Generics in Java

- The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects.
- It makes the code stable by detecting the bugs at compile time.
- we can store any type of objects in the collection, i.e., non-generic.
- Now generics force the java programmer to store a specific type of objects.
- The Java Generics allows us to create a single class, interface, and method that can be used with different types of data (objects).

Note: **Generics** does not work with primitive types (int, float, char, etc..).

Advantage of Java Generics

- There are mainly 3 advantages of generics.

1) **Type-safety:** We can hold only a single type of objects in generics. It does not allow to store other objects.

Example:

```
List list = new ArrayList();
list.add(10);
list.add("10");
```

➢With Generics, it is required to specify the type of object we need to store.

```
List<Integer> list = new ArrayList<Integer>();
list.add(10);
list.add("10");// compile-time error
```

2)Type casting is not required: There is no need to typecast the object.

- Example:

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

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

Example:

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32);//Compile Time Error
```

Example of Generics in Java

```
import java.util.*;
class TestGenerics1{
public static void main(String args[]) {
ArrayList<String> list=new ArrayList
<String>();
list.add("rahul");
list.add("jai");
//list.add(32);//compile time error
String s=list.get(1);//type casting is not required
System.out.println("element is: "+s);

Iterator<String> itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

```
C:\Users\nhrao\Desktop>javac TestGenerics1.java

C:\Users\nhrao\Desktop>java TestGenerics1
element is: jai
rahul
jai
```

Java Generics using Map

```
import java.util.*;

class TestGenerics2{

public static void main(String args[]){

Map<Integer,String> map=new HashMap<Integer,String>();

map.put(1,"vijay");

map.put(4,"umesh");

map.put(2,"ankit");


//Now use Map.Entry for Set and Iterator

Set<Map.Entry<Integer,String>> set=map.entrySet();

Iterator<Map.Entry<Integer,String>> itr=set.iterator();

while(itr.hasNext()){

Map.Entry e=itr.next();//no need to typecast

System.out.println(e.getKey()+" "+e.getValue());

}

  }

}
```

```
1 vijay
2 ankit
4 umesh
```

Java Generics Class

We can create a class that can be used with any type of data. Such a class is known as Generics Class.

Java Generics Method

```
class Main {
  public static void main(String[] args) {

    // initialize the class with Integer data
    DemoClass demo = new DemoClass();

    // generics method working with String
    demo.<String>genericsMethod("Java Programming");

    // generics method working with integer
    demo.<Integer>genericsMethod(25);
  }
}

class DemoClass {

  // creae a generics method
  public <T> void genericsMethod(T data) {
    System.out.println("Generics Method:");
    System.out.println("Data Passed: " + data);
  }
}
```

```
Generics Method: Data Passed:
Java Programming
Generics Method:
Data Passed: 25
```

Example

```
// initialize generic class with Integer data
   GenericsClass<Integer> intObj = new GenericsClass<>(5);
   System.out.println("Generic Class returns: " + intObj.getData());

   // initialize generic class  with String data
   GenericsClass<String> stringObj = new GenericsClass<>("Java
Programming");
   System.out.println("Generic Class returns: " + stringObj.getData());
  }
}
```

Generic class

- A class that can refer to any type is known as a generic class. Here, we are using the T type parameter to create the generic class of specific type.

Example:

Creating a generic class:

```
class MyGen<T>{
T obj;
void add(T obj){this.obj=obj;}
T get(){return obj;}
}
```

The T type indicates that it can refer to any type (like String, Integer, and Employee).

Generic Example to find max values from the list

```
• public class MaximumTest {
  // determines the largest of three Comparable objects

  public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
    T max = x;   // assume x is initially the largest

    if(y.compareTo(max) > 0) {
      max = y;   // y is the largest so far
    }

    if(z.compareTo(max) > 0) {
      max = z;   // z is the largest now
    }
    return max;   // returns the largest object
  }

  public static void main(String args[]) {
    System.out.printf("Max of %d, %d and %d is %d\n\n",
      3, 4, 5, maximum( 3, 4, 5 ));

    System.out.printf("Max of %.1f,%.1f and %.1f is %.1f\n\n",
      6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ));

    System.out.printf("Max of %s, %s and %s is %s\n","pear",
      "apple", "orange", maximum("pear", "apple", "orange"));
  }
}
```

output

```
C:\Users\nhrao\Desktop>javac MaximumTest.java

C:\Users\nhrao\Desktop>java MaximumTest
Max of 3, 4 and 5 is 5

Max of 6.6,8.8 and 7.7 is 8.8

Max of pear, apple and orange is pear
```

Generic Example to find max and min values from the list

```
class MyClass<T extends Comparable<T>>
{
T[] vals;
MyClass(T[] o)
{
vals = o;
}
public T min()
{
T v = vals[0];
for(int i=1; i < vals.length; i++)
if(vals[i].compareTo(v) < 0)
v = vals[i];
return v;
}
public T max()
{
T v = vals[0];
for(int i=1; i < vals.length;i++)
if(vals[i].compareTo(v) > 0)
v = vals[i];
return v;
}
}
class GenDemo
{
public static void main(String args[])
{
int i;
Integer inums[]={10,2,5,4,6,1};
Character chs[]={'v','p','s','a','n','h'};
Double d[]={20.2,45.4,71.6,88.3,54.6,10.4};
MyClass<Integer> iob = new MyClass<Integer>(inums);
MyClass<Character> cob = new MyClass<Character>(chs);
MyClass<Double>dob = new MyClass<Double>(d);
System.out.println("Max value in inums: " + iob.max());
System.out.println("Min value in inums: " + iob.min());
System.out.println("Max value in chs: " + cob.max());
System.out.println("Min value in chs: " + cob.min());
System.out.println("Max value in chs: " + dob.max());
System.out.println("Min value in chs: " + dob.min());
}
}
```

output

```
C:\Users\nhrao\Desktop>javac GenDemo.java

C:\Users\nhrao\Desktop>java GenDemo
Max value in inums: 10
Min value in inums: 1
Max value in chs: v
Min value in chs: a
Max value in chs: 88.3
Min value in chs: 10.4
```