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.

Before generics, 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.

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety: We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

Without Generics, we can store any type of objects.

```
1. List list = new ArrayList();
2. list.add(10);
3. list.add("10");
4. With Generics, it is required to specify the type of object we need to store.
5. List<Integer> list = new ArrayList<Integer>();
6. list.add(10);
7. list.add("10");// compile-time error
```

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

Before Generics, we need to type cast.

```
1. List list = new ArrayList();
2. list.add("hello");
3. String s = (String) list.get(0);//typecasting
4. After Generics, we don't need to typecast the object.
5. List<String> list = new ArrayList<String>();
6. list.add("hello");
7. 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.

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

Syntax to use generic collection

```
1. ClassOrInterface<Type>
```

Example to use Generics in java

```
1. ArrayList<String>
```

Full Example of Generics in Java

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

```
1. import java.util.*;
2. class TestGenerics1{
3. public static void main(String args[]){
4. ArrayList<String> list=new ArrayList<String>();
5. list.add("rahul");
6. list.add("jai");
7. //list.add(32);//compile time error
8.
9. String s=list.get(1);//type casting is not required
10. System.out.println("element is: "+s);
11. Iterator<String> itr=list.iterator();
12. while(itr.hasNext()){
13. System.out.println(itr.next());
14. }
15. }
16. }
```

```
1. import java.util.*;
2. class TestGenerics1{
3. public static void main(String args[]){
4. ArrayList<String> list=new ArrayList<String>();
5. list.add("rahul");
6. list.add("jai");
7. //list.add(32);//compile time error
8.
9. String s=list.get(1);//type casting is not required
10. System.out.println("element is: "+s);
11.
12. Iterator<String> itr=list.iterator();
13. while(itr.hasNext()){
14. System.out.println(itr.next());
15. }
16. }
17. }
```

Test it Now

Output:

```
element is: jai
rahul
jai
```

Example of Java Generics using Map

Now we are going to use map elements using generics. Here, we need to pass key and value. Let us understand it by a simple example:

```
1. import java.util.*;
2. class TestGenerics2{
3. public static void main(String args[]){
4. Map<Integer,String> map=new HashMap<Integer,String>();
5. map.put(1,"vijay");
6. map.put(4,"umesh");
7. map.put(2,"ankit");
8.
9. //Now use Map.Entry for Set and Iterator
10. Set<Map.Entry<Integer,String>> set=map.entrySet();
11.
12. Iterator<Map.Entry<Integer,String>> itr=set.iterator();
13. while(itr.hasNext()){
14. Map.Entry e=itr.next();//no need to typecast
15. System.out.println(e.getKey()+" "+e.getValue());
16. }
17.
18. }}
```

Test it Now

Output

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

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.

Let's see a simple example to create and use the generic class.

Creating a generic class:

```
1. class MyGen<T>{
```

```
2. T obj;
3. void add(T obj){this.obj=obj;}
4. T get(){return obj;}
5. }
```

The T type indicates that it can refer to any type (like String, Integer, and Employee). The type you specify for the class will be used to store and retrieve the data.

Using generic class:

Let's see the code to use the generic class.

```
1. class TestGenerics3{
2. public static void main(String args[]){
3. MyGen<Integer> m=new MyGen<Integer>();
4. m.add(2);
5. //m.add("vivek");//Compile time error
6. System.out.println(m.get());
7. }}
```

Test it Now

Output

```
2
```

Type Parameters

The type parameters naming conventions are important to learn generics thoroughly. The common type parameters are as follows:

1. T - Type
2. E - Element
3. K - Key
4. N - Number
5. V - Value