Generic Interface

Similar to generic class, when an interface is defined using generic type parameter, it is called generic interface in java. When a generic interface is implemented by either a generic class or non-generic class, the type argument must be specified. The generic interface has main two advantages that are as:

A generic interface can be implemented for different types of data.

It allows placing constraints (i.e. bounds) on the types of data for which interface can be implemented.

Syntax Declaration of Generic Interface

The general syntax to declare a generic interface is as follows:

```
interface interface-name<T>
{
  void method-name(T t); // public abstract method.
}
```

In the above syntax, <T> is called a generic type parameter that specifies any data type used in the interface. An interface can also have more than one type parameter separated by a comma.

We have studied already that whenever there is an interface, we should also have implementation classes that implement all methods of an interface.

The general syntax to write an implementation class that implements the above interface, as follows:

```
class class-name<T>  implements interface-name<T>
{
  public void method-name(T t)
{
 // Write code for this method as per requirement.
 }
}
```

Here, T represents a generic parameter that specifies any data type.

Let’s understand the above syntax with the help of an example.

```
// A generic interface with one type parameter T.
  interface A<T> // Interface name is A.
  {
     public void m1(T t); // An abstract method with parameter t of type T.
   }
// Generic class with one type parameter T implementing generic interface.
    class B<T> implements A<T>
   {
     public T obj;
     public void m1(T t)
     {
        obj = t;
      }
    }
// Non-generic class implementing generic interface.
    class C implements A<Integer>
    {
        public Integer obj;
        public void m1(Integer t)
        {
           obj = t;
        }
    }
```

- **Generic Interface Example Programs**

1. Let’s create a very simple example program where we will create a generic interface A and implementing class Test. Then the implementation class will implement the method of interface A to display a message.

Program code 1:

```
// A generic interface.
 public interface A<T>
 {
// Declare an abstract method with parameter t of type T.
   public void m1(T t);
}
// This generic class implement generic interface.
public class Test<T> implements A<T> {
public void m1(T t)
{
  System.out.println("m1-Test");
}
public static void main(String[] args)
{
  Test<Integer> t = new Test<Integer>();
    t.m1(25);
  Test<String> t2 = new Test<String>();
    t2.m1("John");
  }
}
```

Output:

```
        m1-Test
        m1-Test
```

2. Let’s take an example program where we will create a generic interface and implementation class. Then the implementation class will be used to display the class name of object passed to the method.

Program code 2:

```
public interface Fruit<T>
{
// This method can accept any type of object.
  public void taste(T fruit);
}
public class AnyFruit<T> implements Fruit<T>
{
 public void taste(T fruit)
 {
  // Get the class name of object passed to this method.
          String fruitname = fruit.getClass().getName();
          System.out.println(fruitname);
 }
}
public class Banana {
}
public class Orange {
}
public class GenericClass {
```

```
public static void main(String[] args)
{
// Create an object of class Banana and pass it to class AnyFruit.
    Banana b = new Banana();
    AnyFruit<Banana> fruit1 = new AnyFruit<Banana>();
    fruit1.taste(b);
// Create an object of Orange and pass it to class AnyFruit.
    Orange o = new Orange();
    AnyFruit<Orange> fruit2 = new AnyFruit<Orange>();
     fruit2.taste(o);
 }
}
```

Output:

```
        Generics.Banana
        Generics.Orange
```

3. When a non-generic class implements a generic interface, the type parameter does not follow class name. The type parameters must be replaced by actual types while implementing the generic interface.

Let's take a very simple example program based on it.

<u>Program code 3:</u>

```
public interface Myinterface<T>
{
   T m1();
}
// A non-generic class implementing a generic interface.
  public class Myclass implements Myinterface<Integer>
```

```
 {
     public Integer m1()
     {
         System.out.println("m1-Myclass");
         return null;
      }
public static void main(String[] args)
{
     Myclass obj = new Myclass();
     obj.m1();
  }
}
```

Output:

```
        m1-Myclass
```