Java Generics

Generics was first introduced in Java5. Now it is one of the most profound feature of java programming language. Generic programming enables the programmer to create classes,interfaces and methods in which type of data is specified as a parameter. It provides a facility to write an algorithm independent of any specific type of data. Generics also provide type safety. Type safety means ensuring that an operation is being performed on the right type of data before executing that operation.

Using Generics, it has become possible to create a single class ,interface or method that automatically works with all types of data(Integer, String, Float etc). It has expanded the ability to reuse the code safely and easily.

Before Generics was introduced, generalized classes,interfaces or methods were created using references of type Object because Object is the super class of all classes in Java, but this way of programming did not ensure type safety.

Syntax for creating an Object of a Generic type

```
Class_name <data type> reference_name = new Class_name<data type> ();

OR

Class_name <data type> reference_name = new Class_name<>();
```

Copy

This is also known as **Diamond Notation** of creating an object of Generic type.

Example of Generic class

```

//<> brackets indicates that the class is of generic type

class Gen <T>

{

  T ob;     //an object of type T is declared<

  Gen(T o)  //constructor

  {

    ob = o;

  }

  public T getOb()

  {

    return ob;

  }
}



class Demo

{

  public static void main (String[] args)

  {

    Gen < Integer> iob = new Gen<>(100);    //instance of Integer type Gen Class

    int  x = iob.getOb();

    System.out.println(x);

    Gen < String> sob = new Gen<>("Hello");  //instance of String type Gen Class

    String str = sob.getOb();

    System.out.println(str);

  }

}
```

Copy

```
100

Hello
```

In the above program, we first passed an Integer type parameter to the Generic class. Then, we passed a String type parameter to the same Generic class. Hence, we reused the same class for two different data types. Thus, Generics helps in code reusability with ease.

Generics Work Only with Objects

Generics work only with objects i.e the type argument must be a class type.You cannot use primitive datatypes such as int, char etc. with Generics type. It should always be an object. We can use all the Wrapper Class objects and String class objects as Generic type.

```
Gen<int> iOb = new Gen<int>(07);    //Error, can't use primitive type
```

Copy

Generics Types of different Type Arguments are never same

Reference of one generic type is never compatible with other generic type unless their type argument is same. In the example above we created two objects of class **Gen**, one of type **Integer**, and other of type **String**, hence,

```
iob = sob;  //Absolutely Wrong
```

Copy

An array of Generic type cannot be created

Creation of a generic type array is not allowed in Generic programming. We can make a reference of an array, but we cannot instantiate it.

For example, In the above program, in class Gen,

T a[]; **//this is allowed**

T a[] = new T[10]; **//this is not allowed**

Generic Type with more than one parameter

In Generic parameterized types, we can pass more than one data type as parameter. It works the same as with one parameter Generic type.

```
class Gen <T1,T2>
{
  T1 name;
  T2 value;

  Gen(T1 o1,T2 o2)
  {
    name = o1;
    value = o2;
  }

  public T1 getName()
  {
    return name;
  }

  public T2 getValue()
  {
    return value;
  }

}
```

```
class Demo

{

  public static void main (String[] args)

  {

    Gen < String,Integer> obj = new Gen<>("StudyTonight",1);


    String  s = obj.getName();

    System.out.println(s);


    Integer i = obj.getValue();

    System.out.println(i);

  }

}
```

Copy

```
StudyTonight

1
```

Note: Since there are two parameters in Generic class - T1 and T2, therefore, while creating an instance of this Generic class, we need to mention two data types that needs to be passed as parameter to this class.

Generic Methods

You can also create generic methods that can be called with different types of arguments. Based on the type of arguments passed to generic method, the compiler handles each method.

The syntax for a generic method includes a type-parameter inside angle brackets, and should appear before the method's return type.

```
<type-parameter> return_type method_name (parameters) {...}
```

Copy

Example of Generic method

```
class Demo

{

  static <V, T> void display (V v, T t)
  {
    System.out.println(v.getClass().getName()+" = " +v);
    System.out.println(t.getClass().getName()+" = " +t);
  }
  public static void main(String[] args)
  {
    display(88,"This is string");
  }
}
```

Copy

```
java lang.Integer = 88
java lang.String = This is string
```

Generic Constructors in Java

It is possible to create a generic constructor even if the class is not generic.

Example of Generic Constructor

```
class Gen
{
  private double val;
  <T extends Number> Gen(T ob)
  {
    val = ob.doubleValue();
  }
  void show()
  {
    System.out.println(val);
  }
}


class Demo
```

```
{

  public static void main(String[] args)

  {

    Gen g = new Gen(100);

    Gen g1 = new Gen(121.5f);

    g.show();

    g1.show();

  }

}
```

Copy

```
100.0

121.5
```

Generic Bounded type Parameter

You can also set restriction on the type that will be allowed to pass to a type-parameter. This is done with the help of extends keyword when specifying the type parameter.

```
 < T extends Number >
```

Copy

Here we have taken **Number** class, it can be any wrapper class name. This specifies that T can be only be replaced by **Number** class data itself or any of its subclass.

Generic Method with bounded type Parameters

In this example, we created a display method that allows only number type or its subclass type.

```
class Demo

{

  static < T, V extends Number> void display(T t, V v)

  {

    System.out.println(v.getClass().getName()+" = " +v);

    System.out.println(t.getClass().getName()+" = " +t);

  }

  public static void main(String[] args)

  {

    display ("this is string",99);
    }

}
```

Copy

```
java.lang.String = This is string

java.lang.Double = 99.0
```

Type V is bounded to Number type and its subclass only. If `display(88,"This is string")` is uncommented, it will give an error of type incompatibility, as String is not a subclass of Number class.