WIA1002/WIB1002 Data Structure

Lab: Generics

1. Create a generic class called MyGeneric that accepts one parameter. Declare a variable called e for the type parameter. Create a no-arg constructor. Create a constructor that accepts one generic parameter. Create a setter and getter method for the generic type.

   Create a test program that creates two instances of generic class of type String called strObj and of type Integer called intObj. Set a value for each of these objects. Display these values using the getter method.

   Answer :

```
public class MyGeneric<E> {
    private E e;

    MyGeneric() {}

    MyGeneric(E e) {
       this.e = e;
    }

    public void set(E e) {
        this.e = e;
    }

    public E get() {
        return e;
    }

   public static void main(String[] args) {
       MyGeneric<String> strObj = new MyGeneric<>();
       MyGeneric<Integer> intObj = new MyGeneric<>();
       strObj.set("Java");
       intObj.set(123);
       System.out.println(strObj.get());
       System.out.println(intObj.get());
   }
}
```

2. In a class called CompareMax, create a generic static method called maximum where the generic type extends the Comparable interface, which receives three parameters. Find the maximum of three values invoked by the main method.

Answer :

```
public class CompareMax {
   // determines the largest of three Comparable objects
   public static <T extends Comparable<T>> T maximum(T x, T y, T z)
   {
      T max = x; // assume x is initially the largest
      if ( y.compareTo( max ) > 0 ){
         max = y; // y is the largest so far
      }
      if ( z.compareTo( max ) > 0 ){
         max = z; // z is the largest now
      }
      return max; // returns the largest object
   }
   public static void main( String args[] )
   {
      System.out.printf( "Max of %d, %d and %d is %d\n\n",
                  3, 4, 5, maximum( 3, 4, 5 ) );

       System.out.printf( "Maxm of %.1f,%.1f and %.1f is %.1f\n\n",
                  6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ) );

       System.out.printf( "Max of %s, %s and %s is %s\n","pear",
         "apple", "orange", maximum( "pear", "apple", "orange" ) );
   }
}
```

3. a) Modify the following program to become a generic class.

```
public class StorePair {
   private int first, second;

   public StorePair(int first, int second) {
       this.first = first;
       this.second = second;
   }

   public int getFirst() {
       return first;
   }

   public int getSecond() {
       return second;
   }

   public void setPair(int first, int second) {
       this.first = first;
       this.second = second;
   }

   public String toString() {
       return "first = " + first + " second = " + second;
   }
}
```

b) Override the Object `equals()` method in the `StorePair` class to compare the first values of two objects for equality.

c) Have the `StorePair` class implement the Comparable interface. Override the `compareTo()`method to compare the first values of two objects.

d) Create a test program that creates three objects of the `StorePair` generic class called `a`, `b` and `c`. Set the first and second values of `a`, `b`, c as (6,4), (2,2), (6,3).

e) Invoke the `compareTo()`and `equals()` methods that compares the three objects created in (d) in the test program.

Answer:

```
//public class StorePairGeneric {
public class StorePairGeneric<E extends Comparable<E>>
   implements Comparable<StorePairGeneric<E>> {

//private int first, second;
private E first, second;

   //public StorePairGeneric(int first, int second){
   public StorePairGeneric(E first, E second){
         this.first = first;
         this.second = second;
   }

   public E getFirst(){
         return first;
   }

   public E getSecond(){
         return second;
   }

   public void setPair (E first, E second){
         this.first = first;
         this.second = second;
   }

   //b) Override the Object equals() method in the StorePair class
   @Override
   public boolean equals(Object o) {
         StorePairGeneric<E> other = (StorePairGeneric<E>) o;
         return this.first.equals(other.first);
   }

   //c) Have the StorePair class implement the Comparable interface.
   //c) Override the compareTo()method to compare the first values
   @Override
   public int compareTo(StorePairGeneric<E> o) {
         return this.first.compareTo(o.first);
   }

   @Override
   public String toString(){
         return "first = " + first + ", second = " + second;
   }

   public static void main(String[] args) {
         //d) Create a test program that creates three objects of the
StorePairGeneric class
         StorePairGeneric<Integer> a = new StorePairGeneric<>(6,4);
```

```
        StorePairGeneric<Integer> b = new StorePairGeneric<>(2,2);
        StorePairGeneric<Integer> c = new StorePairGeneric<>(6,3);

        //e) Invoke the compareTo()and equals() methods that compares
the  first value of three objects
        System.out.println(a.toString());
        System.out.println(b.toString());
        System.out.println(c.toString());

        System.out.println(a.compareTo(b)); //a > b, it returns 1
        System.out.println(a.compareTo(c)); //a == c, it returns 0
        System.out.println(b.compareTo(c)); //b < c, it returns -1

        System.out.println(a.equals(b)); //return false
        System.out.println(a.equals(c)); //return true
        System.out.println(b.equals(c)); //return false

   }

}
```

4. Provide a declaration and implementation of the generic method minmax() that takes in an array of generic type and returns a string with the following format: Min = <minValue> Max = <maxValue>. For instance,

   Integer[] intArr = {5, 3, 7, 1, 4, 9, 8, 2};
   String[] strArr = {“red”, “blue”, “orange”, “tan”};

   String intStr = minmax(intArr); //intStr = “Min = 1 Max = 9
   String str = minmax(strArr); //str= “Min = blue Max = tan

   *Hint : use Comparable interface to compare the values

   Answer :

```
public class MinMax {
    public static void main(String[] args) {
        Integer[] intArray = {5,3,7,1,4,9,8,2};
        String[] strArray = {"red", "blue", "orange", "tan"};

        String intStr = minmax(intArray);
        System.out.println(intStr);

        String strStr = minmax(strArray);
        System.out.println(strStr);
    }

    public static <E extends Comparable<E>> String minmax(E[] array) {
        E min = array[0];
        E max = array[0];

        for(int i=0; i<array.length; i++) {
            if(min.compareTo(array[i])>0)
                min = array[i];
            if(max.compareTo(array[i])<0)
                max = array[i];
        }
        return "Min = " + min + " Max = " + max;
    }
}
```

5. Create a class called FindMax that contains the following:

   Create a Circle class that uses the Comparable interface. Declare the radius variable and a single parameterized constructor that accepts this variable.

   In your main program, create 3 different objects of type array for integers that stores the following values, 1,2,3; a list of string that stores red, green, blue and a circle object of radius 3, 2.9 and 5.9. Invoke the max method as below:

   ```
   public static <E extends Comparable<E>> E max(E[] list)
   ```

   The max method above returns the maximum value in an array.

Answer:

```
public class FindMax {
  public static void main(String[] args) {
    Integer[] numbers = {1, 2, 3};
    System.out.println(max(numbers));

    String[] words = {"red", "green", "blue"};
    System.out.println(max(words));

    Circle[] circles = {new Circle(3), new Circle(2.9), new
Circle(5.9)};
    System.out.println(max(circles));
  }

  static class Circle implements Comparable<Circle> {
    double radius;

    public Circle(double radius) {
      this.radius = radius;
    }

    @Override
    public int compareTo(Circle c) {
      if (radius < c.radius)
        return -1;
      else if (radius == c.radius)
        return 0;
      else
        return 1;
    }

    @Override
    public String toString() {
      return "Circle radius: " + radius;
    }
  }

  public static <E extends Comparable<E>> E max(E[] list) {
    E max = list[0];
```

```
    for (int i = 1; i < list.length; i++) {
      if (max.compareTo(list[i]) < 0) {
        max = list[i];
      }
    }

    return max;
  }
}
```

6. In a class called `MinMaxTwoDArray`, write two generic methods:
   a. First method returns the minimum element in a two-dimensional array. Below is the method signature:

      ```
      public static <E extends Comparable<E>> E min(E[][] list)
      ```

   b. Second method returns the maximum element in a two-dimensional array. Below is the method signature:

      ```
      public static <E extends Comparable<E>> E max(E[][] list)
      ```

   c. Create a test program that creates one instance of generic class of type Integer called `numbers` with the elements: {4, 5, 6}, {1, 2, 3}. Display the minimum and maximum elements using the `min` and `max` methods.

Answer:

```
public class MinMaxTwoDArray {

	//Write a generic method that returns the minimum element //in
	a two-dimensional array.
	public static <E extends Comparable<E>> E min(E[][] list){
		E min = list[0][0];

		for (int i = 0; i < list.length; i++) {
			for (int j = 0; j < list[i].length; j++) {
				if (min.compareTo(list[i][j]) > 0) {
				min = list[i][j];
				}
			}
		}

		return min;
	}


	//Write a generic method that returns the maximum element //in
	a two-dimensional array.
	public static<E extends Comparable<E>> E max(E[][] list) {
		E max = list[0][0];

		for (int i = 0; i < list.length; i++) {
			for (int j = 0; j < list[i].length; j++) {
				if (max.compareTo(list[i][j]) < 0) {
				max = list[i][j];
				}
			}
		}

		return max;
	}
```

```
	public static void main(String[] args) {
		//create an object of 2D array for integers
		Integer[][] numbers = { {4, 5, 6}, {1, 2, 3} };

		//test the min and max generic methods
		System.out.println(min(numbers));
		System.out.println(max(numbers));

	}

}
```