**Name:**

**Student ID:**

**ISTD 50.001 Quiz 1**

Date: 29-Sept-2017, Friday

**3.15 p.m. to 4.45 p.m.**

* Write your name and ID at the top of this page.
* Answer all 4 questions. You can secure a maximum of 100 points.
* For questions 1-3, submit your code to eDimension
* For question 4, write your answers on this exam paper.
* All answers will be manually graded. You may be able to earn partial credit for questions.
* **You are not allowed to use any Internet accessing or communicating device during the exam.**
* **You are not allowed to consult anyone inside or outside of the classroom other than the course instructors of 50.001.**
* This exam is open book. You may refer to the course slides / notes and your personal notes.
* You can access Java API reference: http://docs.oracle.com/javase/8/docs/api/
* You can use Android Studio / other IDE to test your programs.
* Good luck!

**Q.1 [Total: 25 points]**

**Part-1 [15 points]** Design a class named **Line** to represent a straight line in 2D. The class contains:

a) Four private **double** data fields named **x1, y1, x2, y2** that specify the two endpoints of the straight line: (x1, y1), (x2, y2)

b) A public no-arg constructor that creates a default Line with endpoints (0, 0) and (1,1).

c) A public constructor that creates a Line with the specified x1, y1, x2, y2: public Line(double x1, double y1, double x2, double y2)

d) A method computeSlope() that returns the slope of the straight line. You can assume that the slope is always finite.

e) A method computeLength() that returns the length of the straight line, i.e. Euclidean distance between (x1, y1) and (x2, y2).

f) A method to override **public String toString()** to allow printing of the Line object as follow:

Line:(x1,y1);(x2,y2)

where (x1, y1) and (x2, y2) are the endpoints. See example Test code.

g) Implement the Comparable<Line> interface to allow sorting of Line objects based on their length. See example Test code.

**Part-2 [5 points]**

Design a class named **LineSlopeComparator** that implements the Comparator<Line> interface to allow sorting of Line objects based on their slope. See example Test code.

**Part-3 [5 points]**

Design a static method:

public static Line minLineSlope(Line m1, Line m2, Line m3, LineSlopeComparator c)

The method **minLineSlope** takes in 3 Line objects: m1, m2, m3, and returns the Line object with the minimum slope. The method also takes in an object of LineSlopeComparator class that you have created in Part-2. Full credit will be given only if you use the LineSlopeComparator object for the comparison.

Note that you can obtain the start code and test code from eDimension.

Submit your code: Line.java, LineSlopeComparator.java, TestLine.java (with minLineSlope method) as a single zip file to eDimension.

**public class** TestLine {  
 **public static void** main(String[] args) {  
 Line m = **new** Line(0,0, 2,2);  
 System.***out***.println(m.computeLength());  
 System.***out***.println(m.computeSlope());  
 System.***out***.println(m);  
  
  
 List<Line> listLine = **new** ArrayList<Line>();  
 listLine.add( **new** Line(0,0, 1,2));  
 listLine.add( **new** Line(0,0, -1,0));  
 listLine.add( **new** Line(0,0, 3,4));  
  
 Collections.*sort*(listLine);  
 System.***out***.println(listLine);  
  
 Collections.*sort*(listLine, **new** LineSlopeComparator());  
 System.***out***.println(listLine);  
  
 Line y = *minLineSlope*(  
 **new** Line(0,0, 1,2),  
 **new** Line(0,0, -1,0),  
 **new** Line(0,0, 3,4),  
 **new** LineSlopeComparator());  
  
 System.***out***.println(y);  
  
 }  
  
 **public static** Line minLineSlope(Line m1, Line m2, Line m3, LineSlopeComparator c) {

// TODO: implement minLineSlope  
  
 }  
}  
  
*/\*  
output:  
  
2.8284271247461903  
1.0  
Line:(0.0,0.0);(2.0,2.0)  
[Line:(0.0,0.0);(-1.0,0.0), Line:(0.0,0.0);(1.0,2.0), Line:(0.0,0.0);(3.0,4.0)]  
[Line:(0.0,0.0);(-1.0,0.0), Line:(0.0,0.0);(3.0,4.0), Line:(0.0,0.0);(1.0,2.0)]  
Line:(0.0,0.0);(-1.0,0.0)  
 \*/*

**Q.2 [Total: 25 points]**

An array is a fixed-size data structure. Once an array is created, its capacity cannot be changed.

In this question, you are going to use arrays to implement dynamic data structures: **SmartArrayList**. SmartArrayList is similar to an array with elements stored in **consecutive memory locations**. However, the capacity of the SmartArrayList can change after it is created. The trick is to create a larger new array to replace the current array, if the current array cannot hold new elements in the list. We focus on a SmartArrayList that contains only String element. Initially, an array of String [] is created with a default capacity = INIT\_CAPACITY. Initial code can be found in eDimension.

Implement class SmartArrayList. The class contains:

a) A method **public void add(int index, String s)** that inserts a String element at a specified index, where 0<= index <= no\_of\_element in the SmartArrayList.

When inserting a new String element into the array, first make sure that there is enough room in the array. If not, create a new array twice as large as the current one. Copy the elements from the current array to the new array. The new array now becomes the current array. Before inserting a new element at a specified index, shift all the elements after the index to the right and increase the list size by 1. See example Test code.

If index < 0 or index > no\_of\_element in the SmartArrayList, the method add(int index, String s) should return with no effect.

b) A method **public void add(String s)** that appends a String element at the end of the SmartArrayList. You need to ensure there is enough room in the array. If not, perform similar action as in (a).

c) A method public **String get(int index)** that returns the String element at a specified index. The method returns null if the index is inappropriate (what would be an inappropriate index for this operation?)

d) A method public **void set(int index, String s)** that sets the element at a specified index to the input String element. The method returns with no effect if the index is inappropriate.

e) A method public **void remove(int index)** that deletes the element at a specified index. Your code needs to shift all the elements after the index to the left by one position and decrease the list size by 1. The method returns with no effect if the index is inappropriate.

f) A method to override **public String toString()** to allow printing of the SmartArrayList object as in the example Test code.

Note that SmartArrayList is similar to the standard JAVA ArrayList. But you should provide your own implementation and cannot inherit any code from ArrayList.

Submit your SmartArrayList.java code to eDimension.

**public class** SmartArrayList {  
 **private static final int *INIT\_CAPACITY*** = 2;  
 **private int size**=0; // no of stored element  
 **private** String[] **data** = **new** String[***INIT\_CAPACITY***];  
  
  
 **public** SmartArrayList() {  
  
 }

// TODO: IMPLEMENT THE CLASS

}

**public class** TestSmartArrayList {  
 **public static void** main(String[] args) {  
 SmartArrayList w = **new** SmartArrayList();  
 w.add(**"hello"**);  
 w.add(**"student"**);  
 w.add(1, **"dear"**);  
  
 System.***out***.println(w);  
  
 w.add(4, **"java"**);  
 w.add(3, **"python"**);  
  
 System.***out***.println(w);  
  
 System.***out***.println(w.get(2));  
 w.set(2,**"instructor"**);  
 w.remove(0);  
 System.***out***.println(w);  
  
  
 w.add(0,**"nus"**);  
 w.add(1,**"ntu"**);  
 w.add(0,**"sutd"**);  
  
 System.***out***.println(w);  
  
  
 }  
}  
  
  
*/\*  
  
[hello,dear,student]  
[hello,dear,student,python]  
student  
[dear,instructor,python]  
[sutd,nus,ntu,dear,instructor,python]  
  
  
 \*/*

**Q.3 [Total: 30 points]**

You are leading a software development team and your team is receiving many job assignments to finish.

Your team has a list of job assignments to finish, each requiring a certain amount of time to complete. Your team has exactly **3 members** to work on these assignments. Each member can only work on one assignment at a time. Once a member starts an assignment, s/he must complete that before taking up another assignment. Being effective and hard-working software developers, the team members do not require any break between the job assignments.

Can you write a static method to determine the smallest amount of time your team will need to finish all the job assignments, given the list of assignments to complete?

**public static int** computeTimeFinish(ArrayList<Integer> l)

The input is an ArrayList of n integers (1 <= n <= 40), where n is the number of job assignments. The n integers, t1, t2, …, tn, where 1<= ti <= 30, indicate the time in days required to finish each job assignment.

The output is the smallest amount of time that is needed to finish all the job assignments, given the list of assignments.

**public class** TimeFinishAssignment {

**public static void** main(String[] args) {  
 ArrayList<Integer> listOfAssignment = **new** ArrayList<>();  
 listOfAssignment.add(6);  
 listOfAssignment.add(7);  
 listOfAssignment.add(8);  
 listOfAssignment.add(9);  
 listOfAssignment.add(10);  
  
 System.***out***.println(*computeTimeFinish*(listOfAssignment));  
 *// output: 15* }  
  
 **public static int** computeTimeFinish(ArrayList<Integer> l) {

//TODO: implement computeTimeFinish  
 }  
  
}

Test case:

30

15 10 20

6 7 8 9 10

3 4 3 5 6 2 9 2

Outputs:

30

20

15

12

**Q.4 [Written][20 points]**

a) In the context of Object oriented programming, explain the term “Constructor chaining”. Why is it important? How “Constructor chaining” can be achieved in Java?

b) What are the differences between **instance variable** and **static variable**?

c) Can a static method call an instance method of the same class definition? Explain your answer.