PNG UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

CS220—Programming IV

SEMESTER 2 EXAMINATION

DATE: 27Th OCTOBER 2021

TIME ALLOWED: 3 HOURS

INSTRUCTIONS

1. Write your name and student number clearly on the front of the answer booklet.
2. There are seven (7) questions in the Examination Booklet.
3. You have 10 minutes to read this paper. Please **do not** write during this time.
4. All answers must be written in the examination answer booklet.
5. Attempt All Questions.
6. Notes, mobile phones, textbooks and dictionaries are NOT allowed in this exam.
7. Use blue or black ink—not pencil or red ink.
8. **Do not write** anything or **turn this page** until you are told to do so.

Marking Scheme

Questions	Marks
Question 1	/8
Question 2	/16
Question 3	/10
Question 4	/10
Question 5	/20
Question 6	/10
Question 7	/26
Total	**/100**

Question 1: [2 + 6 = 8 marks]

a) What is a unit test in software development?

b) The *max* method returns maximum number from a list of numbers.
Create a test class called `TestClass.java` and write one test case (junit test) for the method.

```
public int max(int[] list) {
    if(list.length==0) {
        return 0;
    }
    int max=list[0];
    for(int n:list) {
        if(n>max) {
            max=n;
        }
    }
    return max;
}
```

Question 2: [10 + 6 = 16 marks]

For this question refer to the following code. The class `C.java` has a generic `max` method that returns maximum element from a 2-dimensional array. Note that the *`compareTo()`* method of the form *`p.compareTo(q)`* returns -1 if `p<q` and it returns 1 if `p>q`.

```
import java.util.ArrayList;

public class C{

    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++) {
                E e=list[i][j];
                if(max.compareTo(e)<0) {
                    max=e;
                }
            }
        }
        return max;
    }
}
```

Below is the test class for the class `C.java`.

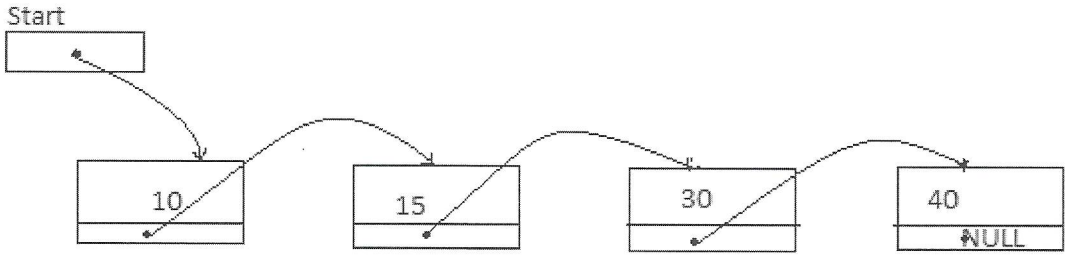
```
import static org.junit.Assert.*;

public class UnitTest {
@Test
public void testMaxNumbers() {
        Integer[][] x= {{1,3,10,0},{9,8,2,7}};
        Integer ex=new Integer(10);
        assertEquals(ex,C.max(x)); //1
        assertEquals(C.max(x),ex); //2
        assertFalse(C.max(x)>0);   //3
        assertTrue(0==C.max(x));   //4
        assertNotEquals(C.max(x),new Integer(5)); //5

    }
}
```

a) For each assert statement given in the test class, state whether it will pass or fail. Also state whether each assert statement is valid or not. If not valid then correct it.

b) The assertions given in the test method `testMaxNumbers` is for testing array of numbers. Since the method is generic, you can also pass in 2-dimensional array of any reference types. Now, write a test method called `testMaxString`. Define 2-dimensional String array in the method and initialize it will some values and write one test case using assert statements.

Question 3: [4 + 6 = 10 marks]

a) Here is a linked list containing integers in sorted fashion.



Explain the steps involved is inserting a new node with value 20 using block diagram.

b) For this question, refer to program in ***Appendix A***. Implement the `addToLast()` method in line 52.

Question 4: [3 + 2 + 5 = 10 marks]

a) What are the three parts to GUI programming?
b) What is a container object in GUI programming?
c) Create a frame in Java with following properties:

```
Size: 200 pixels by 200 pixels.
Title: "New Form".
Visibility: The frame should be visible on the screen.
```

Question 5: [3+2+(4+5)+3+3=20 marks]

a) State the characteristics of a recursive method.
b) Explain the difference between a recursion and an iteration.
c) The `Fibonacci Series` begins with 0 and 1, and each subsequent number is the sum of the preceding two numbers as given below.

```
Series: 0  1  1  2  3  5  8  13  21  34  55  89 ...
Indexes:0  1  2  3  4  5  6  7   8   9   10  11
```

(i) Write a recursive definition for computing the Fibonacci Series.
(ii) Write a recursive method for the recursive definition in (i) and identify the base case.

d) Show the output of the following program.

```
public class Test {
public static void main(String[] args) {
    computeN(10);
}
public static void computeN(int n) {
if (n > 0) {
        computeN(n - 1);
        System.out.print(n + " ");
        }
    }
}
```

e) What is wrong with the following program?

```
public class Test {
 public static void main(String[] args) {
   computeD(1234567);
 }

 public static void computeD(double n) {
  if (n != 0) {
     System.out.println(n);
     computeD(n / 10);
   }
  }
}
```

Question 6: [2 + 2 + 2 + 2 + 2= 10 marks]

a) What is the key benefit of using generics in Java?

b) What is erasure? Why Java generics are implemented using erasure?

c) You can use a generic class without specifying a concrete type like this:

```
ArrayList list=new ArrayList();
```

What is the equivalent declaration of the code using generic type?

d) What is wrong in the following code?

```
ArrayList dates = new ArrayList();
dates.add(new Date());
Date date = dates.get(0);
```

e) What is a bounded type?

Question 7: [2 + 2 + 2 + 4 + 5 + 3 + 8 = 26 marks]

For this question, refer to program in ***Appendix B***. You are expected to write generic methods to add shapes to a shape basket from which information about each shape instances stored and can be retrieved. The shape basket would hold different types of shapes such as Circles, Rectangles, Triangles, and Quadrilaterals etc. The shapes can be regular shapes or irregular shapes as given in Appendix B.

Hint: Use generic `ArrayList` class to store the shape objects and use the wildcard sub-typing where necessary.

a) Explain if this code fragment is valid or not.

```
Shapes<RegularShape> rShape=new Shapes<RegularShape>();
rShape.shape=new Triangle(1,2,3,"Triangle 1");
```

b) In the `ShapeBasket.java` class, implement the method signature below.

```
public void addShapesToBasket(<argument>)
```

This method will be used to insert shapes into the basket from the `ShapeTest.java` class as given in lines 10 and 34. The method should allow any shapes to be passed as an argument through the generic type `Shapes<T>` class.

c) In the `ShapeBasket.java` class, implement the method signature below.

```
public void addRegularShapesToBasket(<argument>)
```

This method will be used to insert shapes into the basket from the `ShapeTest.java` class as given in lines 17, 22 and 29. The method should allow only regular shapes to be passed as an argument through the generic type `Shapes<T>` class.

d) When you pass an instance of generic type `Shapes<Triangle>` as an argument in the method in (c), it will give and error. However, if you pass the argument to method in (b), there will be no error. Explain why the error happens.

e) Implement the following method in `ShapeBasket.java` class.

```
public void getStatistics().
```

This method will print name, area and the color of each shape stored in the basket, when invoked from the `ShapeTest.java` class as given in line 36. The output would be given like this:

```
Shapes..............................
Circle 2        314.000000      PINK
Circle 1        78.500000       RED
Rectangle 1     50.000000       White
Rectangle 2     250.000000      BLUE
Triangle 1      6.000000
```

.....END OF EXAMINATION.....

Appendix A: Linkedlist

```
1  package cs220_linklist;
2  public class LinkedList {
3      private Node head;
4      public LinkedList() {
5          head=null;
6      }
7      //This is the inner class for Node class
8              private class Node {
9              private String item;
10             private int count;
11             private Node next;//A node contains reference to another node
12         //Constructors
13             public Node() {
14                 next=null;
15                 item=null;
16                 count=0;
17             }
18             public Node(String newItem,int newCount,Node linkValue) {
19                 setData(newItem,newCount);
20                 next=linkValue;
21             }
22             //Mutator methods
23             public void setData(String newItem,int newCount) {
24                 item=newItem;
25                 count=newCount;
26             }
27             public void setLink(Node newLink) {
28                 next=newLink;
29             }
30             //Accessor methods
31             public String getItem() {
32                 return item;
33             }
34             public int getCount() {
35                 return count;
36             }
37             public Node getLink() {
38                 return next;
39             }
40         }//End of inner Node class
```

```
42      * Add a node at the start of the list with specified data
43      * The added node will be the first node in the linked list
44      * */
45-     public void addToStart(String itemName,int itemCount) {
46          head=new Node(itemName,itemCount,head);
47      }
48-     /*
49      * Add a node at the end of the list with specified data
50      * The added node will be the last node in the linked list
51      * */
52-     public void addToLast(String itemName,int itemCount) {
53          //Implement your logic here
54      }
55-     /*
56      * Removes the head node and returns true if  the list contains at least on node.
57      * Return false if the list was empty
58      * */
59-     public boolean deleteHeadNode() {
60          if(head!=null) {
61              head=head.getLink();
62              return true;
63          }
64          return false;
65      }
66
67-     public boolean contains(String item) {
68          return find(item)!=null;
69      }
70
71-     private Node find(String item) {
72          Node position=head;
73          String itemAtPosition;
74          while(position!=null) {
75              itemAtPosition=position.getItem();
76              if(itemAtPosition.equals(item))
77                  return position;
78              position=position.getLink();
79          }
80          return null;
81      }
82  }
```

Usage of the *addToLast()* method in main class.

```
2  public class LinkedListDemo  {
3-     public static void main(String[] args) {
4          LinkedList  list1=new LinkedList();
5          list1.addToLast("Lenz",1);
6          list1.addToStart("Mary", 2);
7          list1.addToStart("Peter", 3);
8          list1.addToStart("Tom", 4);
9          list1.addToStart("Tim", 5);
10         list1.addToStart("Juie", 6);
11         list1.addToLast("Terita",7);
12     }
13 }
```

Appendix B: Generic Shapes

Shapes.java

```
public class Shapes<T> {
    public T shape;
}
```

ShapeBasket.java

```
import java.util.ArrayList;

public class ShapeBasket {

}
```

RegularShape.java

```
public class RegularShape {
    private String color="White";
    private boolean isFilled;
    public void setColor(String color) {
        this.color=color;
    }
    public void setFilled(boolean isFilled) {
        this.isFilled=isFilled;
    }
    public boolean isFilled() {
        return isFilled;
    }
    public String getColor() {
        return color;
    }
}
```

Circle.java

```
public class Circle extends RegularShape{
    private String shapeName="Circle";
    private double radius=1;
    private double PI=3.14;

    public Circle(double radius,String shapeName) {
        this.radius=radius;
        this.shapeName=shapeName;
    }
    public String getShapeName() {
        return shapeName;
    }
    public double getRadius() {
        return radius;
    }
    public double getArea() {
        return PI*radius*radius;
    }
    public double getPerimeter() {
        return 2*PI*radius;
    }
}
```

Rectangle.java

```
public class Rectangle extends RegularShape{
    private String shapeName="Rectangle";
    private double height=1;
    private double width=1;

    public Rectangle(double height,double width,String shapeName) {
        this.height=height;
        this.width=width;
        this.shapeName=shapeName;
    }
    public String getShapeName() {
        return shapeName;
    }
    public double getArea() {
        return width*height;
    }
    public double getPerimeter() {
        return 2*(height+width);
    }
}
```

Triangle

```
public class Triangle {
    private String shapeName="Triangle";
    private double side1=1;
    private double side2=1;
    private double side3=1;
    public Triangle(double side1,double side2,double side3,String shapeName) {
        this.side1=side1;
        this.side2=side2;
        this.side3=side3;
        this.shapeName=shapeName;
    }
    public String getShapeName() {
        return shapeName;
    }
    public double getArea() {
        double s1, s2, s3, s4, area;
        s4 = (side1 + side2 + side3 )/ 2 ;
        area = Math.sqrt(s4 * (s4 - side1) * (s4 - side2) * (s4 - side3));
        return area;
    }
    public double getPerimeter() {
        return side1+side2+side3;
    }
}
```

```
1  package shape;
2  public class ShapeTest {
3      public static void main(String[] args) {
4          ShapeBasket sBasket=new ShapeBasket();
5          Shapes<Circle> circleShape1=new Shapes<>();
6          Circle c1=new Circle(5,"Circle 1");//Create Circle 1
7          c1.setColor("RED");
8          c1.setFilled(true);
9          circleShape1.shape=c1;
10         sBasket.addShapesToBasket(circleShape1);
11         Shapes<Circle> circleShape2=new Shapes<>();
12
13         Circle c2=new Circle(10,"Circle 2");//Circle 2
14         c2.setColor("PINK");
15         c2.setFilled(true);
16         circleShape2.shape=c2;
17         sBasket.addRegularShapesToBasket(circleShape2); //Add circle 2 to basket
18
19         Shapes<Rectangle> rectangeShape1=new Shapes<Rectangle>();
20         Rectangle rectangle1=new Rectangle(5,10,"Rectangle 1");//Rectangle 1
21         rectangeShape1.shape=rectangle1;
22         sBasket.addRegularShapesToBasket(rectangeShape1); //Add Rectangle 1 to basket
23
24         Shapes<Rectangle> rectangeShape2=new Shapes<Rectangle>();
25         Rectangle rectangle2=new Rectangle(5,50,"Rectangle 2");//Rectangle 2
26         rectangle2.setColor("BLUE");
27         rectangle2.setFilled(true);
28         rectangeShape2.shape=rectangle2;
29         sBasket.addRegularShapesToBasket(rectangeShape2);  //Add Rectangle 2 to basket
30
31         Shapes<Triangle> triangleShape1=new Shapes<Triangle>();
32         Triangle triangle1=new Triangle(3,4,5,"Triangle 1");//Triangle 1
33         triangleShape1.shape=triangle1;
34         sBasket.addShapesToBasket(triangleShape1); //Add Triangle 1 to basket
35         System.out.printf("Shapes.....\n");
36         sBasket.getStatistics();
37     }
38 }
39
```