές πρέπει να είναι **public, static** και **final** *(άμεσα ή έμμεσα)*.Οι μέθοδοι **public**.
- Μια διεπαφή ορίζεται χρησιμοποιώντας την λέξη κλειδί - **interface** αντί της δεσμευμένης λέξης class.

Σύνταξη:

```
public interface <interface name>
{
   Δηλώσεις Σταθερών <constant declarations>;
   Υπογραφές  Μεθόδων <method signatures>;
}
```

Διεπαφές/Διασυνδέσεις *(Interfaces) (3/13)*

- Μια κλάση που ***υλοποιεί*** *τη διεπαφή* έχει στην επικεφαλίδα της τη δεσμευμένη λέξη ***implements*** και περιλαμβάνει τις μεθόδους με τον κώδικα που υλοποιεί την λειτουργικότητά τους. Η διεπαφή *'προδιαγράφει'* τις μεθόδους με την υπογραφή τους (όνομα μεθόδου, τύπος παραμέτρων).

class <*ClassName*> **implements** <*InterfaceName*> { ...}

// συνδυασμός κληρονομικότητας και υλοποίηση Διεπαφής

class **<***ClassName*> **extends** <*SuperClass*> **implements** <*InterfaceN-1*, <*InterfaceN-2*, {...}

// σαν πολλαπλή κληρονομικότητα

- Οι διεπαφές μπορούν να **επεκτείνουν | κληρονομούν** *(extends)* άλλες διεπαφές.

Διεπαφές/Διασυνδέσεις *(Interfaces) (4/13)*

- Μια κλάση μπορεί να ***επεκτείνει*** *(extends)* μόνον μία υπερκλάση *(απλή κληρονομικότητα)*, όμως μπορεί να ***υλοποιεί*** *(implements)* περισσότερες από μία διεπαφές *(έμμεσα πολλαπλή κληρονομικότητα)*.

- Οι ιδιότητες του **πολυμορφισμού** ισχύουν και μεταξύ των διεπαφών και των κλάσεων που τις υλοποιούν και των υποκλάσεων τους, διότι και εδώ η υλοποίηση της μεθόδου μιας διεπαφής εξαρτάται από τον τύπο του αντικειμένου που την καλεί.

- Αν μια κλάση δεν ορίσει όλες τις μεθόδους ενός *interface* που υλοποιεί τότε πρέπει να δηλωθεί σαν *abstract.*

Διεπαφές/Διασυνδέσεις *(Interfaces)* *(5/13)*

Γιατί χρησιμοποιούμε τις διεπαφές;

- Για να χρησιμοποιούμε την **λειτουργικότητα** των αντικειμένων και όχι την υλοποίησή τους (που δεν την γνωρίζουμε).
- Για να **εκτελούνται παρόμοιες μέθοδοι** από μη συσχετιζόμενες κλάσεις.
- Για να πετύχουμε **πολλαπλή κληρονομικότητα**.

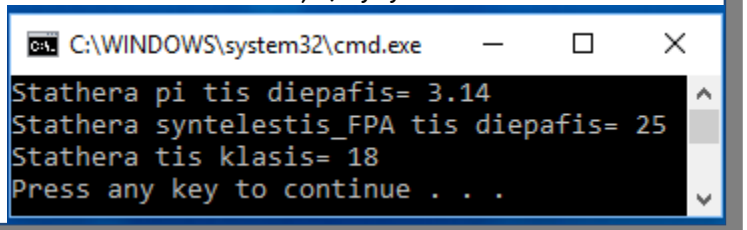
Διεπαφές/Διασυνδέσεις *(Interfaces)* *(6/13)*

Η διεπαφή μπορεί να **ορίσει *σταθερές*** *(που είναι άμεσα ή έμμεσα public, static και final),* που χρησιμοποιούνται απ' ευθείας με το όνομά τους στις *κλάσεις που υλοποιούν την διεπαφή.*

Παράδειγμα:

```
interface OrismosConstants {
  double pi=3.14;
  int syntelestis_FPA=25; }

class TestConstants implements OrismosConstants {
 public static final int CONST = 18;
 public static void main(String args[])  {
  System.out.println("Stathera pi tis diepafis= "+pi);
  System.out.println("Stathera syntelestis_FPA tis diepafis=
                     "+syntelestis_FPA);
  System.out.println("Stathera tis klasis= "+CONST);}}
```



Διεπαφές/Διασυνδέσεις *(Interfaces) (7/13)*

Υλοποίηση **μεθόδων της διεπαφής** *(κλασική χρήση)*, σε κλάση που υλοποιεί την διεπαφή.

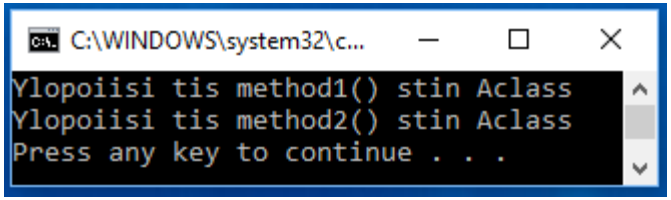
Παράδειγμα:

```
interface MyDemoInterface {
   public void method1();
   public void method2();}

class Aclass implements MyDemoInterface {

  public void method1() {
    System.out.println("Ylopoiisi tis method1() stin Aclass"); }
  public void method2() {
    System.out.println("Ylopoiisi tis method2() stin Aclass"); }

  public static void main(String arg[]){
      MyDemoInterface obj = new Aclass();
      obj. method1();
      obj. method2(); } }
```



Διεπαφές/Διασυνδέσεις *(Interfaces) (8/13)*

Κληρονομικότητα διεπαφής:

Όπως είπαμε, η διεπαφή μπορεί να **κληρονομεί** *(extends)* άλλη διεπαφή όπως μια κλάση κληρονομεί μια άλλη κλάση. Η child-διεπαφή κληρονομεί τις σταθερές και μεθόδους της parent-διεπαφής.

Προσοχή στην υλοποίηση των μεθόδων από την κληρονομικότητα των διεπαφών

Στο παρακάτω παράδειγμα, ορίζονται ***υποχρεωτικά*** και οι δύο μέθοδοι στην κλάση Demo, που ενώ υλοποιεί μόνο την διεπαφή Diepafi2 (...θα έπρεπε να ορίσει μόνο την method2()), πρέπει να ορίσει και την method1() **που *κληρονομεί από την Diepafi1***.

Παράδειγμα:

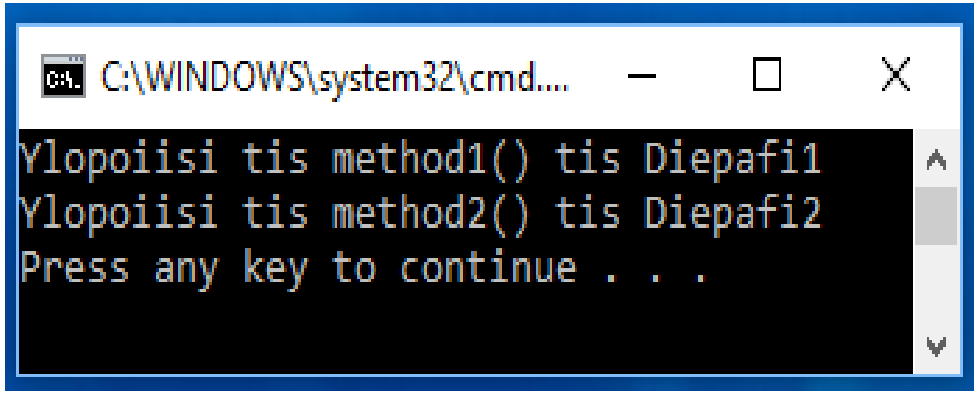
```
interface Diepafi1{
   public void method1();}

interface Diepafi2 extends Diepafi1 {
   public void method2();}
```

Διεπαφές/Διασυνδέσεις *(Interfaces)* *(9/13)*

```
class Demo implements Diepafi2{
 public void method1(){
  System.out.println("Ylopoiisi tis method1() tis
                      Diepafi1");}
 public void method2(){
 System.out.println("Ylopoiisi tis method2() tis
                      Diepafi2");}}

class TestInterface2 {
 public static void main(String args[]) {
    Demo obj = new Demo();
    obj. method1();
    obj. method2(); }}
```



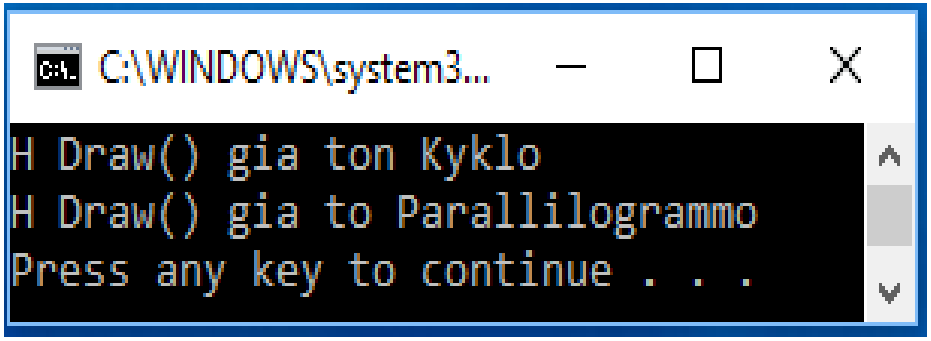
Διεπαφές/Διασυνδέσεις *(Interfaces) (10/13)*

Η διεπαφή Shape ορίζει την μέθοδο draw() που θα υλοποιηθεί στις δύο κλάσεις Circle και Rectangle που υλοποιούν την διεπαφή. Δες την κλάση ShapeConstruction που υλοποιεί το πρότυπο *(pattern)* ***Factory***.

```
interface Shape
 {public void draw();}
class Circle implements Shape {
  public void draw()
   {System.out.println("H Draw() gia ton Kyklo");}}
class Rectangle implements Shape {
  public void draw()
   {System.out.println("H Draw() gia to Parallilogrammo");}}
class ShapeConstruction {
  public Shape getShape(String s){
    if (s.equals("Circle")){return new Circle();}
    if (s.equals("Rectangle")){return new Rectangle();}
    return null; }}
```

Διεπαφές/Διασυνδέσεις *(Interfaces) (11/13)*

```
class TestInterfaces {
  public static void main(String args[]) {
   ShapeConstruction sc=new ShapeConstruction();
   Shape sh1=sc.getShape("Circle");//το sh1 θα οριστεί αντικείμενο circle
   sh1.draw();
   Shape sh2=sc.getShape("Rectangle");//το sh2 θα οριστεί αντ. rectangle
   sh2.draw();
  }
 }
```



Διεπαφές/Διασυνδέσεις *(Interfaces) (12/13)*

Η διεπαφή Emvadon ορίζει την μέθοδο *computeEmvadon()* που θα υλοποιηθεί Στις κλάσεις Rectangle και Triangle, υλοποιώντας την διεπαφή. Προσέξτε τις εντολές: **Emvadon emv;** και **emv = rect;**

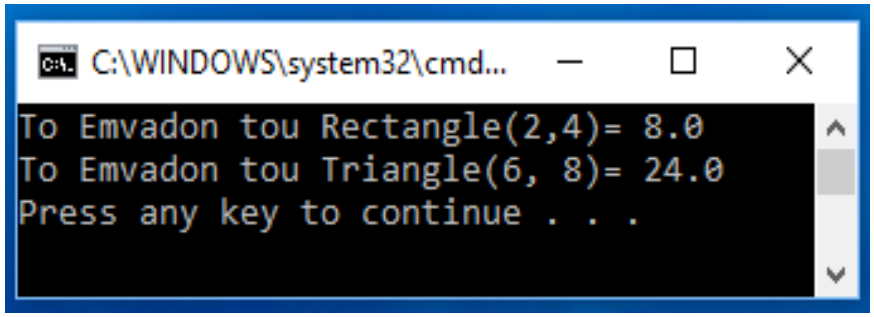
```
interface Emvadon
{ float computeEmvadon(float x, float y);}

class Rectangle implements Emvadon {
 public float computeEmvadon(float x, float y){
   return(x * y);}}

class Triangle implements Emvadon {
 public float computeEmvadon(float x, float y) {
   return(x * y/2);}}
```

Διεπαφές/Διασυνδέσεις *(Interfaces) (13/13)*

```
class InterfaceEmvadon {
 public static void main(String args[]){
  Rectangle rect = new Rectangle();
  Triangle tri = new Triangle();
  Emvadon emv; //μπορούμε αναφορά, όχι δημιουργία αντικειμένου με τον τελεστή new
  emv = rect;
  System.out.println("To Emvadon tou Rectangle(2,4)= "+
                     emv.computeEmvadon(2,4));
  emv = tri;
  System.out.println("To Emvadon tou Triangle(6, 8)= "+
                     emv.computeEmvadon(6,8));}}
```



Ξαναγράφοντας Διεπαφές *(Πρόβλημα επεκτασιμότητας της διεπαφής)*

Σκεφτείτε την περίπτωση όπου έχετε αναπτύξει μια διεπαφή π.χ.

```
public interface Transactions {
   void debit(String n, double b, double p);
   void credit(String n, double b, double p);}
```

- Υποθέστε ότι θέλετε να χρησιμοποιήσετε και άλλο είδος δοσοληψίας π.χ.
  void metafora(String n, double b, double p);
- Αν την προσθέσουμε στην ήδη υπάρχουσα διεπαφή, τότε οι κλάσεις που υλοποιούν την διεπαφή θα παρουσιάσουν πρόβλημα (λάθος), γιατί δεν υλοποιούν όλες τις μεθόδους.

Λύση του προβλήματος

Δημιουργία νέας διεπαφής που θα κληρονομεί *(extends)* από την transactions και θα ορίζει την νέα λειτουργικότητα, π.χ.:

```
public interface MetaforaPosou extends Trasanctions {
   void metafora(String n, double b, double p); }
```

Άσκηση – 1 *(Πολλαπλή κληρονομικότητα)*

Να γραφεί πρόγραμμα που χειρίζεται τις δοσοληψίες ενός λογαριασμού. Το πρόγραμμα ορίζει την κλάση **Account** με πεδία (1) ***eponymia***, *String,* (2) ***balance***, *double* και (3) ***poso_kinisis,*** *double* και επιπλέον πλήρη δομητή, getters() και την toString(). Για την εκτέλεση των δοσοληψιών/κινήσεων το πρόγραμμα ορίζει τις διεπαφές:

(1) ***iDebit*** : με την μέθοδο **void debit(String name, double balance, double poso);**

(2) ***iCredit*** : με την μέθοδο **void credit(String name, double balance, double poso);**

(1) ***iMetafora*** : με την μέθοδο **void metafora(String name, double balance, double poso);**

και τις κλάσεις:

(1) **Debit,** που κληρονομεί *(extends)* την Account και υλοποιεί *(implements)* την *iDebit*. Η κλάση περιέχει ένα πλήρη δομητή, και υλοποιεί την μέθοδο debit().

(2) **Credit**, που κληρονομεί *(extends)* την Account και υλοποιεί *(implements)* την *iCredit*. Η κλάση περιέχει ένα πλήρη δομητή, και υλοποιεί την μέθοδο credit().

(3) **Metafora**, που κληρονομεί *(extends)* την Account και υλοποιεί *(implements)* την *iMetafora*. Η κλάση περιέχει πλήρη δομητή, και υλοποιεί την μέθοδο metafora(), που εμφανίζει εκτός από το ποσό μεταφοράς και το νέο υπόλοιπο (balance).

Στην κλάση **AccountTest** *(με την main),* θα ορίσετε ένα πίνακα N – αντικειμένων Account, τύπου Debit, Credit και Metafora, και θα εκτελέσετε τις δοσοληψίες. Χρησιμοποιήστε την **instanceof** (π.χ. **If (pinakas[i] instanceof Debit)** {….}).

Σύγκριση χαρακτηριστικών απλών κλάσεων, Αφηρημένων κλάσεων και Διεπαφών

Χαρακτηριστικό	Απλή κλάση	Αφηρημένη κλάση	Διεπαφή
Μπορούν να δημιουργηθούν αντικείμενα;	Ναι	Όχι	Όχι
Μπορούν να οριστούν πεδία και συγκεκριμένες μέθοδοι;	Ναι	Ναι	Όχι
Μπορούν να οριστούν σταθερές;	Ναι	Ναι	Ναι
Μια κλάση μπορεί να επεκτείνει (extends):	0 ή 1	0 ή 1	0
Μια κλάση μπορεί να υλοποιήσει (implements):	0	0	1..*
Μπορούν να οριστούν αφηρημένες μέθοδοι;	Όχι	Ναι	Ναι
Μπορούν να δηλωθούν μεταβλητές αυτών των τύπων;	Ναι	Ναι	Ναι

Προκαθορισμένες Διεπαφές του API της Java

Comparable *(Interface java.lang.Comperable)*

Για την σύγκριση αντικειμένων

Η διεπαφή περιέχει μια μέθοδο την **compareTo()** που συγκρίνει το αντικείμενο που καλεί την μέθοδο με το αντικείμενο που περνάει σαν παράμετρος στην μέθοδο. Η μέθοδος επιστρέφει ένα αρνητικό ακέραιο αριθμό, ή το 0, ή ένα θετικό ακέραιο αριθμό ανάλογα αν το καλών αντικείμενο είναι μικρότερο, ίσο ή μεγαλύτερο από το αντικείμενο παράμετρο της μεθόδου.

```
public interface Comparable {
   public int compareTo(Object o);
}
```

- Χρησιμοποιείται για να συγκρίνουμε αντικείμενα μιας συγκεκριμένης δομής με ***τελικό σκοπό την ταξινόμηση των αντικειμένων***.

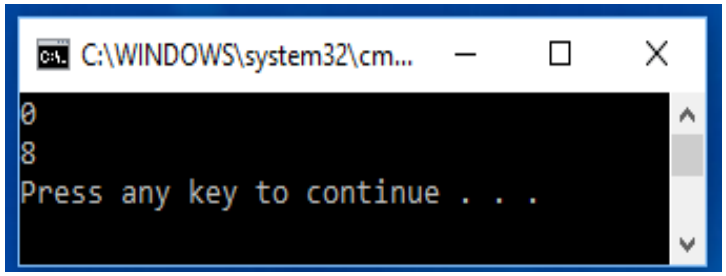
Comparable – CompareTo()*(1/4)*

Χρήση της CompareTo() για έλεγχο – σύγκριση Stings

Στη σύγκριση str1 με str2 επιστρέφει 0 (ίσα)

```
class Test1 {
   public static void main(String args[]) {
      String str1 = "I love Java";
      String str2 = new String("I love Java");
      String str3 = new String("And I love programming");

      int result = str1.compareTo(str2);
      System.out.println(result);
      result = str2.compareTo(str3);
      System.out.println(result);
}}
```



Comparable – CompareTo() *(2/4)*

Για να ταξινομήσουμε αντικείμενα (π.χ. πίνακα αντικειμένων) ως προς κάποιο χαρακτηριστικό τους, μπορούμε να χρησιμοποιήσουμε την **Array.sort(),** αφού όμως προηγουμένως έχουμε δηλώσει ότι τα αντικείμενα υλοποιούν *(implements)* την διεπαφή **Comparable** και υπερβαίνουν *(override)* την μέθοδο **ComareTo()**. Διαφορετικά η Array.sort() δεν θα δουλέψει (θα πάρουμε μήνυμα λάθους) στην ταξινόμηση αντικειμένων. Το παρακάτω παράδειγμα ταξινομεί τα βότανα ως προς την ποσότητα.

```
import java.util.Arrays;
class Votana implements Comparable<Votana>{
 private String VotanoName;
 private String VotanoDesc;
 private int quantity;
 public Votana(String VotanoName, String VotanoDesc, int quantity){
  super();
  this.VotanoName = VotanoName;
  this.VotanoDesc = VotanoDesc;
  this.quantity = quantity; }
```

Comparable – CompareTo() *(3/4)*

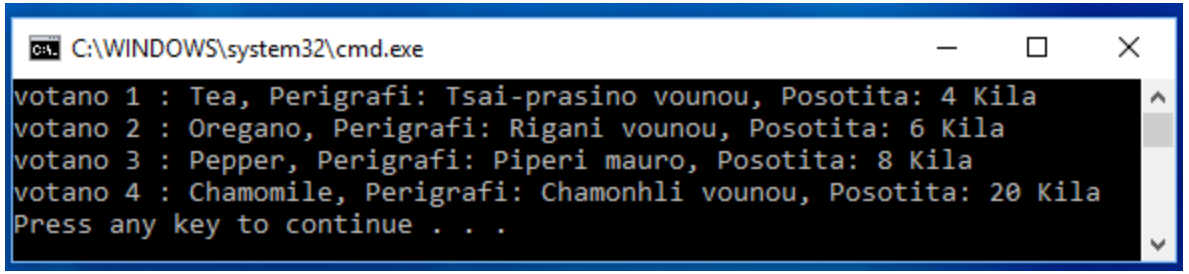
```
public String getVotanoName()
 {return VotanoName;}
public void setVotanoName(String VotanoName)
 {this.VotanoName = VotanoName;}
public String getVotanoDesc()
 {return VotanoDesc;}
public void setVotanoDesc(String VotanoDesc)
 {this.VotanoDesc = VotanoDesc;}
public int getQuantity()
 {return quantity;}
public void setQuantity(int quantity)
 {this.quantity = quantity;}

public int compareTo(Votana compareVotana) {
 int compareQuantity = ((Votana)compareVotana).getQuantity();
 //ascending order
 return this.quantity - compareQuantity;
 //gia descending order: return compareQuantity - this.quantity;
}}
```

Comparable – CompareTo() *(4/4)*

```
class SortVotanaObjects{
 public static void main(String args[]){
   Votana[] votano = new Votana[4];
   Votana tea = new Votana("Tea", "Tsai-prasino vounou ",4);
   Votana chamomile = new Votana("Chamomile", "Chamonhli vounou",20);
   Votana oregano = new Votana("Oregano", "Rigani vounou",6);
   Votana pepper = new Votana("Pepper", "Piperi mauro",8);
   votano[0]=tea;
   votano[1]=chamomile;
   votano[2]=oregano;
   votano[3]=pepper;

   Arrays.sort(votano);
   int i=0;
   for(Votana temp: votano){
    System.out.println("votano " + ++i + " : " + temp.getVotanoName()
         +", Perigrafi: " + temp.getVotanoDesc()+", Posotita: " +
            temp.getQuantity()+" Kila");}
}}
```



Cloneable *(1/4)*

Cloneable :

Για να δημιουργήσουμε ένα **ακριβές αντίγραφο** *(copy)* ενός αντικειμένου με την χρήση της μεθόδου **clone()** της κλάσης **Object,** υλοποιούμε την διεπαφή Clonable. Κάνουμε υπέρβαση *(override)* της μεθόδου clone() και καλό είναι να χρησιμοποιούμε την:

throws CloneNotSupportedException, στην υπογραφή της μεθόδου.

Παράδειγμα:

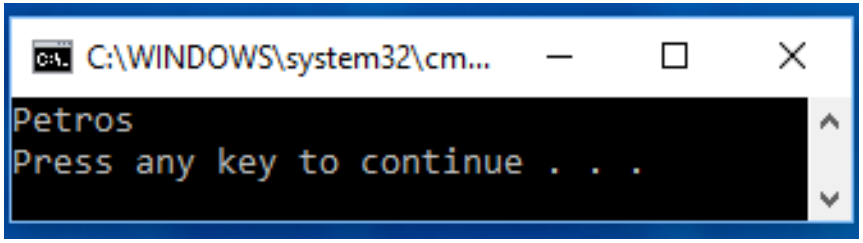
```
class CloneableName implements Cloneable {
   private String Aname;

   public CloneableName(String n)
    {this.Aname = n;}

   public String getName()
    {return Aname;}
   // Overriding clone() method tis klasis Object
   public Object clone()throws CloneNotSupportedException{
       return (CloneableName)super.clone(); }
```

Cloneable *(2/4)*

```
public static void main(String[] args) {
    CloneableName obj1 = new CloneableName("Petros");
       try {
          CloneableName obj2 = (CloneableName)obj1.clone();
          System.out.println(obj2.getName());
       } catch (CloneNotSupportedException e) {
             e.printStackTrace();
        }
}}
```



- Προσέξτε την χρήση της try – catch. Η εξαίρεση CloneNotSupportedException λαμβάνεται όταν εκτελέσουμε την μέθοδο clone().

Cloneable *(3/4)*

Παράδειγμα:

Η κλήση της **clone()** μπορεί να γίνει και μέσα από άλλη μέθοδο.

```
class CopyObject implements Cloneable {
 int x;
 double y;
 // H methodos kalei tin clone() tis Object
 CopyObject mycloneObject()
 {
  try {
   // klisi tis clone
   return (CopyObject)super.clone();
  } catch(CloneNotSupportedException e) {
      System.out.println("Lathos stin antigrafi!!");
      return null; }
}}
```

Cloneable *(4/4)*

```
class CloneDemo1 {
 public static void main(String args[]) {
   CopyObject z1 = new CopyObject();
   CopyObject z2;
   z1.x = 9;
   z1.y = 7.40;
   z2 = z1.mycloneObject(); // clone z1
   System.out.println("z1: " + z1.x + " " + z1.y);
   System.out.println("z2: " + z2.x + " " + z2.y);
 }}
```