Experiment#		Student ID	
Date		Student Name	

2) Discuss the necessity of Interfaces in Generics.

Interfaces play a crucial role in generics, providing several benefits & enabling powerful programming techniques.

The necessity of Interfaces in Generics are:

i) Type Constraints

ii) Polymorphism

iii) Method Declarations

iv) Bounded Wild Cards

v) Generic Collections

vi) Generic Algorithms

In-Lab:

1) Write a Java Program to identify the Maximum Value and Minimum Value in the arrays of different datatypes like Integer, String, Character & float by incorporating the concept of Generics with interfaces.

Procedure/Program:

```
public interface MinMax <T extends Comparable <T>>{
    T find Max(T[]array);
    T find Min (T[] array);
}
Public class Int MinMax implements MinMax <Integer>{
  Public Integer find Max(Integer[]array){
   Integer max = array[0];
   for (Integer Value: array) {
    if(value. compare To (max) >0) {
        max =value;
   }
}
   return max;
 }
public Integer findMin(Integer [] array){
    Integer min =array[0];
    for (IntegerValue : array) {
       if(value.compareTo(min)<0) {
          min =value;
    } }
    retum min;
 }
}
```

```
public class MinMaxTest {
  public static void main(String[] args) {
    Integer[] intArray = {5,3,8,1,4};
    Integer MinMax int MinMax = new Integer MinMax();
    System.out.println("Max Integer: "+int MinMax.Find Max(intArray));
    System.out.println("Min Integer: "+intMin Max.Find Min(intArray));
  }
}
```

✓ **Data and Results:**

O/P:

Max Integer : 8

Min Integer : 1

✓ **Analysis and Inferences:**

By using MinMax Interface, Concrete classes & MinMax Test class we can compile & run this program successfully.

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) List the benefits of Generics.

Benefits of Generics :

i) Type Safety

ii) Code Reusability

iii) Elimination of Casting

iv) Improved performance.

2) Discuss about the various types of Generics implementation in Java.

i) public class Box<T>{

}

ii] Public interface Pair<K, V>{

K getKey();

V getValue();

}

3) Illustrate about "Type Parameter Naming Conventions" in Generics

When defining types parameters in java generics, some of the common connections are;

E: Element (collections)

K: Key (Maps)

V: Value (Maps)

T: Type (general cases)

N: Number (numeric types)

4) State about Generic Classes with example

Generics in java allow you to create classes, interfaces & methods that operate on types specified.

Ex:

```
public class Box <T> {
  private T item;
    - - - -
    - - - -
    - - - -
}
public static void main (String[] args) {
   Box <Integer> integerBox = new Box <>();
}
```

5) State about Generic Interfaces with example.

```
Public interface Pair <K, V>{
      K getkey();
      V getValue();
}
public class GenericInterfaceTest {
    public static void main (String[]args) {
      OrderedPair < Integer, String > P1 = new OrderedPair <>(1, "one");
System.out.println ("key :"+P1. getkey()+"; Value :"+P1-get
                                                  Value());
    }
}
```

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1) Create a generic class that implements a binary search algorithm. Test the class with different data types such as integers, doubles, and strings.

Procedure/Program:

```
import java.util.Arrays;

public class GenericBinarySearch <T extends Comparable <T> {

public int binarySearch(T[] ar, T key){

    int l=0;

    int r= ar.l-1;

    while (l<=r) {

       int m=l+(r-l)/2;

       if (ar[m].compareTo(key)==0) {

          retum m;

       }

       if(ar[m].compareTo(key)<0) {

          l=m+1;

       }else{

          r=m-1;

       }

}

 retum -1;

}

}
```

```
public class BinarySearchTest {
    public static void main(String[] args) {
        Integer[] i = {1,2,3,4,5,6,7,8,9};
Generic BinarySearch <Integer> is = new Generic Binary Search<>();
        int ik = 5;
        int iR = is.binarySearch(i, ik);
System.out.println("Integer"+ik+"Found at index:"+iR);
        Double[] d = {1.1, 2.2, 3.3, 4.4};
GenericBinarySearch<Double>ds = new GenericBinary Search<>();
        double dk = 3.3;
        double dR = ds.binarySearch(d,dk);
        system.out.println("Double"+dk+"found at index:"+dR);
    }
}
```

Experiment#		Student ID	
Date		Student Name	

2) Create a generic method that sorts an array of objects using a bubble sort algorithm. Test the method with different types of objects such as integers, doubles, and strings.

Procedure/Program:

```
import java.util.Arrays;
public class GenericBubbleSort{
    public static <T extends Comparable<T>> void bubbleSort(T[]
        array) {
    int n = array.length;
    T temp;
    boolean swapped;
for(int i=0; i<n-1; i++) {
     swapped = false;
for (int j=0; j<n-1-i; j++) {
     if(array[j].compareTo(array[j+1] >0) {
      temp = array[j];
      array[j] = array[j+1];
      array[j+1] = temp;
      swapped = true;
   }
} if(!swapped) {
      break;
}
}
}
```

Experiment#		Student ID	
Date		Student Name	

✓ **Data and Results:**

O/P:

original Int Array : [5, 3, 8, 1, 4]

Sorted : [1, 3, 4, 5, 8]

Original Double Array : [2.4, 1.6, 3.8, 5.5]

Sorted : [1.6, 2.4, 3.8, 5.5]

✓ **Analysis and Inferences:**

When the program is executed for 'Bubble Sort Test' the output will display original & sorted arrays for int & double data types.

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.