**Practical No. 1**

**Name:- Rohit Phatangare.**

**Roll No.:- 44.**

**Div:- A.**

**Aim**: **Assignments on Java Generics Program**

**1**:- Write a Java Program to demonstrate a Generic Class .

**Description: -** A **generic class** in Java is a class that can work with multiple data types while providing type safety. It enables you to define a class where the data type of a field, method, or parameter is specified as a parameter when the class is instantiated. This ensures that the data type is known at compile time and provides type checking at compile time.

1. **Type Safety:** Generic classes ensure type safety by allowing you to specify the data type at compile time. This helps catch type-related errors early in the development process.
2. **Code Reusability:** Generic classes promote code reusability by allowing you to write classes that can operate on a wide range of data types without duplicating code.
3. **Collections Framework:** Java s Collections Framework extensively uses generic classes. For example, ArrayList<T>, LinkedList<T>, and HashMap<K, V> are generic classes that can store elements of various types.

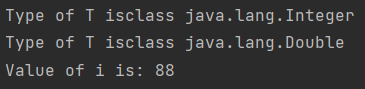
* **class GenericClass<T>** : This line declares a generic class called GenericClass , where <T> is a type parameter that represents a placeholder for the actual data type. You can replace T with any valid Java identifier (usually single uppercase letters like T , E , etc.) to represent the type.
* **T ob**; : This line declares a field named ob of type T , where T represents the generic type. This field will hold an object of the generic type.
* **GenericClass(T obj)** : This is the constructor of the generic class. It accepts an argument obj of type T and assigns it to the ob field.
* **T getObj()** : This method returns the object stored in the ob field. The return type of the method is T .
* **void showType()** : This method prints the type of the object stored in the ob field using the getClass() method. It demonstrates how you can work with the generic type and access its properties.

In your DemoGenericClass class, you instantiate GenericClass objects with specific data types ( Integer and Double ) and demonstrate how to use the generic class to store and retrieve values of those types. The type parameter <T> is replaced with Integer and Double when creating objects of GenericClass , allowing you to work with different data types while maintaining type safety.

**Code:-**

class GenericClass<T>  
{  
 T ob;//type T variable  
 GenericClass(T obj)  
 {  
 ob=obj;  
 }  
 T getObj() //int getObj()  
 {  
 return ob;//int i=objName.getObj();  
 }  
 void showType()  
 {  
 System.*out*.println("Type of T is"+ob.getClass());  
 }  
}  
  
public class DemoGenericClass  
{  
 public static void main(String[] args)  
 {  
//GenericClass g=new GenericClass();  
 GenericClass<Integer> iob=new GenericClass<Integer>(88);  
 GenericClass<Double> dob=new GenericClass<Double>(7.5);  
 iob.showType();//Integer  
 dob.showType();  
 Integer i=iob.getObj();  
 System.*out*.println("Value of i is: "+i);  
 }  
}

**Output:-**



**2**:- Write a Java Program to demonstrate Generic Methods.

**Description:-**

This Java code defines a class named genericMethod that contains a main method along with a static method named print. Heres a description of the code:

1. The class genericMethod is declared.

2. In the main method:

* An integer array i is declared and initialized with values {1, 2, 3, 4, 5} .
* There s a commented out line ( //System.out.println(i);) that appears to be an unused statement. This line is commented out, so it won’t affect the programs execution.
* The print method is called with the integer array i as an argument. This is done to print the elements of the array.

3. The print method is defined with the following characteristics:

* It’s a static method, which means it can be called without creating an instance of the genericMethod class.
* It takes an integer array list as a parameter.
* Inside the method, theres a for loop that iterates through the elements of the input array list .
* During each iteration of the loop, it prints the value of the current element followed by a space.
* After printing all elements, it prints an empty line to separate the output.

In summary, this code demonstrates a simple Java program that prints the elements of an integer array using a separate method called print. The print method takes an integer array as an argument and iterates through it, printing each element on a new line. The code in the main method initializes an integer array and then calls the print method to display its contents.

**Code:-**

class genericMethod{   
 public static void main(String args[]){   
 int[] i= {1,2,3,4,5};  
 *print*(i);  
 }   
 public static void print(int[] list){  
 for(int i=0;i<list.length;i++){  
 System.*out*.println(list[i]+" ");  
 }  
 System.*out*.println();  
 }  
}

**Output:-**



**3**:- Write a Java program to demonstrate Collection interface methods.

**Description:**

This Java code defines a class named TestCollectionMethods that demonstrates various operations on collections, specifically using ArrayLists and the methods provided by the Java Collections Framework. Here's a description of the code

**1**. In the **main** method:

* An ArrayList<String> named collection1 is created and initialized with four String elements: "NewYork," "London," "Mumbai," and "Delhi."
* The elements of collection1 are printed to the console using System.out.println .

**2**. A Collection<String> named collection2 is created as an ArrayList<String> . It is initialized with three String elements: "Pune," "Portland," and "Mumbai."

**3.** An ArrayList<String> named c is created, and its contents are initialized by copying all the elements from collection1 using the addAll method.

**4.** Another Collection<String> named c1 is created and initialized by cloning collection1 . This is done using a type cast because the clone method returns a generic Object type, so it is explicitly cast to ArrayList<String> .

**5.** The contents of c (a copy of collection1 ) are printed to the console.

**6.** The addAll method is used to add all elements from collection2 to c . Now, c contains a combination of elements from both collection1 and collection2 .

**7.** The contents of c after adding elements from collection2 are printed to the console.

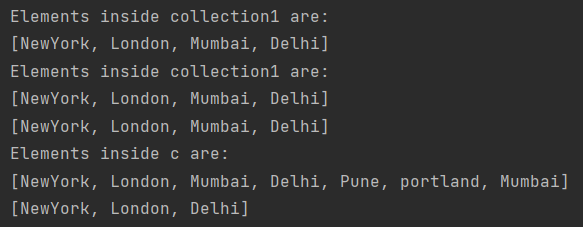
**8.** The removeAll method is used to remove all elements from c that are also present in collection2 . After this operation, c will only contain elements that were unique to collection1 .

In summary, this code demonstrates how to perform common collection operations such as copying elements from one collection to another, cloning a collection, adding all elements from one collection to another, and removing elements that are present in another collection. It uses ArrayLists and Collection methods to illustrate these operations.

**Code:-**

//WAP to demonstrate Collection interface methods  
import java.util.\*;  
public class TestCollectionMethods  
{  
 public static void main(String[] args)  
 {  
 ArrayList<String> collection1=new ArrayList<>();  
 collection1.add("NewYork");  
 collection1.add("London");  
 collection1.add("Mumbai");  
 collection1.add("Delhi");  
 System.*out*.println("Elements inside collection1 are:");  
 System.*out*.println(collection1);  
 Collection<String> collection2=new ArrayList<>();  
 collection2.add("Pune");  
 collection2.add("portland");  
 collection2.add("Mumbai");  
 //collection1.addAll(collection2);  
 ArrayList<String> c = new ArrayList<>();  
 c.addAll(collection1);  
 Collection<String> c1=(ArrayList<String>)(collection1.clone());  
 System.*out*.println("Elements inside collection1 are:");  
 System.*out*.println(c);  
 //c.retainAll(collection2);  
 System.*out*.println(c);  
 c.addAll(collection2);  
 System.*out*.println("Elements inside c are:");  
 System.*out*.println(c);  
 c.removeAll(collection2);  
 System.*out*.println(c);  
 }  
}

**Output:-**

****