Generic Programming

Courtesy: *Java The Complete Reference by* Herbert Schildt

Generics

- Through the use of generics, it is possible to create classes, interfaces, and methods that will work with various kinds of data.
- With generics, we can define an algorithm once, independently of any specific type of data, and then apply that algorithm to a wide variety of data types without any additional effort.

Syntax:

```
2  class Generic_class < T >
3  {
4      T member1;
5
6      // .. other members
7
8      //  methods
9  }
10
11 class Generic_Demo
12 {
13     public static void main(String args[])
14     {
15         Generic_class<Integer>  int_Obj = new Generic_class<Integer>( 88 );
16
17         Generic_class<String>  string_Obj = new Generic_class<String>( "Hello" );
18     }
19 }
```

Example-1:

```
// A simple generic class.
// Here, T is a type parameter that will be replaced by a real type
// when an object of type Gen is created.
class Gen<T> {
  T ob; // declare an object of type T

  Gen(T o) {
    ob = o;
  }

  // Return ob.
  T getob() {
    return ob;
  }

  // Show type of T.
  void showType() {
    System.out.println("Type of T is " +
                       ob.getClass().getName());
  }
}
```

```java
// Demonstrate the generic class.
class GenDemo {
  public static void main(String args[]) {
    // Create a Gen reference for Integers.
    Gen<Integer> iOb;

    iOb = new Gen<Integer>(88);

    // Show the type of data used by iOb.
    iOb.showType();

    // Get the value in iOb.
    int v = iOb.getob();
    System.out.println("value: " + v);

    // Create a Gen object for Strings.
    Gen<String> strOb = new Gen<String>("Generics Test");

    // Show the type of data used by strOb.
    strOb.showType();

    // Get the value of strOb.
    String str = strOb.getob();
    System.out.println("value: " + str);
  }
}
```

Example-1:

```
// Demonstrate the generic class.
class GenDemo {
  public static void main(String args[]) {
    // Create a Gen reference for Integers.
    Gen<Integer> iOb;

    iOb = new Gen<Integer>(88);

    // Show the type of data used by iOb.
    iOb.showType();

    // Get the value in iOb.
    int v = iOb.getob();
    System.out.println("value: " + v);

    // Create a Gen object for Strings.
    Gen<String> strOb = new Gen<String>("Generics Test");

    // Show the type of data used by strOb.
    strOb.showType();

    // Get the value of strOb.
    String str = strOb.getob();
    System.out.println("value: " + str);
  }
}
```

```
class Gen<T>
{
  T ob;

  Gen( T o )
  {  ob = o; }

  T getob()
  {   return ob;  }

  void showType()
  {
    System.out.println("Type of T is "
          + ob.getClass().getName() );
  }
}
```

OUTPUT:

```
Type of T is java.lang.Integer
value: 88
Type of T is java.lang.String
value: Generics Test
```

- **Note**: Generics work only with Reference Types

Example:

```
Gen<int>  intOb  = new  Gen<int>( 53 );     // Error, can't use primitive type
```

Invalid type

Example-2:

```
1  // NonGen is functionally equivalent to Gen
2  // but does not use generics.
3  class NonGen
4  {
5    Object ob; // ob is now of type Object
6
7    // Pass the constructor a reference to an object of type Object
8    NonGen(Object o)
9    {
10       ob = o;
11   }
12
13   // Return type Object.
14   Object getob()
15   {
16       return ob;
17   }
18 }
```

Example-2:

```
class NonGenDemo {
  public static void main(String args[]) {
    NonGen iOb;

    // Create NonGen Object and store
    // an Integer in it. Autoboxing occurs.
    iOb = new NonGen(88);

    // Get the value of iOb.
    // This time, a cast is necessary.
    int v = (Integer) iOb.getob();
    System.out.println("value: " + v);

    // Create another NonGen object and
    // store a String in it.
    NonGen strOb = new NonGen("Non-Generics Test");

    // Get the value of strOb.
    // Again, notice that a cast is necessary.
    String str = (String) strOb.getob();
    System.out.println("value: " + str);

    iOb = strOb; // This compiles, but is conceptually wrong!
    v = (Integer) iOb.getob(); // run-time error!
  }
}
```

```
class NonGen
{
  Object ob;

  NonGen(Object o)
  {
      ob = o;
  }

  Object getob()
  {
      return ob;
  }
}
```

OUTPUT:

```
value: 88
value: Non-Generics Test
Exception in thread "main" java.lang.ClassCastException: class java.lang.String cannot be cast to class
 java.lang.Integer (java.lang.String and java.lang.Integer are in module java.base of loader 'bootstrap
')
        at NonGenDemo.main(NonGenDemo.java:39)
```

A Generic Class with Two Type Parameters

```
1  // Generic class with two type parameters
2  class Two_Generic_Types< T1 , T2 >
3  {
4      T1 ob1;
5      T2 ob2;
6
7      Two_Generic_Types( T1 o1 , T2 o2 )
8      {
9          ob1 = o1 ; ob2 = o2;
10     }
11
12     // Other code...
13 }
14
15 class Generic_Demo
16 {
17     Two_Generic_Types<Integer,String> Generic_obj
18
19                 = new Two_Generic_Types<Integer,String>( 10 , "Hello" );
20
21         // Other code...
22 }
```

Example-3:

```
// A simple generic class with two type
// parameters: T and V.
class TwoGen<T, V> {
  T ob1;
  V ob2;

  TwoGen(T o1, V o2) {
    ob1 = o1;
    ob2 = o2;
  }

  void showTypes() {
    System.out.println("Type of T is " +
                       ob1.getClass().getName());

    System.out.println("Type of V is " +
                       ob2.getClass().getName());
  }

  T getob1()
  {  return ob1; }

  V getob2()
  {  return ob2; }
}
```

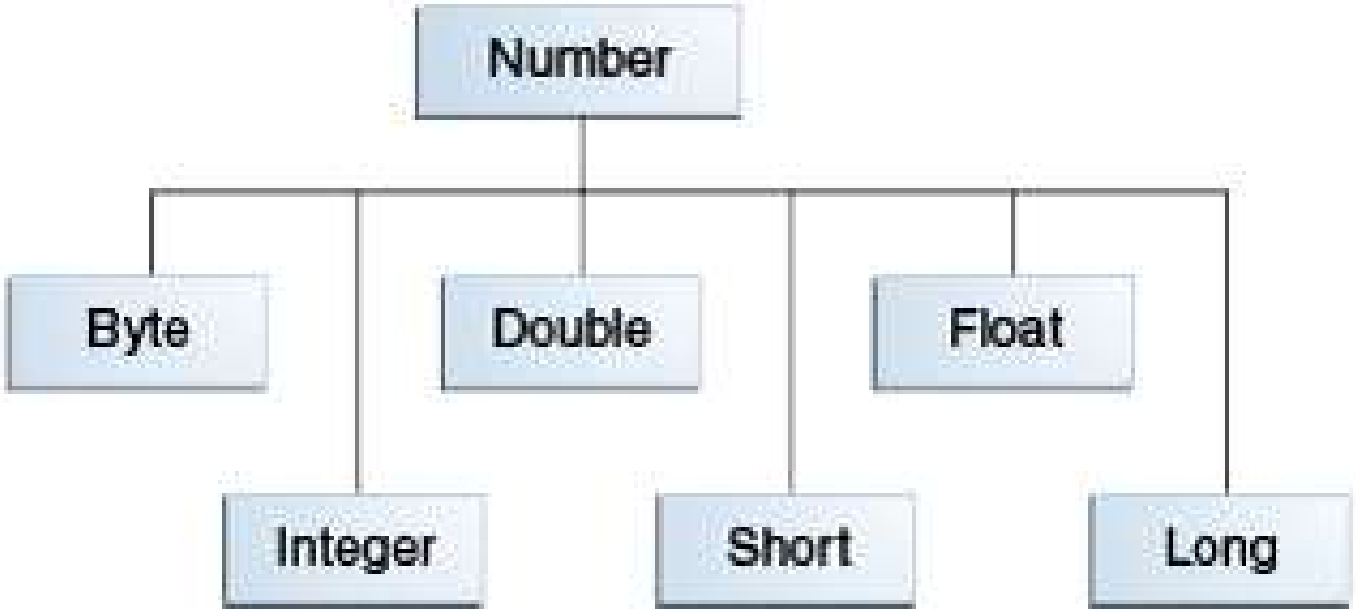
Example-3:

```
1  // A simple generic class with two type
2  // parameters: T and V.
3  class TwoGen<T, V> {
4    T ob1;
5    V ob2;
6
7    TwoGen(T o1, V o2) {
8      ob1 = o1;
9      ob2 = o2;
10   }
11
12   void showTypes() {
13     System.out.println("Type of T is " +
14                        ob1.getClass().getName());
15
16     System.out.println("Type of V is " +
17                        ob2.getClass().getName());
18   }
19
20   T getob1()
21   {  return ob1; }
22
23   V getob2()
24   {  return ob2; }
25 }
```

```
28 class SimpGen {
29   public static void main(String args[]) {
30
31     TwoGen<Integer, String> tgObj =
32       new TwoGen<Integer, String>(88, "Generics");
33
34     tgObj.showTypes();
35
36     // Obtain and show values.
37     int v = tgObj.getob1();
38     System.out.println("value: " + v);
39
40     String str = tgObj.getob2();
41     System.out.println("value: " + str);
42   }
43 }
```

OUTPUT:

```
Type of T is java.lang.Integer
Type of V is java.lang.String
value: 88
value: Generics
```


Number
Byte
Double
Float
Integer
Short
Long