C212 Lab 5

Intro to Software Systems

Objectives:

What you will learn:

- Array (One Dimensional and 2 Dimensional) and ArrayList
- Common Algorithms
- Writing class & Unit Testing

Lab Instructions:

Complete the following exercises. For each, review the requirements given below and complete the required work. Please compress all files into a zip file and submit it through Canvas. The grading scheme is also provided on Canvas.

1. Min-Max Length of Strings

Find the smallest and largest strings from a given list of strings, where small and large refer to the length of the strings.

Implementation Details:

- Prompt the user to enter a sequence of strings/words and input -1 to end.
- Store these strings in an **ArrayList**.
- Now, for each string in this **ArrayList,** calculate the length and store those values in an another **ArrayList**.
- Find the minimum and maximum length values from this **ArrayList**, you can use built-in Java functions if you would like but it is not required.
- Store those two values in one **Array** and add the strings with the minimum and maximum values to another **Array** in the following order for both {minValue, maxValue} and output these two **Arrays**.
- **Make sure that you take input in the main class so that you can test your code.**

Examples:

- Input**:**
  strs = {“Jack”, “Sam”, “Christopher”}
  sumStrs = {4, 3, 11}
  Output: [3, 11] [“Sam”, “Christopher”]

- Input:
  strs = {“Brown”, “Blue”, “Green”, “Pink”}
  sumStrs = {5, 4, 5, 4}
  Output: [4, 5] [“Blue”,”Brown”]

2. Find Larger Triangles

Find the triangles which have area larger than a specified value.

Implementation Details:

- Create a class **"MyTriangle"** which has the following attributes:
    - **Base:** *double*
    - **Height:** *double*
- Initialize Base and Height values using a constructor.
- Create another class "**LargeTriangles**", you should implement the method -
- *findTriangles* and your test client (or main) in this class only.
- Create 10 MyTriangle objects with base and height values of your choice in the main method and store them in an **ArrayList**.
- Take a threshold area value as a user input via Scanner.
- Find the triangles whose areas are larger than the threshold value and store them in another **ArrayList**.

public ArrayList< MyTriangle *> findTriangles (ArrayList***< MyTriangle>** *triangleList, double thresholdArea)*

- Return this **ArrayList** to the main method and print each larger triangle's base and height in the main method.
- **Note:** You should implement your test client in **LargeTriangle's** main method. Use at least 3 threshold values.

Examples:

trianglesList = {{height: 5, base:1 } , { height: 5 , base:2},{ height: 5 , base:3 }, {height: 5, base:4}, {height: 6 , base:1 },{ height: 6 , base:2 },{ height: 6 , base:3 },{ height: 6 , base:4 },{ height: 6 , base:5 },{ height: 6 , base:6 } }

trianglesList is an ***ArrayList*** *of* ***MyTriangle*** *objects, we have instantiated 10 triangle objects with base and height values of our choice.*

- Input:
  thresholdArea= 14
  Output:
  R1 - Height: 6 Base: 5
  R2 - Height: 6 Base: 6

3. Student Directory

A high school would like to store all their students' information in one place to be able to filter it by grade, age, and year in the future. Write a Java program to store this data and apply a filter.

Implementation Details:

- You must initialize a **2D Array** which holds information about all the students that are in the high school, where each individual array holds the information of a student in the following order:
  - [studentID, studentAge, studentYear, studentGrade]
  - All of the following variables should be integers, so studentYear should go from 9-12 and studentGrade should go from 0-100.
- As mentioned early you can **hardcode** the students' information into the "students" **Array** but you should display the following filtering information to the user and take user input for filtering the students.
  - Minimum Age
  - Minimum Year
  - Minimum Grade
- If a user inputs "**-1**" for any of the parameters, those parameters should not be used for filtering.
- After filtering based on the user input, you should **return an Array of student IDs** which satisfy the given parameters and return an **empty Array** if no students match.
- **Write at least two Junit tests.**

Examples:

- 2D Array:
  students = {{1, 18, 12, 90}, {2, 16, 10, 75}, {3, 15, 9, 85}}

  Input:
  User provides the following filter parameters via Scanner input:
  Minimum Age: -1
  Minimum Year: -1
  Minimum Grade: 80

  Output:
  1
  3

- 2D Array
  students = {{1, 18, 12, 90}, {2, 16, 10, 75}, {3, 15, 9, 85}}

  Input:
  User provides the following filter parameters via Scanner input:
  Minimum Age: 16
  Minimum Year: -1
  Minimum Grade: 83

  Output:
  1

4: MyArrayList Implementation

Simulate an **ArrayList** with limited functionality using **Arrays**.

Implementation Details:

- Create a class called **MyArrayList** that offers some of the functionalities provided by the ArrayList class in Java. Use the given class skeleton.
- **Write JUnit tests for each method.**

```
public class MyArrayList {
        private int size;
        private int[] data;

        public MyArrayList() { }

        public void add (int n) { }
        public void remove (int index) { }
        public String toString() { }
        public int getSize () { }
        public boolean isEmpty{ }
        public boolean contains (int n) { }
        public int indexOf (int n) { }
}
```

1. public MyArrayList () { }
   a. The first constructor with no arguments should initialize the size of **MyArrayList** and the data **Array** for storing values in **MyArrayList** following the idea that it should be an empty **MyArrayList** with size 0 and of default capacity 16.

2. public void add (int n) { }
   a. This method should add an int n to the end of the current **Array** data and increments the size of **MyArrayList**. Make sure that you adjust the **Array** in the case that it is no longer big enough to add more values into it.

3. public void remove (int index) { }
   a. Remove should remove the value at the index passed to the method and adjust the rest of **MyArrayList** so that it is still in order and the size is adjusted accordingly. You should also check that you are not removing an empty index.

4. public String toString () { }
   a. Returns a string value of the entire **MyArrayList**. **To receive full points, string must be in this form**: Each value should have a comma and a space except the last one. There should be no space before the closing bracket.
   Example: [1, 2, 3, 4, 5]

5. public int getSize () {}
    a. Returns the current size of **MyArrayList**.

6. public boolean isEmpty () { }
    a. Returns true if the the size of **MyArrayList** is 0, otherwise returns false.

7. public boolean contains (int n) { }
    a. Checks if **MyArrayList** contains the int n that is passed to the method and returns true if it does and false if it doesn’t.

8. public int indexOf (int n)
    a. Returns the index of the first occurrence of int n in **MyArrayList**. If this value is not in **MyArrayList**, this method should return an index of -1.