NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
MIDTERM TEST FOR CS1020
AY2015/16 Semester 2

CS1020 – Data Structures and Algorithms I

5 March 2016 Time allowed: 1 hour 30 minutes

INSTRUCTIONS TO CANDIDATES

1. This test paper consists of **TEN (10)** questions and comprises **TWELVE (12)** printed pages.
2. This is a **CLOSED BOOK** test. No cheat sheet is allowed.
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**.

Section A

Questions 0 - 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. Question 0 is a bonus question and its mark is added only if the total mark for questions 1 to 10 is less than 30.

0. A/P Tan Sun Teck told the class a story in his first lecture. What is the story about?

   (A) Baby (B) Star Wars (C) James Gosling (D) Tennis

1. A class attribute has a default value depending on its type. The following list shows the types and their corresponding default values.

   i. **int**: 0
   ii. **int**: no default value
   iii. **char**: '\u0000' (null character with ASCII value 0)
   iv. **char**: null
   v. **String**: null
   vi. **String**: "" (empty string)

   Which of the above are correct?

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

2. What is the output of the following program?

```
import java.util.ArrayList;

public class Q2 {
   public static void main(String[] args) {
      int[] array = {1, 2, 3, 4, 5};
      ArrayList<Integer> list = new ArrayList<Integer>();

      for (int i=0; i<5; i++) {
         list.add(i%2, array[i] + array[(i+4)%5]);
      }

      System.out.println(list);
   }
}
```

   (A) **[9, 5, 7, 3, 6]**
   (B) **[9, 7, 5, 6, 3]**
   (C) **[9, 5, 3, 7, 6]**
   (D) **[9, 5, 7, 6, 3]**
   (E) None of the above

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

```
try {
   int x = 0;
   int y = 5/x;
}
catch (ArithmeticException ae) {
   System.out.println("Arithmetic Exception");
}
catch (Exception e) {
   System.out.println("Exception");
}
System.out.println("End");
```

(A) **End**

(B) **Arithmetic Exception**

(C) **Arithmetic Exception**
End

(D) **Exception**
End

(E) **Arithmetic Exception**
Exception
End

4. Assume that all necessary packages have been imported. What is the output of the following code fragment? The API of the **Point** class is given in Appendix A.

```
Point[] p = new Point[3];

for (int i=1; i<3; i++) {
   p[i].setLocation( p[i-1].getX()+1, p[i-1].getY()+2 );
}

System.out.println(Arrays.toString(p));
```

(A) **[java.awt.Point[x=0,y=0], java.awt.Point[x=0,y=0], java.awt.Point[x=0,y=0]]**

(B) **[java.awt.Point[x=0,y=0], java.awt.Point[x=1,y=2], java.awt.Point[x=2,y=4]]**

(C) Random output as the array elements have not been initialized.

(D) Compilation fails as first line contains syntax error.

(E) NullPointerException occurs at run time.

5. What is the output of the following program?

```
class E1 {
   protected int a;

   public E1(int x) { a = x; }

   public int getA() { return a; }

   public int f(int x) { return x+a; }
}

class E2 extends E1 {
   protected int a;

   public E2(int y) { super(y+100); }

   public int getA() { return a; }
}

public class Q5 {
   public static void main(String[] args) {
      E2 obj = new E2(12);
      System.out.println(obj.getA() + ", " + obj.f(34));
   }
}
```

(A) **0, 34**

(B) **0, 146**

(C) **12, 46**

(D) **12, 146**

(E) **112, 146**

Section B

6. [2 marks]
Given the following class definitions of D1 and D2:

```
class D1 {
   public int y;

   public D1(int i) {
      y = i;
   }
}
```

```
class D2 {
   public void f(D1 d1) {
      d1.y = 5;
      d1 = new D1(3);
      System.out.print(d1.y);
   }
}
```

What is the output of the following code fragment?

```
D1 d1 = new D1(2);
D2 d2 = new D2();
d2.f(d1);
System.out.println(" " + d1.y);
```

7. [3 marks]
Study the following program:

```
class A {
  public void m() {
    System.out.println(this.getClass().getName() + ".m in A");
  }

  public void n() {
    System.out.println(this.getClass().getName() + ".n in A");
  }
}

class B extends A {
  public void n() {
    System.out.println(this.getClass().getName() + ".n in B");
  }
}

class C extends B {
  public void n() {
    System.out.println(this.getClass().getName() + ".n in C");
  }

  public void p() {
    System.out.println(this.getClass().getName() + ".p in C");
  }
}

public class Q7 {

  public static void main(String[] args) {
    B b = new C();
    /* Line 31: call a method */
  }
}
```

this.getClass().getName() returns the name of the class of this object, that is, A, B or C.

What is the output of the program if line 31 is replaced by each of the following method calls? If there is an error, indicate so and state the reason (no mark will be given if the reason is wrong or not offered).

i. `b.m();`

ii. `b.n();`

iii. `b.p();`

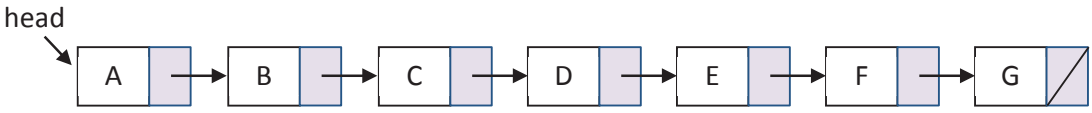
8. [5 marks]
The following algorithm takes a linked list with its first node pointed to by **head** and performs two things: (1) modifies the given linked list and (2) creates a new list with its first node pointed to by **head2**.

You may assume that the given linked list contains at least two nodes. **head**, **head2**, **curr**, **curr2** and **jump** are all ListNode variables.

The algorithm does not create any new nodes or delete any existing nodes; it merely modifies the pointers. The getNext() and setNext() methods are the standard methods discussed in class.

```
head2 = head.getNext();              // Line 1
curr = head;                         // Line 2
curr2 = head2;                       // Line 3
while (curr2 != null) {              // Line 4
    jump = curr2.getNext();          // Line 5
    curr.setNext(jump);              // Line 6
    curr2.setNext(jump.getNext());   // Line 7
    curr = curr.getNext();           // Line 8
    curr2 = curr2.getNext();         // Line 9
}                                    // Line 10
```

a. Trace the algorithm on the following linked list with 7 nodes and draw the resulting two linked lists. [2 marks]



b. Explain why the algorithm does not work in general (still assuming that the given linked list has at least two nodes), and add code into the algorithm to correct it.

You may indicate where the new code should go by referring to the line numbers, for example: "insert the following code after line 2."

You should not modify or remove any of the given lines in the algorithm, or no marks will be awarded. Keep your answer simple and short. [3 marks]

9. [6 marks]
Suppose the following two **ArrayList** objects **numbers** and **digits** have been created as follows.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
ArrayList<Integer> digits = new ArrayList<Integer>();
```

The ArrayList **numbers** contains a list of Integer objects with positive values, while the ArrayList **digits** contains the ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

Write a code fragment (not the whole program) to go through each element in **numbers**, and for each digit in that element, strike off the corresponding digit in **digits**.

For example, if **numbers** contains { 123, 999, 2046 }, then processing its first element 123 would strike off the digits 1, 2, and 3 from **digits**, processing 999 would strike off the digit 9 from **digits**, and processing 2046 would strike off 0, 4 and 6 from **digits**. Hence, in the end **digits** contains only 5, 7 and 8.

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

10. [9 marks]

A **MyPolygon** class is defined in the partial code below. The vertices of a polygon are stored in an **ArrayList** with each vertex being a **Point** object with integer x- and y-coordinates. The **toString()** method is not shown.

The APIs of **Point** and **ArrayList** classes are given in Appendices A and B respectively.

```
import java.awt.Point;
import java.util.ArrayList;
import java.text.DecimalFormat;

class MyPolygon {
   private double _perimeter;
   private ArrayList<Point> _vertices;

   public MyPolygon(ArrayList<Point> pts) {
      _vertices = new ArrayList<Point>(pts);
      _perimeter = computePerimeter();
   }

   private double computePerimeter() {
      double sum = 0.0;

      /* to be completed */
      return sum;
   }

   public String toString() {
      /* Omitted for brevity */
   }
}
```

A partial client program **TestPolygon** is given below.

```
import java.util.ArrayList;
import java.util.Scanner;
import java.awt.Point;

public class TestPolygon {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int size = sc.nextInt();
      ArrayList<Point> pts = new ArrayList<Point>(size);
      for (int i=0; i<size; i++) {
          Point pt = new Point(sc.nextInt(), sc.nextInt());
          pts.add(pt);
      }

      if (isValid(pts)) {
          MyPolygon polygon = new MyPolygon(pts);
          System.out.println(polygon);
      }

   /* To complete the isValid method */
}
```

10. (continued…)

a. You are to complete the **computePerimeter()** method in **MyPolygon** to compute the perimeter of the polygon.

Full credit is given only if you use the appropriate method(s) in the API to keep your code short. [4 marks]

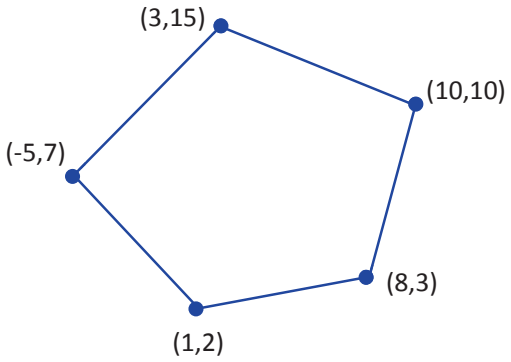
b. A list of vertices represents a valid polygon if (1) there are at least three vertices and (2) there are no duplicate vertices among all the vertices. For simplicity, you may assume that other properties of a valid polygon, such as no two edges should cross each other, are satisfied.

You are to write the **isValid(…)** method in **TestPolygon**, which returns true if the list of vertices represents a valid polygon, or false otherwise.

Do not use sorting in your solution. [5 marks]

Below is a sample run. Input data are shown in bold. The polygon is illustrated below (diagram not drawn to scale) with a perimeter of 42.08.

```
5
1 2
8 3
10 10
3 15
-5 7
[5: 42.08: (1,2)-(8,3)-(10,10)-(3,15)-(-5,7)]
```



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 ==