Java Generics Short Note 20 February , 2022

Agenda of Generics:

1. Introduction
2. Type-Safety
3. Type-Casting
4. Generic Classes
5. Bounded Types
6. Generic methods and wild card character(?)
7. Communication with non generic code
8. Conclusions

Why we use Java Generics?

We will use java generics because of generics provides type-safety and also it's resolve type casting problems.

Now we learn about Array , Collection and Generics:

Array Example give below:

```
public class Example_1 {

        public static void main(String[] args) {
                String[] s=new String[10];
                s[0]="Java";
                s[1]="Programming";
                //s[2]=10;
         //here we will get compile time error

                String n1=(String)s[0];
                String n2=s[0];

                System.out.println(s[0]);
                System.out.println(s[1]);
                System.out.println(n1);
        }
}
```

This example which we Learn:

1. A Array will be fixed size
2. A Array Always Provide us type-safety.
3. We can only add the value of the type that we will put inside the array. Otherwise the compile time error will show.
4. A array type casting is not mandatory.
5. Such as this example we can add only string type value.

Output:

```
Java
Programming

Java
```

Collection Example give below:

```
import java.util.*;
public class Example_2 {
        public static void main(String[] args) {
                ArrayList l=new ArrayList();
                l.add("Java");
                l.add("Programming");
                l.add(10);

                String n1=(String)l.get(0);
                String n2=(String)l.get(1);
                //String n3=(String)l.get(2);//here type
casting error do not show
                System.out.println(l.get(0));
                System.out.println(l.get(1));
                System.out.println(l.get(2));

                System.out.println(n1);
                System.out.println(n2);
                //System.out.println(n3);//also error do not
show
        }
}
```

This example which we Learn:

1. This is a collection type array. In this array we can add any type value.
2. Collection do not provide us type safety.
3. Compile time do not show any error but run time it will show error.
4. Type casting is mandatory here.
5. It is not fixed size.

Output:

```
Java
Programming
10
```

Type casting error show in runtime

Generics Example give below:

```
import java.util.*;
public class Example_3 {
        public static void main(String[] args) {

        ArrayList<String> l=new ArrayList<String>();
                l.add("Wornoz");
                l.add("Qurishe");
                //l.add(10);//here will show in compile     time
error
                // because we can add only string type value

                System.out.println(l.get(0));
                System.out.println(l.get(1));

                String n1=l.get(0);//type casting not
mandatory
                String n2=(String)l.get(1);//here use type
casting

                System.out.println("without type- casting:"
+n1);
                System.out.println("with type-casting:" +n2);
        }
}
```

This example which we Learn:

1. For this Array List we can add only string type value. otherwise we get compile time error.so we can it provides type safety.
2. It is not fixed size.
3. Type casting not mandatory here.
4. It can detect error at compile time.

Output:

```
Wornoz
Qurishe
without type-casting:Wornoz
with type-casting:Qurishe
```

```
ArrayList<String> l=new ArrayList<String>();

(Base Type) ====ArrayList
(Parameter Type)====String

List<String> l1=new ArrayList<String>();//valid
Collection<String> l1=new ArrayList<String>();//valid
ArrayList<Object> l1=new ArrayList<String>();//not valid

ArrayList<int> l1=new ArrayList<int>();//not valid
```

Polymorphism concept is applicable only for the base type but not parameter type.

Collection concept applicable for object. This object can be class name, interface name. but not primitive type. also , as a parameter we use wrapper class for primitive data types.

Now we know also Generics Type:

1. T - Type (T means type parameter, type parameter means any type).also, as a type parameter we can use only class or interface . primitive type are not allowed as a type parameter.
2. E – Element
3. K – Key
4. N – Number
5. V – Value

Normal Java Class and Generic Java Class Example Given Below:

Normal Java Class Example Format	Generic Java Class Example Format

Class Test{
::::::::
}
	Class Test<T>{
	:::::::::
	}

	After class name we have to declare "<T>"type parameter. Type parameter T can be any type.

Based on our requirement we can define our own Generic classes also	Class Test<T>{
	:::::::::
	}
	
	Main method :
	Test<Gold> t1=new Test<Gold>();///this is valid
	Test<Silver> t2=new Test<Silver>();//this is valid

Java 1.4v a non-generic array list declared	**Java 1.5v a generic array list declared**
This is a normal java class	This is a java generic class

```java
public class Example_4 {
        Object o;

        public Object getO() {
                return o;
        }

        public void setO(Object o) {
                this.o = o;
        }

        public static void main(String[] args) {
                Example_4 l1 = new Example_4();
                l1.setO("hello");

                Example_4 l2 = new Example_4();
                l2.setO(10);

                System.out.println(l1.getO());
                System.out.println(l2.getO());
        }
}
```

```java
public class Example_5<T> {
        T obj;

        public T getObj() {
                return obj;
        }

        public void setObj(T obj) {
                this.obj = obj;
        }

        public static void main(String[] args) {
                Example_5<String> e1=new
Example_5<String>();
                e1.setObj("hello");

                Example_5<Integer> e2=new
Example_5<Integer>();
                e2.setObj(10);

                System.out.println(e1.getObj());
                System.out.println(e2.getObj());
        }
}
```

```
hello
10
```

```
hello
10
```

we can add any type value by using object .also we use getter and setter method. Such as this example we will add string and integer type value.

T Means any type parameter.it is a generic class. we can add any type value by using generic class object. also we use getter and setter method.
we can add any type value by using object .also we use getter and setter method.
Such as this example we will add string and integer type value.

```java
class Example_6<T>{
        T obj;

        public T getObj() {
                return obj;
        }

        public void setObj(T obj) {
                this.obj = obj;
        }
         public void show() {
            System.out.println("the type of object is :"
                            +obj.getClass().getName());
         }
}
```

Save name is : GenericsDemo.java

```java
public class GenericsDemo {

        public static void main(String[] args) {
                Example_6<Integer> t1=new
Example_6<Integer>();
                t1.setObj(10);
                t1.show();
                System.out.println(t1.getObj());

                Example_6<String> t2=new
Example_6<String>();
                t2.setObj("Java");
                t2.show();
                System.out.println(t2.getObj());

                Example_6<Double> t3=new
Example_6<Double>();
                t3.setObj(10.55);
                t3.show();
                System.out.println(t3.getObj());
        }
```

Based on our requirement we can create our own generic classes also.
Example:

Output:

```
class Account<T> {
::::::::::::::::::::::::::::
}
Account<Gold> g1=new Account<Gold>();
Account<Silver> g2=new Account<Silver>();
```

```
the type of object is :java.lang.Integer
10
the type of object is :java.lang.String
Java
the type of object is :java.lang.Double
10.55
```

Bounded Types: Bound Means a Boundary . A Boundary Will be a class, Interface , methods. If we want to give a boundary then we have to use the “extends” keyword.but we can not use “implements” keyword.

Two types boundary.
1.bounded types (there will be a boundary such as: Class Test <T extends Number>){}.

Bounded type example

```
public class Example_7<T extends Number> {
        T obj;

        public T getObj() {
                return obj;
        }

        public void addObj(T obj) {
                this.obj = obj;
        }

public static void main(String[] args) {
       Example_7<Integer> t1=newExample_7<Integer>();
        t1.addObj(12);
        System.out.println("Integer Value is:" +t1.getObj());

        Example_7<Double> t2=new Example_7<Double>();
        t2.addObj(12.55);
        System.out.println("Double Value is:" +t2.getObj());

        /*
        Example_7<String> t3=new Example_7<String>();
        t3.addObj("Java");
        System.out.println("Integer Value is:" +t3.getObj());
        */
        }
}
```

1. Bounded will be a class or interface.
2.must be use extends keywords.
3. If bounded any class then we will add either class or child class type.
4. such as: this example we can’t add string type value . cause of , string is not child of number class.
5. we can't define bounded types by using implements and super keyword.
(Same to inteface)

Output:

```
Integer Value is:12
Double Value is:12.55
```

Show compile time error.

```
Public class Test<T extends Number & Runnable>{
::::::::::::::::::::::
}//valid

Public <T extends Number & Runnable>void m1(){
::::::::::::::::::::::
}//this method is valid
```

1.As a type parameter , at a same time,we can access class or inteface.also we can add one more interface class.but class name will be firstly then after interface name.

2. never can not use one more class at a time.

Also same as method

```
public class Example_8{

public static void main(String[] args) {
ExampleInterface<Runnable> r1=new ExampleInterface<Runnable>();
System.out.println(r1.getClass());

ExampleInterface<Thread> r2=new ExampleInterface<Thread>();
System.out.println(r2.getClass());
        }
}
```

This example is a bounded type interface class .

Output:

```
class practice.ExampleInterface
class practice.ExampleInterface
```

`class ExampleInterface<T extends Runnable>{`
 	
`}`	

2.unbounded (there will no boundary such as: Class Test <T >){}.

UnBounded type example	

public class Example_9<T> {
 	
        public static void main(String[] args) {
                Example_9<Integer> t1=new Example_9<Integer>();
                Example_9<Double> t2=new Example_9<Double>();
                Example_9<String> t3=new Example_9<String>();
                Example_9<Runnable> t4=new Example_9<Runnable>();
                Example_9<Thread> t5=new Example_9<Thread>();
                Example_9<Number> t6=new Example_9<Number>();
                //all are valid cause of this is unbounded type.
        }
}
	1. Here, we add value creating object. And using getter setter method.

Generic Methods and Wild Cards(?)

"?" is a wildcard character.
"Extends", is a keyword.

Method	describe	Remarks

`public static void display(List<?> list){`
`}`
	Any type value added using this method.	Because , no type parameter use here.

`public static void display(List<String> list){`
`}`
	Only string type value we can pass this method.	Use string type .

`public static void display(List<? extends Number> list){`
`}`
	We can call this method number class or its child class.	Given example under.

`public static void display(List<? super Integer > list){`
`}`
	Here we can pass Integer value or Integer Super class value(that is double)	Given example under.

Method Bounded type example by using wildcard/upper bound	

public static void display(List<? extends Number> list){
:::;
                                        }	

import java.util.*;
public class Example_10 {
 	
public static double Sum(ArrayList<? extends Number> num) {
        double sum=0.0;
        for (Number number : num) {
                sum=sum+number.doubleValue();
        }
        return sum;
    }
public static void main(String[] args) {
        ArrayList<Integer> t1=new ArrayList<Integer>();
                t1.add(30);
                t1.add(40);
        System.out.println("Displaying The Sum Result is= " +Sum(t1));
 	
        ArrayList<Double> t2=new ArrayList<Double>();
                t2.add(30.5);
                t2.add(40.5);
        System.out.println("Displaying The Sum Result is= " +Sum(t2));
	Using this method we can pass either number class or its child .
	
	Other wise we will get compile time error.
	
	This is a wild card method.
	
	Output:
	
	Displaying The Sum Result is=
	70.0
	Displaying The Sum Result is=
	71.0

```
        }
}
```

Method Bounded type example by using wildcard/ Lower bound

```
public static void display(List<? extends Integer> list){
…………………………………………………………………………………………………;
                                                                        }
```

```
import java.util.*;
public class Example_11 {

        public static void display(List<? super Integer> list) {
                for (Object object : list) {
                        System.out.println(object);
                }
        }

        public static void main(String[] args) {
                List<Integer> i1=Arrays.asList(12,13,15);
                System.out.println("Lower bound Integer values:");
                display(i1);

                List<Number> d1=Arrays.asList(12.34,43.55,56.89);
                System.out.println("Lower bound Number values:");
                display(d1);
        }
}
```

Here it is a lower bound wildcard example. From this example we learn, if we use "super" keyword Then we add value either that class or parent class value.

Such as this example , we add only integer value and that parent class value. By mistake we can add any other value then we will get compile time error.

Output:

```
Lower bound Integer values:
12
13
15
Lower bound Number values:
12.34
43.55
56.89
```

Communication With Non-Generic Code:

```
import java.util.*;
public class Example_12 {

        public static void methodOne(ArrayList l) {
                /*--this area is a non  generic area
                 * we can add any type value
                --*/
                l.add(10);//valid
                l.add("Java");//valid
        }
        public static void main(String[] args) {
                /*--this area is a generic area
                 * we can add only string type value
                --*/
                ArrayList<String> l1=new ArrayList<String>();
                l1.add("Hello");//valid
                //l.add(10);//not valid.it is integer type

                methodOne(l1);//in non generic area we add generic
area value
                System.out.println(l1);
                //l.add(34);//not valid
        }
}
```

Output:

```
[Hello, 10, Java]
```