Practical 1

Aim :- Implementation of bounded type, Generics, different collections interfaces and Lambda expressions

About Java generics :-

Generics means parameterized types. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, 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.

The Object is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature.

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, however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.

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. The classes, which accept one or more parameters, are known as parameterized classes or parameterized types.

Bounded Types with Generics

There may be times when you want to restrict the types that can be used as type arguments in a parameterized type. For example, a method that operates on numbers might only want to accept instances of Numbers or their subclasses. This is what bounded type parameters are for.
Sometimes we don’t want the whole class to be parameterized. In that case, we can create a Java generics method. Since the constructor is a special kind of method, we can use generics type in constructors too.

Suppose we want to restrict the type of objects that can be used in the parameterized type. For example, in a method that compares two objects and we want to make sure that the accepted objects are Comparables.

How to Declare a Bounded Type Parameter in Java?
List the type parameter’s name,
Along with the extends keyword
And by its upper bound. (which in the below example c is A.)

Syntax
<T extends superClassName>

Collections in Java

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects.

Iterator Interface

Iterator interface provides the facility of iterating the elements in a forward direction only.

ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types.

LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

HashMap

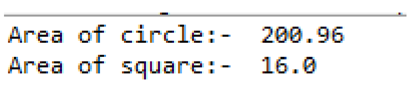
HashMap is a part of the Java collection framework. It uses a technique called Hashing. It implements the map interface. It stores the data in the pair of Key and Value. HashMap contains an array of the nodes, and the node is represented as a class. It uses an array and LinkedList data structure internally for storing Key and Value.

1. **Interface shape with method Area() create Circle and Square which implements Class Shape. Create a generic class BoundedShape that extends shape. And implement the generics and use area function accordingly.**

Code :-

```
package javagenerics;

import java.util.*;
interface shape{
	void area();
}

class circle implements shape
{
	public float radius;
	public circle(float radius) {

		this.radius = radius;
		}
 public void area(){
		System.out.println("Area of circle:-  " + radius * radius*3.14);
	}
}

class square implements shape
{
	public float radius;
	public square(float radius) {

		this.radius = radius;
		}
 public void area(){
		System.out.println("Area of square:-  " + radius * radius);
	}
}


class boundedshape <T extends shape>
{

	public T a;

	void findArea() {
		a.area();
		}
}

public class genericsbound {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
boundedshape<shape> b = new boundedshape<shape>();
		b.a = new circle(8);
		b.findArea();
		b.a = new square(4);
		b.findArea();
	}

}
```

Output :-



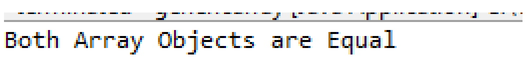
```
Area of circle:-  200.96
Area of square:-  16.0
```

2. **Create Generic class , contains array of generic type, write a method to calculate average of the members of the array, write a method to check whether the average of array of different object is equal or not.**

Code :-

```
package javagenerics;
import java.util.*;



class generic <T extends Number>
{
	T[] array1;



	public generic(T[] array)
	{
		array1 = array;
	}

	public double calculateAverage()
	{
		double total = 0;
		for(T e : array1)
		{

			total += e.intValue();
		}

		return total/array1.length;
	}


	public void equals(generic<Number> otherarray)
	{
		if(this.calculateAverage() == otherarray.calculateAverage())
		{
			System.out.println("Both Array Objects are Equal");


		}
		else
		{
			System.out.println("Both Array Objects aren't Equal");

		}
	}
}


public class genericarray {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Integer arr[] = {1,2,3,4,5};
		generic<Integer> obj1 = new generic<>(arr);
		Double arr1[] = {1.0,2.0,3.0,4.0,5.0};
		generic<Number> obj2 = new generic<>(arr1);
		obj1.equals(obj2);
	}

}
```

Output :-



```
Both Array Objects are Equal
```

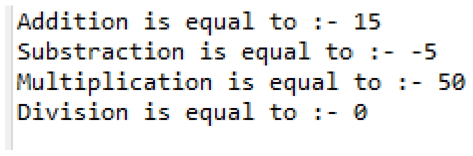
3. **Perform addition, subtraction, multiplication as well as division using Lambda Expression**

Code :-

```
package Practical1;
import java.util.*;
interface Mathfunc{
	public int arithfunc(int a,int b);

}
public class program5 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub


	Mathfunc add = (int a,int b) ->{
		return a+b;

	};

	Mathfunc sub = (int a,int b) ->{
		return a-b;

	};

	Mathfunc mul = (int a,int b) ->{
		return a*b;

	};

	 Mathfunc div = (int a,int b) ->{
		return a/b;

	};

	System.out.println("Addition is equal to :- " + add.arithfunc(5, 10));
	System.out.println("Substraction is equal to :- " + sub.arithfunc(5, 10));
```

```
      System.out.println("Multiplication is equal to :- " + mul.arithfunc(5, 10));
      System.out.println("Division is equal to :- " + div.arithfunc(5, 10));
       }

}
```

Output :-



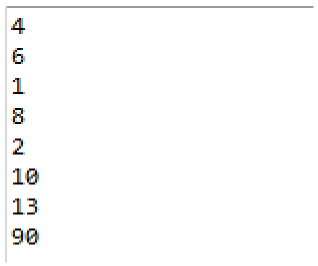
```
Addition is equal to :- 15
Substraction is equal to :- -5
Multiplication is equal to :- 50
Division is equal to :- 0
```

4. **Create an ArrayList of type Interger, add element into it traverse the arraylist and print the elements**

Code :-

```
package Practical1;
import java.util.*;

public class program1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
	 ArrayList<Integer> list = new ArrayList<Integer>();
	 list.add(4);
	 list.add(6);
	 list.add(1);
	 list.add(8);
	 list.add(2);
	 list.add(10);
	 list.add(13);
	 list.add(90);

Iterator<Integer> iterator = list.iterator();

		while(iterator.hasNext())
		{

			System.out.println(iterator.next());
		}


	}

}
```

Output :-



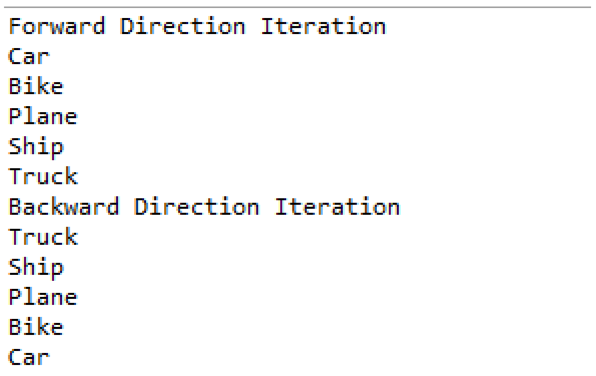
```
4
6
1
8
2
10
13
90
```

5. **Create a LinkedList of type String add 5 elements and traverse the list and from both sides**

Code :-

```
package Practical1;
import java.util.*;
public class program2 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
	  LinkedList<String> ll = new LinkedList<String>();
	  ll.add("Car");
	  ll.add("Bike");
	  ll.add("Plane");
	  ll.add("Ship");
	  ll.add("Truck");

	  ListIterator<String> listiterator = ll.listIterator();
	  System.out.println("Forward Direction Iteration");
	  while(listiterator.hasNext())
	  {
		  System.out.println(listiterator.next());
	  }

	  System.out.println("Backward Direction Iteration");
	  while(listiterator.hasPrevious())
	  {
		  System.out.println(listiterator.previous());
	  }
	}

}
```

Output :-



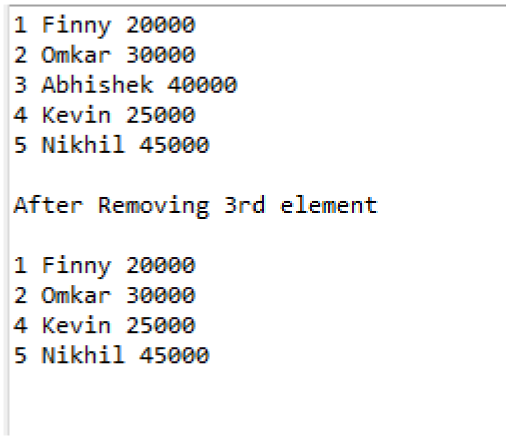
```
Forward Direction Iteration
Car
Bike
Plane
Ship
Truck
Backward Direction Iteration
Truck
Ship
Plane
Bike
Car
```

6. **Create an employee class (id, name, salary) create an Arralist of type employee, add 5 employee, traverse the ArrayList and print the elements, Remove one element and print the list**

Code :-

```
package Practical1;
import java.util.*;


class employee{
	int id;
	String name;
	int salary;

	employee(int id, String name, int salary)
	{
		this.id = id;
		this.name = name;
		this.salary = salary;
	}
}
public class program3 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		employee emp1 = new employee(1,"Finny",20000);
		employee emp2 = new employee(2,"Omkar",30000);
		employee emp3 = new employee(3,"Abhishek",40000);
		employee emp4 = new employee(4,"Kevin",25000);
		employee emp5 = new employee(5,"Nikhil",45000);

		ArrayList<employee> list = new ArrayList<employee>();
		list.add(emp1);
		list.add(emp2);
		list.add(emp3);
		list.add(emp4);
		list.add(emp5);

		Iterator<employee> iterator = list.iterator();

		while(iterator.hasNext())
		{
			employee emp = iterator.next();
			System.out.println(emp.id+" "+emp.name+" "+ emp.salary);
		}

		list.remove(2);
		System.out.println(" ");
		System.out.println("After Removing 3rd element");
		System.out.println(" ");
		Iterator<employee> iteratorl = list.iterator();
		while(iteratorl.hasNext())
		{
			employee empl = iteratorl.next();
			System.out.println(empl.id+" "+empl.name+" "+empl.salary);
		}

	}

}
```

Output :-



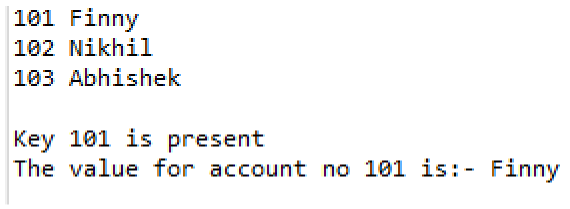
```
1 Finny 20000
2 Omkar 30000
3 Abhishek 40000
4 Kevin 25000
5 Nikhil 45000

After Removing 3rd element

1 Finny 20000
2 Omkar 30000
4 Kevin 25000
5 Nikhil 45000
```

7. **Create a class Customer(Account_no Integer, Name Sting), Create a HashMap of type Customer put elements, print elements, check if element with account number 101 is present or not? What is the value for Customer 101.**

Code :-

```
package Practical1;
import java.util.*;
import java.util.Map.Entry;
class customer
{
        int Account_no;
        String Name;

        customer(int a,String n)
        {
              Account_no = a;
              Name = n;
        }
}

public class program4 {

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            customer c1 = new customer(101,"Finny");
            customer c2 = new customer(102,"Nikhil");
            customer c3 = new customer(103,"Abhishek");

            HashMap<Integer,customer> map = new HashMap<>();
            map.put(c1.Account_no,c1);
            map.put(c2.Account_no,c2);
            map.put(c3.Account_no,c3);
            Iterator<Entry<Integer, customer>> entry = map.entrySet().iterator();
            while(entry.hasNext())
            {
                  Entry<Integer, customer> e= entry.next();
                  System.out.println(e.getValue().Account_no +" "+e.getValue().Name);
            }

            if(map.containsKey(101))
            {
                  System.out.println("");
                  System.out.println("Key 101 is present");
                  System.out.println("The value for account no 101 is:- " +
map.get(101).Name);
            }
            else
            {
                  System.out.println("The above entry is not present");
            }

      }

}
```

Output :-



```
101 Finny
102 Nikhil
103 Abhishek

Key 101 is present
The value for account no 101 is:- Finny
```