Generics In Java

The Java Generics programming was introduced in Java 5 to deal with type-safe objects.
Now what do we mean by Type-Safety?

Let us take an example:-

- **Create a ArrayList without generics:-**

```
List list = new ArrayList();
list.add("hello");
String s=list.get(0);// Compile time error
String s = (String) list.get(0);//typecasting
```

- **Create a ArrayList with generics:-**

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s=list.get(0);// No CE as type casting not required
list.add(10);// Compile error
```

Type Erasure:- When Java compiler sees code written using Generics it completely erases that code and convert it into raw type i.e. code without Generics. All type related information is removed during erasing. So your ArrayList<Gold> becomes plain old ArrayList prior to JDK 1.5, formal type parameters e.g. <K, V> or <E> gets replaced by
either Object or Super class of the Type.

Creating a generic class:-
To create a generic class we have to define a parameter type using <T> where T means any type of Object.
Eg:-

```
class Test<T>
{
   // An object of type T is declared
   T obj;
   Test(T obj) {
       this.obj = obj;
     }  // constructor
   public T getObject()  {
           return this.obj;
     }
}
```

// To create an instance of generic class

```
BaseType <Type> obj = new BaseType <Type>()
```

Note: In Parameter type we can not use primitives like
'int','char' or 'double'.

```
// Driver class to test above
class Main
{
   public static void main (String[] args)
   {
      // instance of Integer type
      Test <Integer> iObj = new Test<Integer>(15);
      System.out.println(iObj.getObject());

      // instance of String type
      Test <String> sObj =
                     new Test<String>("GeeksForGeeks");
      System.out.println(sObj.getObject());
   }
}
```

We can also pass multiple Type Parameters in generic classes:-

```
// A Simple Java program to show multiple
// type parameters in Java Generics

// We use < > to specify Parameter type
class Test<T, U>
{
   T obj1;  // An object of type T
   U obj2;  // An object of type U

   // constructor
   Test(T obj1, U obj2)
   {
      this.obj1 = obj1;
      this.obj2 = obj2;
   }

   // To print objects of T and U
   public void print()
   {
      System.out.println(obj1);
      System.out.println(obj2);
   }
}

// Driver class to test above
class Main
{
   public static void main (String[] args)
   {
      Test <String, Integer> obj =
         new Test<String, Integer>("GfG", 15);

      obj.print();
   }
}
```

Generic Method:-

We can also write generic method that can be called with different types of arguments based on the type of arguments passed to generic method, the compiler handles each method.

```
// A Simple Java program to show working of user defined
// Generic method

class Test
{
   // A Generic method example
   static <T> void genericDisplay (T element)
   {
      System.out.println(element.getClass().getName() +
                         " = " + element);
   }

   // Driver method
   public static void main(String[] args)
   {
       // Calling generic method with Integer argument
      genericDisplay(11);

      // Calling generic method with String argument
      genericDisplay("GeeksForGeeks");

      // Calling generic method with double argument
      genericDisplay(1.0);
   }
}
```