NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM TEST FOR CS1020
AY2013/14 Semester 2

CS1020 – Data Structures and Algorithms I

7 March 2014 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. Answer all questions only on the **ANSWER SHEETS** provided.
4. We will collect only the Answer sheets from you.
5. The total mark for this paper is **40**.

Questions 1 - 6: For each of these multiple-choice questions, there is only one correct answer. Each correct answer is worth one mark, and there is no penalty for wrong answers.

1. Which of the following statements is **false**?

   (A) A .java file may contain more than one class.

   (B) When you create an instantiable class, you must include a default constructor (a constructor with no parameter).

   (C) Not every class needs to have a main() method.

   (D) A class may contain a mix of class methods and instance methods.

   (E) All methods in the **Math** class are class methods.

2. Assuming that **list**, which is an **ArrayList**, contains at least 3 elements, and you want to shift its last element into its second position. For instance, if **list** originally contains [“A”, “B”, “C”, “D”, “E”], then after the shift it becomes [“A”, “E”, “B”, “C”, “D”].

   Which of the following statements to perform the shift is correct?

   (A) `list.add(list.remove(list.size()), 2);`

   (B) `list.add(list.remove(list.size()-1), 1);`

   (C) `list.add(2, list.remove(list.size()-1));`

   (D) `list.add(1, list.remove(list.size()));`

   (E) `list.add(1, list.remove(list.size()-1));`

3. Given the following interface **I**:

```
public interface I {
   public double f(int a);
   public int g(double b);
}
```

   The following class **F** implements the above interface **I**:

```
public class F implements I {

   public static double a = 2.5;

   public double f(int b) {
      return b * (b+1);
   }
}
```

   Compilation of the class **F** yields some error. What is/are the reasons?

   i. The class **F** cannot be public.

   ii. The statement “public static double a = 2.5;” must be removed.

   iii. The parameter **b** in **f()** must be named **a** instead.

   iv. The implementation of method **g()** is missing.

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

4. A mathematical model of the data objects that make up a data type as well as the set of operations defined on these objects is called a/an ________________________.

(A) Abstract Data Type
(B) Data Structure
(C) Primitive Data Type
(D) Encapsulation
(E) Algorithm

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

```
for (int i=1; i<=2; i++) {
   try {
      System.out.print("A");
      if (i == 1) throw new Exception();
      System.out.print("B");
      if (i == 2) throw new ArithmeticException();
      System.out.print("C");

   } catch (ArithmeticException e) {
      System.out.print("D");
      break;

   } catch (Exception e) {
      System.out.print("E");

   } finally {
      System.out.print("F");
   }
}
```

(A) **AEABD**
(B) **AEABDF**
(C) **AEFABD**
(D) **AEFABDF**
(E) **AEBCFABDCF**

6. In the program below, **story** is an array of strings containing 4 lines of text.

```
public class ArrayOfStrings {

   public static void main(String[] args) {
      String[] story =
      { "This is a little story about four people named ",
        "Everybody, Somebody, Anybody and Nobody.",
        "There was an important job to be done and ",
        "Everybody was sure that Somebody would do it." };

      int count = 0;
      for (int i=0; i<story.length; i++) {
         if ((story[i].indexOf("body") > -1) &&
             (story[i].indexOf("body")
                == story[i].lastIndexOf("body"))) {
            count++;
         }
      }
   }
}
```

Which of the following describes the result in the variable **count**?

(A) The number of times the word “body” appears in story.

(B) The number of lines where the word “body” does not appear at all.

(C) The number of lines where the word “body” appears exactly once.

(D) The number of lines where the word “body” appears at least once.

(E) The number of lines where the word “body” appears exactly twice.

7. Given the following overloaded method **m()** of a class **M**:

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

public static double m(double a, int b) {
   return a * b * 2;
}

public static double m(double a, double b) {
   return a * b * 3;
}
```

What is the output of the following statements? If there is an error, write "error" in your answer. [2 marks]

a) `System.out.println(M.m(4.0, 5));`

b) `System.out.println(M.m(4, 5.0));`

8. Given the following class **S**:

```
class S {
   private static int v1 = 0;
   private int v2, v3;

   public S() {
      this(0, 1);
   }

   public S(int x, int y) {
      this.v2 = x;
      this.v3 = y;
   }

   public static int getv1() {
      return v1;
   }

   public int getv2() {
      return v2;
   }

   public int getv3() {
      return v3;
   }

   public void m1(int v) {
      v1 += v;
   }

   public void m2(int v) {
      v2 += v;
      v3 += v2;
   }

   public int m3(int v) {
      int v3 = v;
      v3 += v2;
      return v3;
   }
}
```

8. *(continued…)*
What is the output of the following program? [6 marks]

```
public class TestS {
   public static void main(String[] args) {
      S s1 = new S();
      S s2 = new S(7, 8);

      s1.m1(5);
      s2.m1(5);
      s1.m2(5);
      int x = s2.m3(5);
      System.out.print(S.getv1() + ", ");
      System.out.print(s1.getv3() + ", ");
      System.out.print(s2.getv3() + ", ");
      System.out.println(x);
   }
}
```

9. Given the following class **Die**:

```
import java.util.*;

class Die {

   private Random num;

   public Die() {
      num = new Random();
   }

   public int toss() {
      return num.nextInt(6) + 1;
   }
}
```

Write a program **Game.java** to toss a die 1000 times and determine which value (1 through 6) has appeared the most frequently, and the number of times it appeared.

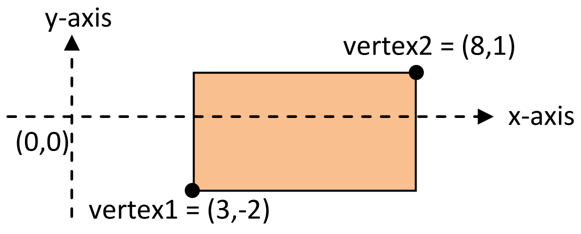
Below is a sample run of the program. If there are more than one value that appeared most frequently, report any of them. The skeleton program is given on the Answer Sheets and you are not supposed to change what is given in the code. [6 marks]

```
3 appeared most frequently, a total of 183 times.
```

10. [7 marks]

A **MyRect** class has been defined for rectangles whose sides are parallel to either the x-axis or y-axis. It consists of 2 attributes: **vertex1** and **vertex2** representing the south-west corner and north-east corner of a rectangle respectively. Both corners are objects of the **Point** class defined in Java API. Information on the **Point** class is given in Appendix A.

The diagram below shows an example of a **MyRect** object with vertex1 at (3, -2) and vertex2 at (8, 1).



The **MyRect** class is shown below.

```
import java.awt.*;

class MyRect {
  /************** Data members **********************/
  private Point vertex1, vertex2;

  /************** Constructor **********************/
  public MyRect(Point v1, Point v2) {
       setVertex1(new Point(v1));
       setVertex2(new Point(v2));
  }

  /**************** Accessors ***********************/
  public Point getVertex1() { return vertex1; }

  public Point getVertex2() { return vertex2; }

  /**************** Mutators ************************/
  public void setVertex1(Point pt) { vertex1 = pt; }

  public void setVertex2(Point pt) { vertex2 = pt; }

  /***************** Overriding methods *****************/
  // Overriding toString() method not shown here

  // Overriding equals() method not shown here
}
```

10. *(continued…)*

a. As we want every **MyRect** object created to have its south-west corner at vertex1 and north-east corner at vertex2, if the user passes in a different pair of opposite vertices of a rectangle, say (3, 1) and (8, -2) to the constructor, the object created will not have the right values for its two vertices.

   Hence, we need to modify the constructor such that regardless of which pair of opposite vertices is passed to the constructor, it places the south-west corner at vertex1 and the north-east corner at vertex 2. Modify the constructor. [5 marks]

b. We want to add a default constructor **MyRect()**. This constructor creates a rectangle with vertices at (0, 0) and (1, 1). The body of the constructor should contain a single statement. [1 mark]

c. We are considering changing the two mutators into private methods. What is the reason? [1 mark]

11. [13 marks]

A city has towns that occupy rectangular plots of land. Towns may overlap. Each town is represented by its south-west point (vertex 1) and north-east point (vertex 2). The x- and y-coordinates fall in the closed range [0, 1000].

A client program **CityPlanner.java** reads a list of towns into an array of **MyRect** objects. (**MyRect** is defined in the previous question.) You may assume that there are at least 2 towns. You may also assume that the array used is a normal array or an **ArrayList**. (Information on the **ArrayList** class is given in Appendix B.)

Let's call this array **towns**.

You are to complete **CityPlanner.java** with the following tasks:

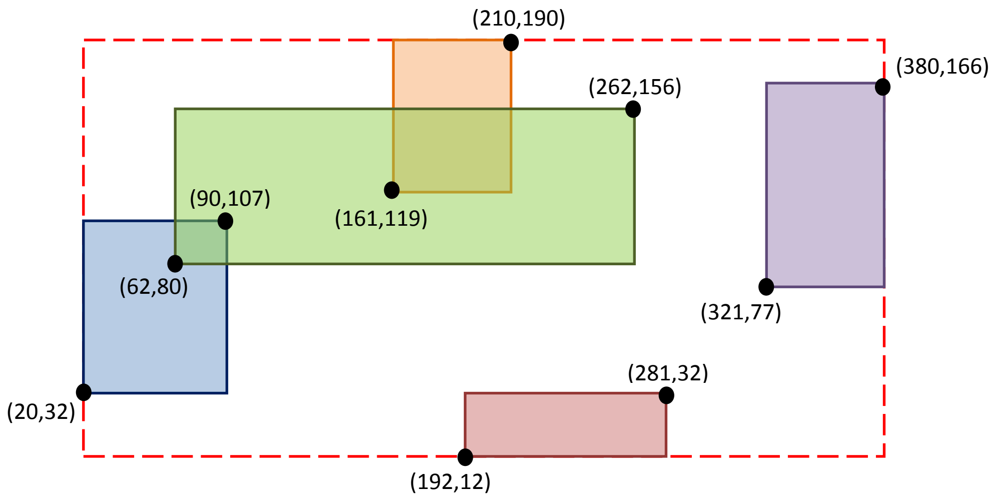
a. Find and return the *bounding rectangle* of the towns. A bounding rectangle is the smallest rectangle that covers all the towns. [6 marks]

```
public static MyRect boundingRect(…)
```

b. Every town has a centre, which is the centre of the rectangle. In other words, the centre is equidistant from the four corners of the rectangle. Determine the *minimum distance* between the centres of any two towns. (Your solution should use a certain **Math** method.) [7 marks]

```
public static double minDistBtwPair(…)
```

The following example shows 5 towns (not drawn to scale) and the answers for the above tasks. Note that this is only one example. The list of towns read from the user may not be the same as shown here.



- Bounding rectangle has vertices at (20, 12) and (380, 190).
- Minimum distance between the centres of any two towns = 43.4 (The top-most town and the biggest town.)

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 ==