OBJECT ORIENTED PROGRAMMING

GENERICS IN JAVA

Generics means parameterized types.

It is possible to create classes that work with different data types.

An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

TYPES OF JAVA GENERICS

Generic Method: Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function.

Generic Classes: A generic class is implemented exactly like a non-generic class. The only difference is that it contains a type parameter section. There can be more than one type of parameter, separated by a comma.

GENERIC CLASS

To create objects of a generic class, we use the following syntax.

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

EXAMPLE 1

```
class Test<T>
{
    T data;
    Test(T d) {
            this.data = d;
     }
    public T getObject() {
            return this.data;
}}
```

EXAMPLE 1 (CONTINUE..)

```
class Main {
    public static void main(String[] args)
    {
        Test<Integer> iObj = new Test<Integer>(15);
        System.out.println(iObj.getObject());
        Test<String> sObj= new Test<String>(“Object Oriented Programming");
        System.out.println(sObj.getObject());
    }
}
```

EXAMPLE 2

```
class Test<T, U>
{
   T data1;  // An object of type T
   U data2;  // An object of type U
Test(T o1, U o2)
   {
      this.data1 = o1;
      this.data2 = o2;
   }
public void print() {
            System.out.println(data1);
            System.out.println(data2); }
}
```

EXAMPLE 2 (CONTINUE..)

```
class Main
{
    public static void main (String[] args)
    {
        Test <String, Integer> obj = new Test<String, Integer>(“OOP", 1145);

        obj.print();
    }
}
```

GENERIC FUNCTIONS

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

EXAMPLE

```
class Test {
static <T> void genericDisplay(T element) { System.out.println(element); }
  public static void main(String[] args)
    {
             genericDisplay(11);
             genericDisplay(“OOP");
             genericDisplay(1.0);
    }
}
```

ADVANTAGES OF GENERICS

Code Reuse: We can write a method/class/interface once and use it for any type we want.

Type Safety: Generics make errors to appear compile time than at run time (It's always better to know problems in your code at compile time rather than making your code fail at run time).