Generics

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

Generics *(1/15)*

Generics (Παραμετρικός Πολυμορφισμός)

Παραμετρικός Πολυμορφισμός:

Η δυνατότητα ορισμού **κλάσεων** και **μεθόδων** με παραμέτρους τύπους δεδομένων **άλλων κλάσεων.** Κλάσεις και μέθοδοι διαχειρίζονται αντικείμενα **διαφορετικών τύπων** με την ίδια υλοποίηση. Τέτοιες κλάσεις και μέθοδοι λέγονται **Generic**.

Πλεονεκτήματα-Κίνητρο:

Γράφουμε ευέλικτο γενικό κώδικα χωρίς να θυσιάζουμε την ασφάλεια των τύπων δεδομένων.

- Χρησιμοποιούνται στις συλλογές και στη Java Reflection
- Πρώτη χρήση στη γλώσσα ML το 1976
- Τώρα σε πολλές άλλες γλώσσες(Haskell, Java, C#, Delphi)
- Στην C++ τα templates είναι παρόμοια αλλά δεν είναι το ίδιο ευέλικτα και δεν υλοποιούν πολλές λειτουργίες όπως τα generics.

Generics *(2/15)*

- Στη Java από την έκδοση 5.0 (v1.5 - 2004) και μετά.
- Παρέχουν ασφάλεια κατά την **μεταγλώττιση** (ισχυρός έλεγχος των τύπων δεδομένων).
- Επιτρέπεται η **συλλογή αντικειμένων σε μία οντότητα**.

❖ ***Δεν απαιτούνται πλέον οι μετατροπές τύπων (casting)***

- *χωρίς generics:* List list = new ArrayList();
  list.add("Java"); String s=(String)list.get(0);

- *με **generics**:* List<String> list = new ArrayList<String>();
  list.add("Java"); String s=list.get(0);

- Επιτρέπει την δημιουργία γενικών – ευέλικτων αλγορίθμων που λειτουργούν σε **συλλογές διαφορετικών τύπων δεδομένων**.
- **Δουλεύουμε πλέον με generics στα Collections**

Generics *(3/15)*

Generic ή παραμετρική κλάση

- Ορίζεται όπως και η απλή κλάση με την διαφορά ότι μετά το όνομα της πρέπει να ακολουθεί τουλάχιστον μια τυπική παράμετρος ανάμεσα στα σύμβολα <>.

```
class name<T1, T2, ..., Tn>
 {
   /* ... */
 }
```

Παράδειγμα:

```
class GenericTest1<T>
      {……….}
```

- Ο ορισμός των αντικειμένων ακολουθεί τους κανόνες των generics. **Δεν επιτρέπεται η χρήση του new για την δημιουργία αντικειμένου Τ** (λάθος new T()).
- Δεν επιτρέπεται η χρήση **static T – πεδίο** στα generics (λάθος:

```
public class GenericsExample<T> {
    private static T member; //lathos, den epitrepetai }
```

- Δεν επιτρέπεται η **δημιουργία κλάσης - exception**

Generics *(4/15)*

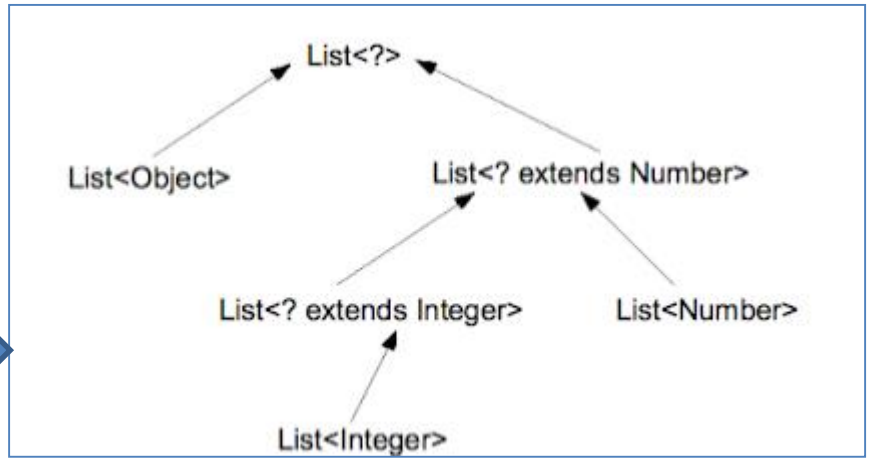
Ορισμοί – Συμβολισμοί

Generic Type	Περιγραφή
<T extends Comparable>	Περιορισμένες *(bounded)* Type Parameters
<T super Comparable>	Περιορισμένες *(bounded)* Type Parameters
Set<?>	Μη περιορισμένος wildcard τύπος
<? extends T>	Περιορισμένος wildcard τύπος
<? Super T>	Περιορισμένος wildcard τύπος
<T extends Comparable<T>>	Αναδρομικός περιορισμένος τύπος

T = για τύπους

N = για αριθμούς

Ταξινόμηση των generic List – τύπων



Generics *(5/15)*

Ασφάλεια των τύπων *(type-safety)*

- Με τη χρήση generic τύπων, ο compiler πιστοποιεί την κλάση και αν διαπιστώσει ασυμβατότητα θα δώσει λάθος.
- Π.χ. αν ορίσουμε ένα ArrayList που θα περιέχει αντικείμενα μετοχές, δεν μπορεί να χρησιμοποιηθεί για άλλα αντικείμενα, π.χ. νομίσματα

  ArrayList<Metoxes> metoxesList = new ArrayList<Metoxes>();

  (το *<Metoxes> λέει στον compiler ότι το ArrayList θα περιέχει μόνο μετοχές)*

Generics wild cards

- Δύο τύποι: **Bounded - Unbounded**

Bounded

- Δύο τρόποι γραφής για άνω και κάτω όριο (π.χ. **<? extends T>** και <? super T>)

Unbounded (ο συμβολισμός <?>), μπορεί να δεχτεί οποιονδήποτε τύπο

Generics *(6/15)*

<?>

- Συμβολίζει ένα άγνωστο τύπο (ή οποιονδήποτε τύπο). Π.χ. αν θέλουμε ένα ArrayList για διάφορους τύπους, τότε το ορίζουμε με τον ορισμό ArrayList<?>

```
ArrayList<?> unknownList = new ArrayList<Number>();
unknownList = new ArrayList<Float>();
```

<? extends T>

- Επιτρέπει όλους τους τύπους **T** ή τις **υποκλάσεις της T** *(extends)*

```
ArrayList<? extends Number> numberList = new ArrayList<Number>();
numberList = new ArrayList<Integer>();
numberList = new ArrayList<Float>();
```

<? super T>

- Το αντίστροφο του προηγούμενου ορισμού, επιτρέπει μόνο τύπους T και super class του T (π.χ. Integer και Number).

```
ArrayList<? super Integer> numberList = new ArrayList<Number>();
numberList = new ArrayList<Integer>();
numberList = new ArrayList<Float>(); //compilation error
```

Generics *(7/15)*

- Παραδείγματα ορισμών με wild cards:

```
Collection<?> coll = new ArrayList<String>();
List<? extends Number> list = new ArrayList<Long>();
Pair<String, ?> pair = new Pair<String, Integer>();
```

- Παραδείγματα ορισμών με unbounded wild cards:

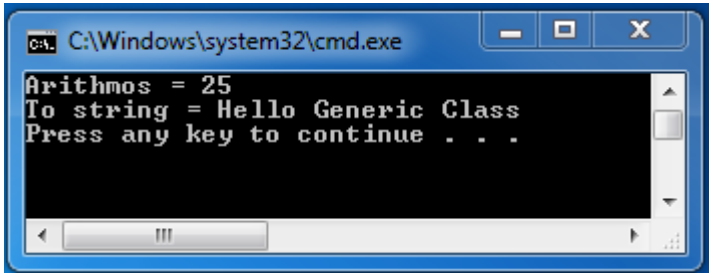
```
ArrayList<?>  list = new ArrayList<Long>();
ArrayList<?>  list = new ArrayList<String>();
ArrayList<?>  list = new ArrayList<Employee>();
```

!!! (θα τους δούμε αναλυτικά και με παραδείγματα στις παρακάτω διαφάνειες)

Generics *(8/15)*

Ορισμός μιας generic κλάσης με 2 διαφορετικά αντικείμενα (**wrapper type** και **reference type**). Η add() αρχικοποιεί το αντικείμενο και η get() το επιστρέφει στο κυρίως πρόγραμμα.

```
class GenericTest1<T> {
  private T t;
  public void add(T t) {this.t = t;}
  public T get() {return t;}

  public static void main(String[] args) {
     GenericTest1<Integer> i1 = new GenericTest1<Integer>();
     GenericTest1<String> s1 = new GenericTest1<String>();

     i1.add(new Integer(25));
     s1.add(new String("Hello Generic Class"));


     System.out.println("Arithmos = " + i1.get());
     System.out.println("To string = " + s1.get());
  } }
```



Generics *(9/15)*

Παράδειγμα με διαφορετικά ζεύγη τύπων:

```
interface Pair<K, V> { public K getKey();  public V getValue(); }

class OrderedPair<K, V> implements Pair<K, V> {
    private K key; private V value;

    OrderedPair(K key, V value) {
     this.key = key;
     this.value = value;
     }

     public K getKey() { return key; }
     public V getValue() { return value; }
  }
```

Generics *(10/15)*

```
class NewClass
{
  public static void main(String[] args)
  {
   Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
   Pair<String, String>  p2 = new OrderedPair<String, String>("hello", "world");
   System.out.println(" "+p1.getKey()+" " +p1.getValue());
   System.out.println(" "+p2.getKey()+" " +p2.getValue());
 }
}
```

```
run:
 Even 8
 hello world
BUILD SUCCESSFUL (total time: 0 seconds)
```

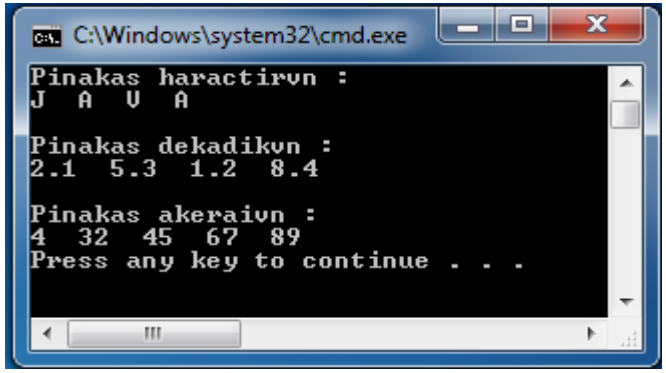
Generics *(11/15)*

Generic Methods

- Μια generic μέθοδος μπορεί να κληθεί με παραμέτρους διαφορετικών τύπων (αναφορές και όχι βασικοί τύποι).
- Στην υπογραφή της μεθόδου γράφεται ο τύπος του επιστρεφόμενου αποτελέσματος μέσα σε <>
- Η μέθοδος μπορεί να δεχτεί μια ή περισσότερες παραμέτρους χωρισμένες με το σύμβολο (,) κόμμα.
- Στο παρακάτω **παράδειγμα** με την ίδια μέθοδο θα εμφανίσουμε τρεις-διαφορετικού τύπου - πίνακες

Generics *(12/15)*

```
class GenericMethod {
   public static < A > void printArray( A[ ] inputArray ) {
      for ( A element : inputArray )  {
         System.out.print(element + "  "); } System.out.println(); }

   public static void main( String args[] )  {
      Character[] charArray = { 'J', 'A', 'V', 'A'};
      Double[] doubleArray = { 2.1, 5.3, 1.2, 8.4 };
      Integer[] intArray = { 4, 32, 45, 67, 89 };
      System.out.println( "Pinakas haractirvn : " );
      printArray( charArray );
      System.out.println();
      System.out.println( "Pinakas dekadikvn : " );
      printArray( doubleArray );
      System.out.println();
      System.out.println( "Pinakas akeraivn : " );
      printArray( intArray  );  } }
```



Generics *(13/15)*

Παράδειγμα χρήσης ζεύγους διαφορετικών τύπων

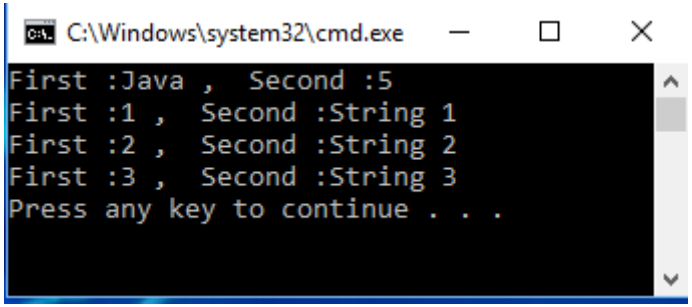
```
import java.util.*;
class Pair<T1, T2>{
  private final T1 first;
  private final T2 second;

  //1os-domitis arxikopoiei tous 2 typous
  public Pair(T1 first, T2 second){
    this.first = first;
    this.second = second; }

 //2os-domitis arxikopoiei zeugos ton 2-typon, to zeugos np
  public Pair(Pair<T1, T2> np){
    this.first = np.getFirst();
    this.second = np.getSecond(); }
```

Generics *(14/15)*

```
public T1 getFirst() {
  return this.first;}

public T2 getSecond() {
  return this.second; }

public String toString(){
  return  "First :" + first.toString() + " , "  +  " Second :" + second.toString(); }}

class TestPairs {
  public static void main(String[] args) {
   Pair<String, Integer> p1 = new Pair<String, Integer>("Java", 5);
   System.out.println(p1);
  //
   ArrayList<Integer> v1 = new ArrayList<Integer>();
   for (int x = 1; x <= 3; x++) v1.add(new Integer(x));
 //
```

Generics *(15/15)*

```
//
   ArrayList<String> v2 = new ArrayList<String>();
   v2.add(new String("String 1"));
   v2.add(new String("String 2"));
   v2.add(new String("String 3"));
 //
  ArrayList<Pair<Integer, String>> v3 = new ArrayList<Pair<Integer, String>>();
   for (int i = 0; i <= 2; i++)
   v3.add(new Pair<Integer, String>(v1.get(i), v2.get(i)));
   for (Pair<Integer, String> p : v3) System.out.println(p);
   }
}
```



Περιορισμένες *(bounded)* Type Parameters *(1/10)*

- Μπορούμε να **περιορίσουμε τους τύπους** που χρησιμοποιούνται ως παράμετροι. Δηλ. να δέχονται μόνο ***συγκεκριμένους τύπους παραμέτρων.*** Χρησιμοποιούμε την δεσμευμένη λέξη **extends**

- Για παράδειγμα σε μία μέθοδο που δέχεται αριθμούς σαν τυπική-παράμετρο, μπορούμε να περιορίσουμε την μέθοδο να δέχεται μόνο ***Number – τύπους***.

- Δήλωση ενός *bounded type parameter* :

  <T extends *superClassName*>

  π.χ. ***class Periorismeni<T extends A>***

  (χρήση μόνο παραμέτρων τύπου Α)

- Αν ορίσουμε **<T extends Number>,** τότε το **άνω όριο** είναι όλοι οι αριθμοί (δηλ. wrapper types αριθμοί, δηλ. Integer, Double, κλπ.)

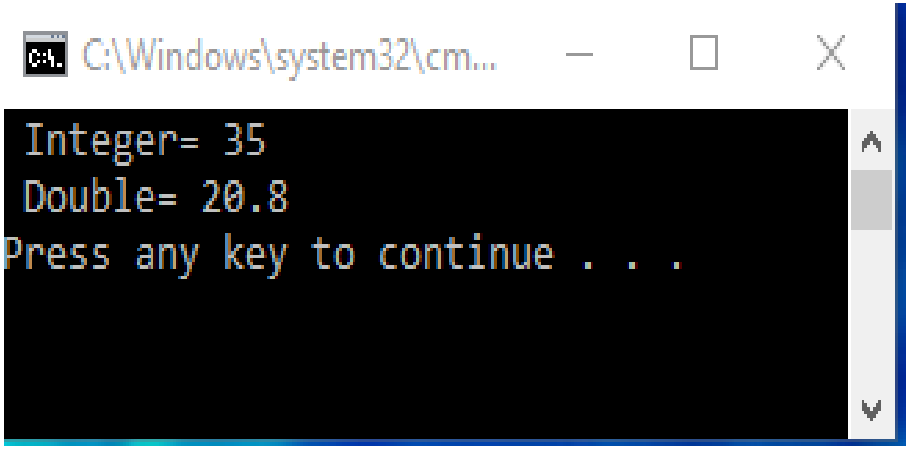
Περιορισμένες *(bounded)* Type Parameters *(2/10)*

Παράδειγμα 1ο:

```
class BoundedEx<T extends Number> {
   private T num;
   void setNum(T num) { this.num = num; }
   T  getNum()  { return num;     }}

class TestBoundedEx {
  public static void main(String[] args) {
  BoundedEx<Integer> x = new BoundedEx<>();
  x.setNum( Integer.valueOf(35));
  // epistrefei ton ar. san Integer
  Integer i = x.getNum();
  System.out.println("Integer= "+i);
```

Περιορισμένες *(bounded)* Type Parameters *(3/10)*

```
//τύπου double

BoundedEx<Double> y = new BoundedEx<>();
    y.setNum( Double.valueOf(20.8));

  // epistrefei ton ar. san double
    Double z = y.getNum();
    System.out.println(" Double= "+z);
} }
```



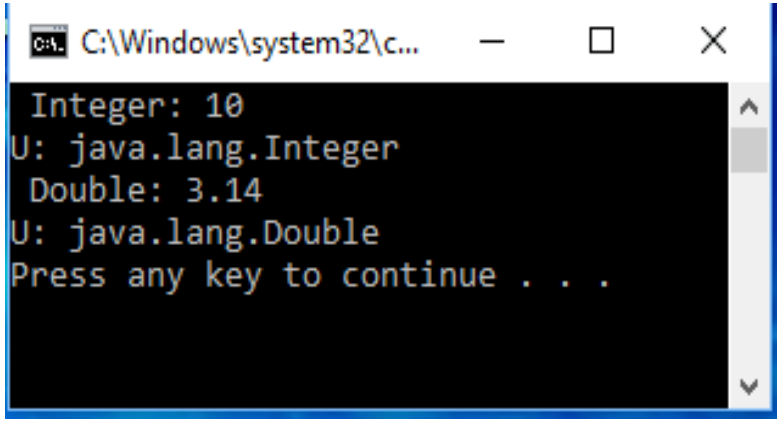
Περιορισμένες *(bounded)* Type Parameters *(4/10)*

Παραλλαγή 1ου Παραδείγματος: Χρήση της **getClass().getName()** *(java reflection)* για την εύρεση της κλάσης του αντικειμένου.

```
class Box1<T extends Number>  {
   private T t;

  public void set(T t) {this.t = t;}
  public T get() {return t;}

  public <U extends Number> void inspect(U u){
        System.out.println("U: " + u.getClass().getName());
}
```

Περιορισμένες *(bounded)* Type Parameters *(5/10)*

```
public static void main(String[] args) {
   Box1<Integer> i = new Box1<Integer>();
   i.set(new Integer(10));
   System.out.println(" Integer: "+i.get());
   i.inspect(10);
 //
   Box1<Double> d = new Box1<Double>();
   d.set(new Double(3.14));
   System.out.println(" Double: "+d.get());
   d.inspect(3.14);
} }
```



Περιορισμένες *(bounded)* Type Parameters *(6/10)*

Παράδειγμα 2ο – Bounded στην Υπερκλάση και στις υποκλάσεις:

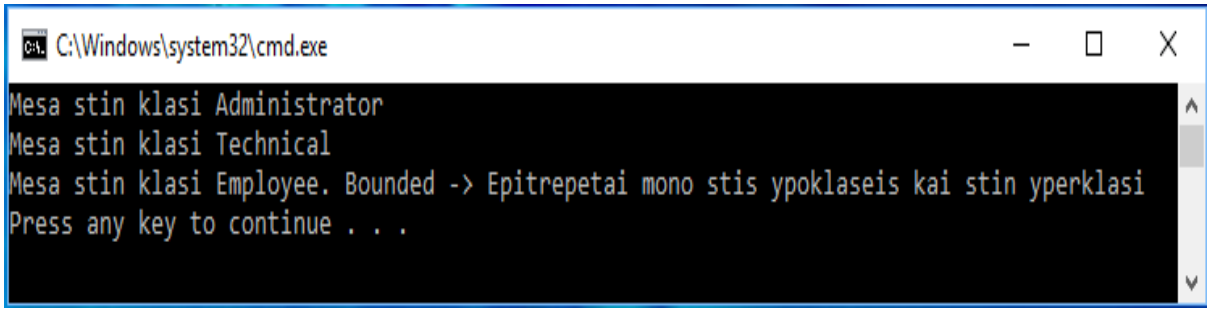
```
class Bound<T extends Employee> {
   private T objRef;
   public Bound(T obj){this.objRef = obj;}
   public void doRunTest(){this.objRef.displayClass();}}

class Employee {
  public void displayClass()  {
      System.out.println("Mesa stin klasi Employee. Bounded -> Epitrepetai mono stis
                          ypoklaseis kai stin yperklasi");
} }
```

Περιορισμένες *(bounded)* Type Parameters *(7/10)*

```
class Technical extends Employee  {
  public void displayClass() {System.out.println("Mesa stin klasi Technical"); } }

class Administrator extends Employee {
  public void displayClass(){System.out.println("Mesa stin klasi Administrator"); }}

class BoundedClass1 {
 public static void main(String a[])
   // pername to antikeimeno typou Administrator san Bounded parametro
   Bound<Administrator> adm = new Bound<Administrator>(new Administrator());
   adm.doRunTest();
  // pername to antikeimeno typou Technical san Bounded parametro
   Bound<Technical> tec = new Bound<Technical>(new Technical());
   tec.doRunTest();
```

Περιορισμένες *(bounded)* Type Parameters *(8/10)*

```
// pername to antikeimeno typou Employee san Bounded parametro
Bound<Employee> emp = new Bound<Employee>(new Employee());
emp.doRunTest();
// pername antikeimeno typou String san Bounded parametron. Lathos!!!
Bound<String> str = new Bound<String>(new String());
str.doRunTest();  } }
```




```
C:\0-OOP_sfetsos\GENERICS\BoundedClass1.java:35: error: type argument String is not within bounds of type-variable T
        Bound<String> str = new Bound<String>(new String());
                ^
  where T is a type-variable:
    T extends Employee declared in class Bound
C:\0-OOP_sfetsos\GENERICS\BoundedClass1.java:35: error: type argument String is not within bounds of type-variable T
        Bound<String> str = new Bound<String>(new String());
                ^
  where T is a type-variable:
    T extends Employee declared in class Bound
2 errors

Tool completed with exit code 1
```

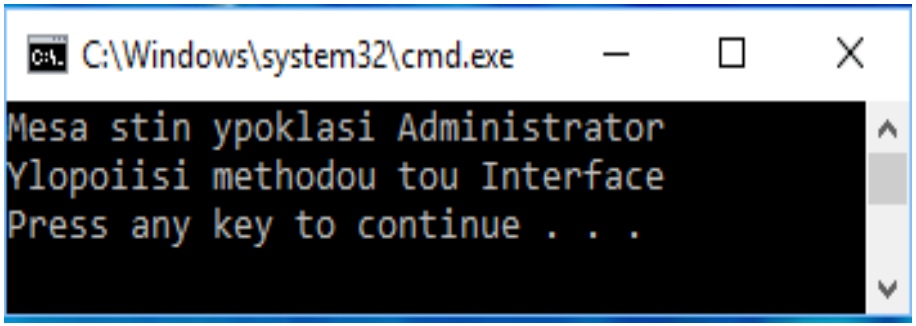
Περιορισμένες *(bounded)* Type Parameters *(9/10)*

Παράδειγμα 3ο – Bounded στην Υπερκλάση και σε Interface (η πρώτη παραμ. Κλάση και τα άλλα interfaces με το σύμβολο &)

```
class Bound<T extends Employee & iMisthos> {
  private T objRef;
  public Bound(T obj){this.objRef = obj;}
  public void doRunTest(){this.objRef.displayClass(); }
  public void displayClass(){System.out.println("Ylopoiisi methodou tou
                                        Interface");}}

interface iMisthos  { public void displayClass(); }

class Administrator extends Employee implements iMisthos {
  public void displayClass()
     { System.out.println("Mesa stin ypoklasi Administrator"); }}
```

Περιορισμένες *(bounded)* Type Parameters *(10/10)*

```
class BoundedClass2 {
    public static void main(String a[]) {

   //antikeimeno ypoklasis san type parametros alla kai ylopoiisi methodou interface
   Bound<Administrator> emp = new Bound<Administrator>(new Administrator());
   emp.doRunTest();
   emp.displayClass();
} }
```



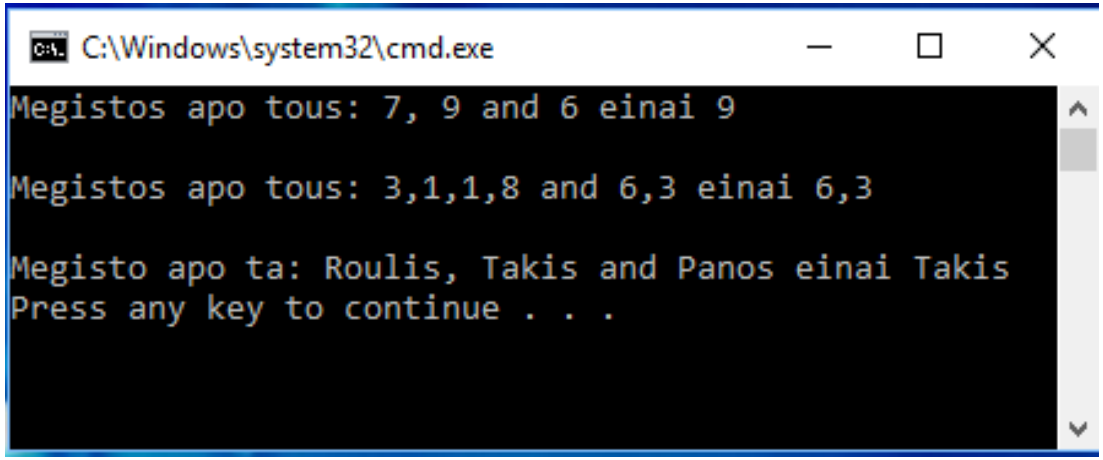
Περισσότερα Παραδείγματα

Παράδειγμα-1: Η χρήση της **extend** *(κλάσεις)* σαν implements των διεπαφών *(interfaces)*. Στο παράδειγμα η generic μέθοδος **maximum** συγκρίνει 3-συγκρίσιμα *(comperable)* αντικείμενα και επιστρέφει το μεγαλύτερο από αυτά.

```
public class Maximum_Arithmitikon_Antikeimenon {

  public static <T extends Comparable<T>> T maximum(T t1, T t2, T t3) {
    T max = t1;   // t1 einai to megalytero

    if(t2.compareTo(max) > 0) {
      max = t2;   // t2 einai to megalytero tora
    }

    if(t3.compareTo(max) > 0) {
      max = t3;   // t3 einai to megalytero tora
    }
    return max;   // epistrefei to magalytero antikeimeno
  }
```

Παραδείγματα

```
public static void main(String args[]) {
    System.out.printf("Megistos apo tous: %d, %d and %d einai %d\n\n",
                      7, 9, 6, maximum( 7, 9, 6 ));

    System.out.printf("Megistos apo tous: %.1f,%.1f and %.1f einai %.1f\n\n",
                      3.14, 1.8, 6.28, maximum( 3.14, 1.8, 6.28 ));

    System.out.printf("Megisto apo ta: %s, %s and %s einai %s\n","Roulis",
                      "Takis", "Panos", maximum("Roulis", "Takis", "Panos"));
} }
```



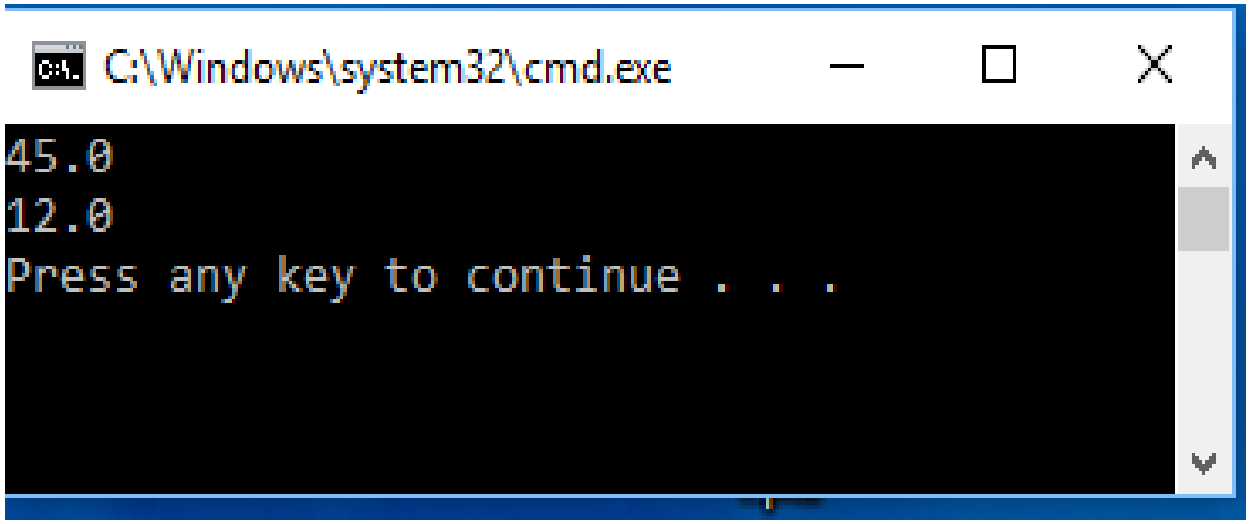
Παραδείγματα

Παράδειγμα-2 (Upper bounded wildcards): Η χρήση της **<? extends Numbers>** επιτρέπει την χρήση όλων των αριθμ. τύπων, εδώ για την εύρεση της sum() διαφορετικών αριθμ. Τύπων.

```
import java.util.*;
public class GenericsExample<T>
{
   public static void main(String[] args)
   {
      //List of Integers
      List<Integer> ints = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);
      System.out.println(sum(ints));

      //List of Doubles
      List<Double> doubles = Arrays.asList(1.5, 2.5, 3.5, 4.5);
      System.out.println(sum(doubles));}
```

Παραδείγματα

```
public static Number sum (List<? extends Number> numbers){
    double s = 0.0;
    for (Number n : numbers)
      s += n.doubleValue();
    return s;
  }
}
```

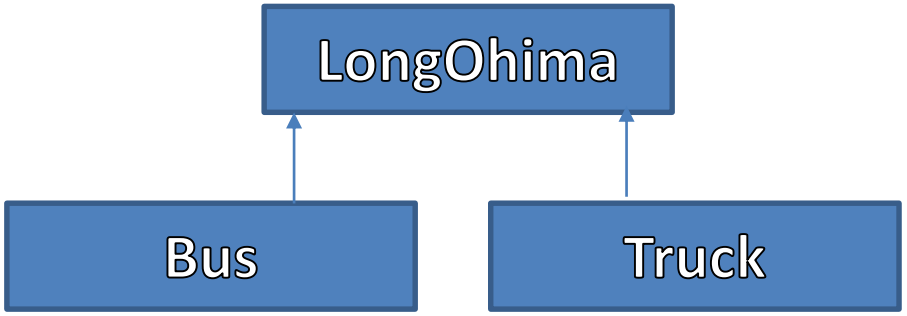


Παραδείγματα

Παράδειγμα – 3: Κληρονομικότητα:

- Σε μία κλάση την **BoundedDemo<T extends LongOhima>** θα χειριστούμε αντικείμενα των δύο υποκλάσεων (Bus και Truck).

- Η παράμετρος T (bounded) μπορεί να αντικατασταθεί μόνο από αντικείμενα τύπου LongOhima (υπερκλάση) ή αντικείμενα υποκλάσεων της (Bus και Truck)



Παραδείγματα

```
abstract class LongOhima {
    String name;
    public abstract void start(); }

class Bus extends LongOhima {
    Bus(String name) {this.name = name;}
    @Override
    public void start() {System.out.println("To leoforeio xekina");}}

class Truck extends LongOhima {
    Truck(String name) {this.name = name;}

    @Override
    public void start() {System.out.println("To fortigo xekina");}}
```

Παραδείγματα

```
class BoundedDemo<T extends LongOhima> {
    T var;
    BoundedDemo(T v) {var = v;}

    public void vehicleName() {System.out.println("To onoma toy ohimatos:
                                        "+var.name);}

    public static void main(String args[])  {
        Bus b = new Bus("Volvo");
        Truck t = new Truck("Mercedes-Benz");
        BoundedDemo<Bus> bus = new BoundedDemo<Bus>(b);
        BoundedDemo<Truck> truck = new BoundedDemo<Truck>(t);

        bus.vehicleName();
        truck.vehicleName();
}}
```