Experiment#		Student ID	
Date		StudentName	[@KLWKS_bot] THANOS

6.GenericswithClassesand Interfaces.

Aim/Objective: To analysethe implementationoftheconceptofGenericswithInterfaces for the real time scenario.

Description: ThestudentwillunderstandtheconceptofGenericswithInterfaces.

Pre-Requisites: ClassesandObjectsinJAVA

Tools: EclipseIDEforEnterpriseJavaandWebDevelopers

Pre-Lab:

1)Discussthe necessityofGenerics Interfaces.

- **Type Safety** – Prevents runtime type errors.
- **Code Reusability** – Works with multiple data types.
- **Consistency** – Reduces redundant code.
- **Scalability** – Useful for dynamic designs.

Example:

java Copy Edit

```java
interface Data<T> {
    void setData(T data);
    T getData();
}
```

In-Lab:

1) WriteaJavaProgramtoidentifytheMaximumValueandMinimumValueinthearrays of different datatypes like Integer, String, Character & float by incorporating the concept of Generics with interfaces.

Procedure/Program:

```
import java.util.Arrays;

interface MinMax<T> {
   T min(T[] array);
   T max(T[] array);
}


class MinMaxImpl<T extends Comparable<T>> implements MinMax<T> {
   public T min(T[] array) { return
Arrays.stream(array).min(T::compareTo).orElse(null); }
   public T max(T[] array) { return
Arrays.stream(array).max(T::compareTo).orElse(null); }
}


public class Main {
   public static void main(String[] args) {
      MinMax<Integer> intMinMax = new MinMaxImpl<>();
      System.out.println("Integer Min: " + intMinMax.min(new Integer[]{3, 5, 1,
4, 2}));
      System.out.println("Integer Max: " + intMinMax.max(new Integer[]{3, 5, 1,
4, 2}));


MinMax<String> stringMinMax = new MinMaxImpl<>();
      System.out.println("String Min: " + stringMinMax.min(new
String[]{"apple", "orange", "banana", "grape"}));
      System.out.println("String Max: " + stringMinMax.max(new
String[]{"apple", "orange", "banana", "grape"}));


MinMax<Character> charMinMax = new MinMaxImpl<>();
      System.out.println("Character Min: " + charMinMax.min(new
Character[]{'d', 'a', 'c', 'b'}));
      System.out.println("Character Max: " + charMinMax.max(new
Character[]{'d', 'a', 'c', 'b'}));
```

```
MinMax<Float> floatMinMax = new MinMaxImpl<>();
      System.out.println("Float Min: " + floatMinMax.min(new Float[]{3.5f, 2.1f,
4.8f, 1.2f}));
      System.out.println("Float Max: " + floatMinMax.max(new Float[]{3.5f, 2.1f,
4.8f, 1.2f}));
   }
}
```

OUTPUT

```
Integer Min: 1
Integer Max: 5
String Min: apple
String Max: orange
Character Min: a
Character Max: d
Float Min: 1.2
Float Max: 4.8
```

✓ **Dataand Results:**

Data

The program finds the minimum and maximum values from arrays.

Result

It correctly identifies the smallest and largest elements in arrays.

✓ **Analysisand Inferences:**

Analysis

Generic methods ensure flexibility, working with different data types.

Inferences

Using generics enhances reusability and simplifies comparison operations efficiently.

VIVA-VOCEQuestions(In-Lab):

1) ListthebenefitsofGenerics.

- **Type Safety**: Prevents runtime errors by enforcing type restrictions.
- **Code Reusability**: Allows generic classes/methods.
- **No Type Casting**: Eliminates explicit casting.
- **Compile-Time Checking**: Catches errors at compile time.
- **Improved Performance**: Reduces unnecessary casting.

2) DiscussaboutthevarioustypesofGenerics implementationinJava.

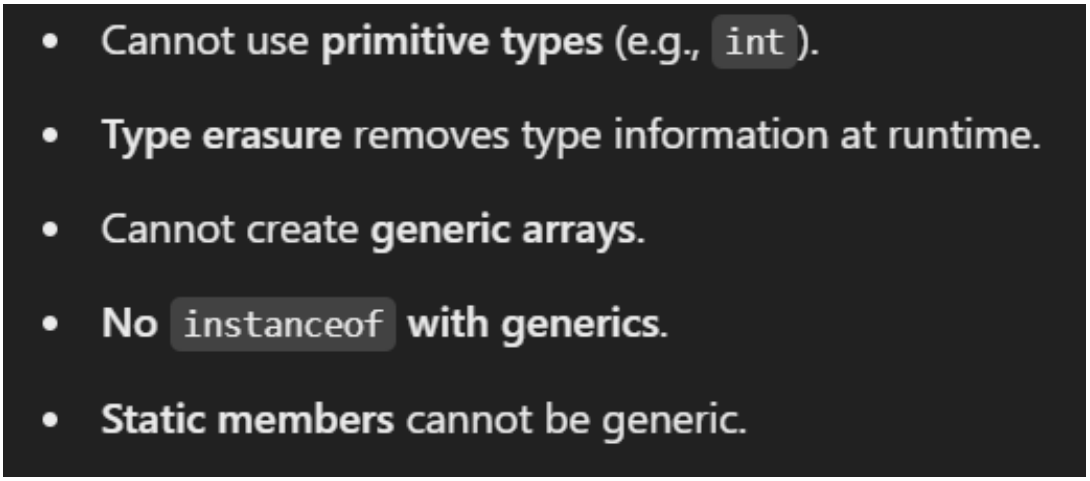
- **Generic Classes**: Classes with type parameters (e.g., `class Box<T>`).
- **Generic Methods**: Methods that use type parameters (e.g., `<T> void print(T value)`).
- **Bounded Types**: Restricts types (e.g., `T extends Number`).
- **Wildcards**: Flexible type (e.g., `List<? extends Number>`).

3) Illustrateabout "TypeParameterNamingConventions"inGenerics

- T: Type
- E: Element (collections)
- K, V: Key, Value (maps)
- N: Number

4) Statethesignificanceofdiamond(<>)operatoringenerics.

5) Listthelimitationsofgenerics.



Post-Lab:

1) Create a generic method that sorts an array of objects using a bubble sort algorithm. Test the method with different types ofobjects 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;
    for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
        if (array[j].compareTo(array[j + 1]) > 0) {
          T temp = array[j];
          array[j] = array[j + 1];
          array[j + 1] = temp;
        }
      }
    }
  }

  public static void main(String[] args) {
    Integer[] intArray = {5, 3, 8, 1, 2};
    Double[] doubleArray = {5.5, 3.3, 8.8, 1.1, 2.2};
    String[] stringArray = {"Banana", "Apple", "Cherry", "Date"};
```

```
        bubbleSort(intArray);
        bubbleSort(doubleArray);
        bubbleSort(stringArray);

        System.out.println("Sorted Integer Array: " + Arrays.toString(intArray));
        System.out.println("Sorted Double Array: " +
Arrays.toString(doubleArray));
        System.out.println("Sorted String Array: " + Arrays.toString(stringArray));
    }
}
```

OUTPUT

```
Sorted Integer Array: [1, 2, 3, 5, 8]
Sorted Double Array: [1.1, 2.2, 3.3, 5.5, 8.8]
Sorted String Array: [Apple, Banana, Cherry, Date]
```

Experiment#		Student ID	
Date		StudentName	[@KLWKS_bot] THANOS

✓ **Dataand Results:**

Data

The program sorts arrays of different data types using generics.

Result

It successfully sorts integer, double, and string arrays correctly.

✓ **Analysisand Inferences:**

Analysis

Bubble sort repeatedly swaps adjacent elements until fully sorted.

Inferences

Using generics enables sorting flexibility across multiple data types.

EvaluatorRemark(ifAny):	**MarksSecured _____ outof50**
	SignatureoftheEvaluatorwithDate

EvaluatorMUSTaskViva-vocepriortosigningandposting marksforeach experiment.