NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM TEST FOR CS1020
AY2014/15 Semester 2

CS1020 – Data Structures and Algorithms I

4 March 2015 Time allowed: 1 hour 30 minutes

INSTRUCTIONS TO CANDIDATES

1. This test paper consists of **ELEVEN (11)** questions and comprises **THIRTEEN (13)** printed pages.
2. This is a **CLOSED BOOK** test. You are allowed to bring in ONE (1) piece of **handwritten** (not printed or photocopied) A4 double-sided reference sheet.
3. Calculators and other electronic devices are not allowed.
4. Answer all questions only on the **ANSWER SHEETS** provided.
5. We will collect only the Answer Sheets from you.
6. The total mark for this paper is **30**.

Questions 1 - 5: For each of these multiple-choice questions, only one answer is correct. Each correct answer is worth one mark, and there is no penalty for wrong answers.

1. What is the output of the following code fragment?

```
String str = 20/3 + "-CS1020-" + 1 + 23;
System.out.println(str);
```

(A) **20/3-CS1020-123**

(B) **6.666666666666667-CS1020-123**

(C) **6-CS1020-123**

(D) **6-CS1020-24**

(E) None of the above.

2. Given the following overloaded methods:

```
public static int m(int a, int b) {
   return a * b;
}

public static int m(int a, double b) {
   return a + (int) b;
}

public static int m(double a, int b) {
   return (int) a – b;
}
```

The call **m(3.0, 5.0)** is made. Which of the following statements correctly describes what would happen?

(A) The call returns the value **15** because none of the methods matches the call, so the first method is used.

(B) The call returns the value **8** because the first closest match is used.

(C) The call returns the value **-2** because the last closest match is used.

(D) The call throws a MismatchedMethodException.

(E) The call causes a compilation error.

3. Which of the following statements are true?

   i. Java supports multiple inheritance.
   ii. In Java, a class may implement multiple interfaces.
   iii. You must explicitly define at least one constructor for every service class you create.
   iv. A subclass does not inherit its superclass' constructor.

   (A) Only (i) and (iii)
   (B) Only (ii) and (iv)
   (C) Only (iii) and (iv)
   (D) Only (i), (iii) and (iv)
   (E) Only (ii), (iii) and (iv)

4. What is the output of the following program?

```
public class Q4 {

   public static void main(String[] args) {
      try {
         System.out.println("Hello-" + 3 / 0);
      }
      catch (ArithmeticException e) {
         System.out.println("World" + e.getMessage());
      }
      finally {
         System.out.println("Bye!");
      }
   }

}
```

   (A) **Hello-**
   (program crashes)

   (B) **World/ by zero**

   (C) **World/ by zero**
   Bye!

   (D) **Hello-**
   World/ by zero
   Bye!

   (E) **Hello-3**
   World/ by zero
   Bye!

5. Study the following partial program:

```
class Triplet <S,T> {
  private S first;
  private T second;
  private S third;

  public Triplet(S a, T b, S c) {
    first = a; second = b; third = c;
  }

  public S getFirst() { return first; }
  public T getSecond() { return second; }
  public S getThird() { return third; }
}

public class Q5 {
  public static void main(String[] args) {
    ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐
    │                                                                                     │
    │                                                                                     │
    └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘
    System.out.println(t.getFirst().getSecond()
                       + ", " + t.getSecond().getThird());
  }
}
```

Which of the following statements can be placed in the dotted box above so that the program can run without errors?

(A)
```
Triplet<String,Double> t
    = new Triplet<String,Double>("Red", 1.23, "Round");
```

(B)
```
Triplet<Integer,Triplet> t
    = new Triplet<Integer,Triplet>(12,
          new Triplet<Integer,Integer>(3, 4, 5), 89);
```

(C)
```
Triplet<Triplet,Triplet> t
    = new Triplet<Triplet,Triplet>
         (new Triplet<Integer,String>(1, "A", 2),
          new Triplet<Double,Integer>(3, 4, 5),
          new Triplet<Integer,String>(6, "B", 7));
```

(D)
```
Triplet<Triplet,Triplet> t
    = new Triplet<Triplet,Triplet>
         (new Triplet<String,Double>("A", 1.23, "B"),
          new Triplet<Integer,Double>(2, 3.4, 5),
          new Triplet<Integer,String>(6, "D", 7));
```

(E) None of the above.

6. Write the output of the following program. [2 marks]

```
public class Q6 {
   public static void main(String[] args) {
      Integer[] arr = new Integer[5];
      arr[0] = 5;
      arr[1] = arr[0] * 0;
      for (Integer elem: arr) {
         if (elem.equals(0) && elem != null) {
              System.out.println(elem);
         }
      }
   }
}
```

7. Write the output of the following program. [3 marks]

```
class A {
   protected int a = 2;

   public void mA() { a++; }
}

class B extends A {
   protected int a = 3;
   protected int b = 5;

   public void mB() { b += a; }
}

class C extends B {
   protected int b = 11;
   protected int c = 13;

   public void mA() { c *= a; super.b++; }

   public void mC() { mA(); }
}

public class Q7 {
   public static void main(String[] args) {
      C obj = new C();
      obj.mC();
      System.out.println(obj.a + " " + obj.b + " "
                         + obj.c);
   }
}
```

8. Write the output of the following program. [3 marks]

```
class T {
   private static int value1 = 123;
   private int value2;

   public void increment() {
      value2 = value1++;
   }

   public int getValue1() {
      return value1;
   }

   public int getValue2() {
      return value2;
   }
}

public class Q8 {

   public static void main(String[] args) {
      T t1 = new T();
      T t2 = new T();
      t1.increment();
      t2.increment();

      System.out.println(t1.getValue1() + " " +
                         t1.getValue2() + " " +
                         t2.getValue1() + " " +
                         t2.getValue2());
   }

}
```

9. An **anagram** is a type of word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once. For example, “torchwood” can be rearranged into “doctorwho”. Other examples include “ticks” and “stick”, “iced” and “dice”, “fringe” and “finger”, “elbow” and “below”.

   However, the following are not anagrams: “mouse” and “mousse”, “teen” and “tent”, “resets” and “tester”.

   Given the incomplete method **areAnagrams(String word1, String word2)** below that returns true if word1 and word2 are anagrams of each other, or false otherwise, complete the code. You may assume that word1 and word2 contain only lower-case letters and no other characters. The part of the code that is given must be used and not to be changed. [3 marks]

```
private static boolean areAnagrams(String word1,
                                   String word2) {

   if (word1.length() != word2.length())
      return false;

   for (int i = 0; i < word1.length(); i++) {

      // to be completed

   }

   return (word2.length() == 0);

}
```

10. [5 marks]

A **Triangle** class has been defined for right-angled triangles whose perpendicular sides are parallel to the x- and y-axes. The attributes are the three vertices of a triangle and its area. Each vertex is a **Point** object with integer x- and y-coordinates.

The API of **Point** class is given in Appendix A.

An incomplete program is given below.

```
import java.awt.*;

class Triangle {
  private Point[] vertices;
  private double area;

  // Default triangle with vertices (0,0), (0,1) and (1,0)
  public Triangle() {
    // To be completed
  }

  public Triangle(int vertex1X, int vertex1Y,
                  int vertex2X, int vertex2Y,
                  int vertex3X, int vertex3Y) {
    // To be completed
  }

  public Triangle(Point vertex1, Point vertex2, Point vertex3) {
    vertices = new Point[3];
    vertices[0] = vertex1;
    vertices[1] = vertex2;
    vertices[2] = vertex3;
    area = computeArea();
  }

  public double getArea() {
    return area;
  }

  public String toString() {
    return "[(" + vertices[0].x + "," + vertices[0].y + "):" +
           "(" + vertices[1].x + "," + vertices[1].y + "):" +
           "(" + vertices[2].x + "," + vertices[2].y + "):" +
           "Area=" + area + "]";
  }

  // Return the area of this triangle
  private double computeArea() {
    // To be completed
  }
}
```

a. Fill in the body of the first constructor with a single statement. [1 mark]

b. Fill in the body of the second constructor with a single statement. [1 mark]

c. Fill in the body of **computeArea()**. You should not use any selection construct ('if' or 'switch'). You may use methods from some other class in the API. [3 marks]

11. [9 marks]

Using the **Triangle** class defined in the previous question, write a client class **TestTriangle** to perform the following:

a. Read a positive integer indicating the number of triangles to be created and added into an **ArrayList**. For each triangle, read the coordinates of its three vertices. You may assume that all data are valid. [5 marks]

b. After creating the **ArrayList** of triangles, look for the triangle with the smallest area and remove it from the **ArrayList**. If there are more than one triangle with the smallest area, remove the first one. [4 marks]

The API of **ArrayList** class is given in Appendix B.

An incomplete program is given below. You are to complete the **createList()** and **removeSmallestTriangle()** methods.

```
import java.util.*;

public class TestTriangle {

  private ArrayList<Triangle> triangles;

  public TestTriangle() {
    createList();
  }

  private void createList() {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter number of triangles: ");
    int num = sc.nextInt();

    // Fill in the code
  }

  private void removeSmallestTriangle() {
    // Fill in the code
  }

  public static void main(String[] args) {
    TestTriangle test = new TestTriangle();
    System.out.println(test.triangles);

    test.removeSmallestTriangle();
    System.out.println("After removing smallest triangle:");
    System.out.println(test.triangles);
  }
}
```

(See next page for sample run.)

Below is a sample run. Input data are shown in bold.

```
Enter number of triangles: 3
Enter 3 vertices: 3 10 3 1 12 1
Enter 3 vertices: -7 9 -4 9 -4 -1
Enter 3 vertices: 12 21 23 36 23 21
[[(3,10):(3,1):(12,1):Area=40.5], [(-7,9):(-4,9):(-4,-1):Are
a=15.0], [(12,21):(23,36):(23,21):Area=82.5]]
After removing smallest triangle:
[[(3,10):(3,1):(12,1):Area=40.5], [(12,21):(23,36):(23,21):A
rea=82.5]]
```

Appendix A: java.awt.Point

Attributes

```
public int x
public int y
```

Constructors

Point() Constructs and initializes a point at the origin (0, 0) of the coordinate space.
Point(int x, int y) Constructs and initializes a point at the specified (x,y) location in the coordinate space.
Point(**Point** p) Constructs and initializes a point with the same location as the specified Point object.

Methods

Modifier and Type	Method and Description
boolean	**equals**(Object obj) Determines whether or not two points are equal.
Point	**getLocation**() Returns the location of this point.
double	**getX**() Returns the X coordinate of this Point2D in double precision.
double	**getY**() Returns the Y coordinate of this Point2D in double precision.
void	**move**(int x, int y) Moves this point to the specified location in the (x,y) coordinate plane.
void	**setLocation**(double x, double y) Sets the location of this point to the specified double coordinates.
void	**setLocation**(int x, int y) Changes the point to have the specified location.
void	**setLocation**(**Point** p) Sets the location of the point to the specified location.
String	**toString**() Returns a string representation of this point and its location in the (x,y) coordinate space.
void	**translate**(int dx, int dy) Translates this point, at location (x,y), by dx along the x axis and dy along the y axis so that it now represents the point (x+dx,y+dy).

Appendix B: java.util.ArrayList <E>

Constructors

ArrayList() Constructs an empty list with an initial capacity of ten.
ArrayList(**Collection**<? extends **E**> c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity) Constructs an empty list with the specified initial capacity.

Methods

Modifier and Type	Method and Description
boolean	**add**(**E** e) Appends the specified element to the end of this list.
void	**add**(int index, **E** element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(**Collection**<? extends **E**> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	**addAll**(int index, **Collection**<? extends **E**> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(**Object** o) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
E	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(**Object** o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty**() Returns true if this list contains no elements.
Iterator<**E**>	**iterator**() Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(**Object** o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.

ListIterator<E>	**listIterator**() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	**listIterator**(int index) Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	**remove**(int index) Removes the element at the specified position in this list.
boolean	**remove**(**Object** o) Removes the first occurrence of the specified element from this list, if it is present.
boolean	**removeAll**(**Collection**<?> c) Removes from this
protected void	**removeRange**(int fromIndex, int toIndex) Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
boolean	**retainAll**(**Collection**<?> c) Retains only the elements in this list that are contained in the specified collection.
E	**set**(int index, **E** element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
List<E>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
void	**trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size.

== END OF PAPER ==