



BITS Pilani
Pilani Campus

Object Oriented Programming CS F213

J. Jennifer Ranjani

email: jennifer.ranjani@pilani.bits-pilani.ac.in

Chamber: 6121 B, NAB

Consultation: Appointment by e-mail



**Query asked during the
previous class**

Type name in Generics

The type name can be named according to programmer's convenience. But the common convention is

T - Type

E - Element

K - Key

N - Number

V – Value

**Note: Let there be an interface T;
class Identity<T extends T> cannot be done.**



Coming back to Generics

Review Question



```
ArrayList al = new ArrayList();  
al.add("Sachin");  
al.add("Rahul");  
al.add(10);
```

```
String s[] = new String[3];  
for(int i=0;i<3;i++)  
s[i] = (String)al.get(i);
```

```
System.out.println(al);  
System.out.println(Arrays.toString(s));
```

Find the output

- a. Compilation Error
- b. Runtime Error
- c. [Sachin, Rahul, 10]
[Sachin, Rahul, 10]

Note:

No compilation error because add(Object o) method in the ArrayList class

Runtime Error because integer object is type case to String

Solution:

Generics

Array List /Generics - Review



```
ArrayList<String> al = new ArrayList<String>();  
al.add("Sachin");  
al.add("Rahul");  
al.add("10");
```

```
String s[] = new String[3];  
for(int i=0;i<3;i++)  
s[i] =al.get(i);
```

```
System.out.println(al);  
System.out.println(Arrays.toString(s));
```

Note:

Compilation Error if
we try to include
al.add(10)

Advantages



- **Type-safety** : We can hold only a single type of objects in generics. It doesn't allow to store other objects.
- **Type casting is not required**: There is no need to typecast the object.
- **Compile-Time Checking**: It is checked at compile time so problem will not occur at runtime.

Wildcard in Generics



```
abstract class Shape{  
final double pi = 3.14;  
double area;  
abstract void draw();  
}  
class Rectangle extends Shape{  
Rectangle(int l,int b){  
area = l*b; }  
void draw(){System.out.println("Area of Rect:"+area);}  
}  
class Circle extends Shape{  
Circle(int r){  
area = pi*r*r; }  
void draw(){System.out.println("Area of circle:"+area);}  
}
```


Wildcard in Generics



```
class test{
//creating a method that accepts only child class of Shape
public static void drawShapes(List<? extends Shape> lists){
for(Shape s:lists){
s.draw();}
}

public static void main(String args[]){
List<Rectangle> list1=new ArrayList<Rectangle>();
list1.add(new Rectangle(3,5));

List<Circle> list2=new ArrayList<Circle>();
list2.add(new Circle(2));
list2.add(new Circle(5));
drawShapes(list1);
drawShapes(list2);
}}
```

Output:

Area of Rect:15.0
Area of circle:12.56
Area of circle:78.5



Comparable Interface

Comparable Interface



- It is used to order the objects of user-defined class.
- It is found in java.lang package and contains only one method named compareTo(Object)
- Elements can be sorted based on single data member eg: account number, name or age.
- We can sort the elements of:
 - String objects
 - Wrapper class objects
 - User-defined class objects

Comparable-Example

```
import java.util.*;

class Account implements Comparable<Account>{
    int acc;
    String name;
    float amt;
    Account(int acc,String name,float amt){
        this.acc = acc;
        this.name = name;
        this.amt = amt; }
    public int compareTo(Account ac){
        if(amt==ac.amt)
            return 0;
        else if(amt>ac.amt)
            return 1;
        else
            return -1; }
    public String toString() {
        return "Acc. No.: "+acc+" Name: "+name+" Amount: "+amt;}
}
```

Comparable-Example



```
class Test{  
    public static void main(String[] args) {  
        List<Account> al = new ArrayList<Account>();  
  
        al.add(new Account(111,"Ankit",5000));  
        al.add(new Account(112,"Ashok",4000));  
        al.add(new Account(123,"Ryan",5000));  
  
        Collections.sort(al);  
  
        for(Account a:al)  
            System.out.println(a);  
    }  
}
```



Comparator Interface

Comparator Interface



- Used to order user defined class
- This interface is found in java.util package and contains 2 methods
 - `compare(Object obj1, Object obj2)`
 - `equals(Object element)`
- It provides multiple sorting sequence
 - Elements can be sorted based on any data member

Comparator - Example



```
import java.util.*;
```

```
class Account{
```

```
    int acc;
```

```
    String name;
```

```
    float amt;
```

```
    Account(int acc,String name,float amt){
```

```
        this.acc = acc;
```

```
        this.name = name;
```

```
        this.amt = amt; }
```

```
    public String toString() {
```

```
        return "Acc. No.: "+acc+" Name: "+name+" Amount: "+amt;}
```

```
    }
```


Comparator - Example



```
class AmtCmp implements Comparator<Account>{  
    public int compare(Account a1,Account a2){  
        if(a1.amt==a2.amt)  
            return 0;  
        else if(a1.amt>a2.amt)  
            return 1;  
        else  
            return -1; }  
}
```

Comparator - Example



```
class AccCmp implements Comparator<Account>{  
    public int compare(Account a1,Account a2){  
        if(a1.acc==a2.acc)  
            return 0;  
        else if(a1.acc>a2.acc)  
            return 1;  
        else  
            return -1; }  
}
```

Comparator - Example



```
class test {  
    public static void main(String[] args) {  
        List<Account> al = new ArrayList<Account>();  
  
        al.add(new Account(123,"Ankit",5000));  
        al.add(new Account(112,"Ashok",4000));  
        al.add(new Account(111,"Ryan",5000));  
  
        System.out.println("Comparison on Amount");  
        Collections.sort(al,new AmtCmp());  
        for(Account a:al)  
            System.out.println(a);  
  
        System.out.println("Comparison on Acc. No.");  
        Collections.sort(al,new AccCmp());  
        for(Account a:al)  
            System.out.println(a);  
    }  
}
```



Overriding Equals method

class Account implements Comparator<Account>{

int acc;

String name;

float amt;

Account(int acc,String name,float amt){

this.acc = acc;

this.name = name;

this.amt = amt; }

public boolean equals(Account a1) {

if (a1 == null)

return false;

if(this.acc != a1.acc)

return false;

if(this.amt != a1.amt)

return false;

if(!(a1.name.equals(this.name)))

return false;

return true;}

Overriding Equals method



```
public String toString() {  
    return "Acc. No.: "+acc+" Name: "+name+" Amount: "+amt;}  
public int compare(Account arg0, Account arg1) {  
    // TODO Auto-generated method stub  
    return 0;}  
}
```

```
class test {  
    public static void main(String[] args) {  
        List<Account> al = new ArrayList<Account>();  
        al.add(new Account(111,"Ryan",5000));  
        al.add(new Account(112,"Ryan",5000));  
        al.add(new Account(111,"Ryan",5000));  
        System.out.println(al.get(0).equals(al.get(2)));  
        System.out.println(al.get(0).equals(al.get(1))); }  
}
```

Multiple Bounds in Generics



```
public class test {  
    public static void main(String[] args) {  
        System.out.printf("Max of %d, %d and %d is %d\n\n",  
            3, 4, 5, maximum( 3, 4, 5 ));  
        System.out.printf("Max of %.1f,%.1f and %.1f is %.1f\n\n",  
            6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ));  
        System.out.printf("Max of %s,%s and %s is %s\n\n",  
            "s", "j", "r", maximum( "s", "j", "r" ));  
    }  
    public static <T extends Comparable<T>> T maximum(T x, T y, T z) {  
        T max = x;  
        if(y.compareTo(max) > 0) {  
            max = y;        }  
        if(z.compareTo(max) > 0) {  
            max = z;        }  
        return max;    }  
}
```

Output:

Max of 3, 4 and 5 is 5

Max of 6.6,8.8 and 7.7 is 8.8

Max of s,j and r is s

Multiple Bounds in Generics



```
public class test {  
    public static void main(String[] args) {  
        System.out.printf("Max of %d, %d and %d is %d\n\n",  
            3, 4, 5, maximum( 3, 4, 5 ));  
        System.out.printf("Max of %.1f,%.1f and %.1f is %.1f\n\n",  
            6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ));  
        System.out.printf("Max of %s,%s and %s is %s\n\n",  
            "s", "j", "r", maximum( "s", "j", "r" ));  
    }  
    public static <T extends Number & Comparable<T>> T maximum(T x, T  
        y, T z) {  
        T max = x;  
        if(y.compareTo(max) > 0) {  
            max = y;        }  
        if(z.compareTo(max) > 0) {  
            max = z;        }  
        return max;    } } }
```

Error:

The method maximum(T, T, T) in the type test is not applicable for the arguments (String, String, String)