Mobile Computing

Quick Lab 2

1 Two dimensional arrays

Today, we're going to do an exercise working with 2 Dimensional Arrays.

Declaring a two dimensional array is very similar to creating a single dimensional array, and is done as follows:

```
int[][] myArray = new int[width][height];
```

So in order to create a two dimensional array with width 640 and height 480, we could say:

```
int[][] values = new int[640][480];
```

Then, remember that we index our array using these dimensions. It is useful to think of them as x and y, with x going from 0 to width and y going from 0 to height. You can query them as follows:

```
int value = values[x][y];
```

For instance, if we wanted to look at coordinate (600,300), we could do the following:

```
int myPixel = values[600][300];
```

If you want to iterate over this array, you'd need a nested for loop structure. For instance, if I wanted to set all the values in my pixel array to 150, I could do the following:

```
for (int x=0; x<640; x++){
  for (int y=0; y<480; y++){
    values[x][y]=150;
  }
}
```

2 Collections

In this lab we will look at the use of collections instead of arrays to store objects. Note that in order to use collections, you need to import the `java.util.*` library. The program itself is a movie ratings system, but we'll get to that later.

In Java, you can use collections by creating objects of that type. So, if you want a linked list for your program, you can use `LinkedList myList = new LinkedList()`. This will create a linked list for you. To add items to it, you can use the `add` method, and to retrieve items you use the `get` method. An example using ArrayList rather than LinkedList is shown below:

```
ArrayList al = new ArrayList();
String s = "Hello";
al.add(s);
String someString = (String)al.get(0);
```

Note that in the above example, the ArrayList is used to store items of type `Object`, meaning that the type of the object is lost when it is added to the collection. This is why we need to cast the retrieved object to `String`. It is much better to make the collection aware of the type of object that will be stored, as it stops us from making mistakes and simplifies the retrieval. If we want the ArrayList to be type-aware (meaning that it can remember the type that was put in it), we use angle brackets to denote the type as shown in the example below:

```
ArrayList<String> al = new ArrayList<String>();
Strings = "Hello";
al.add(s);
String someString = al.get(0);
```

As you can see in the above code, we do not need to cast the object if the collection is aware of the type. You should ALWAYS use type-aware collections.

2.1 Quick Lab 2a - Matrix transposition

1. In Eclipse create a java project called qlab2a
2. In your Program class's main method, you must read in a matrix from standard input. The matrix will be specified with spaces between the columns and with each row on a new line. The input is terminated with a -1. The input format is given in the next section.
3. You must transpose the matrix, outputting the result.
4. Submit your code to the marking system.

2.1.1 Example input

```
3 9 7 1
2 4 8 6
3 7 9 2
-1
```

2.1.2 Example output

```
3 2 3
9 4 7
7 8 9
1 6 2
```

3 Quick Lab 2b - Closest Pair

1. In Eclipse create a java project called qlab2b
2. For this question, consider writing down a plan for your program on paper before you start coding. This will help you clarify what you need to do.
3. In your Program class's main method, you must read in a set of coordinates, with a coordinate appearing on each line. The input is terminated by a -1.

4. Each coordinate has an x and a y value. You must store these coordinates so that you can use them later.

5. Next, you must write a method that calculates the distance between two coordinates, using the Euclidean distance ($\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$)

6. Your program must then output the distance between the closest pair of points (ie. the pair of points that has the smallest Euclidean distance)

3.1 Example Input

```
2,7
3,111
3,9
15,39
130,78
90,91
-1
```

3.2 Example Output

```
2.23606797749979
```

4 Quick Lab 2c - Collections

4.1 Setup

1. In Eclipse create a java project called qlab2c

4.2 Rating

1. In your qlab2c project, create a class called "Rating"

2. The class must contain the following attributes, with get and set methods for each one:

   - username - The name of the person submitting the rating
   - score - The rating awarded out of ten

3. It must also have a constructor which allows for the creation of a rating object using the username and score.eg.
   `Rating myRating = new Rating("movieguy87", 7);`

4. Submit your code for the Rating class to the marking system as a zip file.

4.3 Program

1. In your qlab2c project, create a class called Program

2. This class must contain an ArrayList called `allRatings` that will contain Rating objects.

3. In your Program class, create a method with the following method definition:

```
public double getAverage(ArrayList<Rating> v){
}
```

Note that this method accepts an ArrayList of Rating objects and outputs a double. This method should calculate the average of the scores of all the Rating objects in the ArrayList. Implement (write code for) this method. This method should work by looping through the ArrayList, getting each Rating object in the ArrayList. From each Rating object, it should get the score and add it to a total, from which it should calculate the average of all the scores.

4. The main method must read in a list of usernames and scores seperated by a `;` and must use these to create Rating objects which it will store in the `allRatings` object. The input should be terminated by a -1. An example input is shown below:

```
movieguy87;9
priya85;5
pravesh;3
joe;8
-1
```

Which should produce the following output:

```
6.25
```

5. Note that the `allRatings` object is not static, so you will have to make an instance of the Program class in order to use the `allRatings` ArrayList and the `getAverage` method.

6. The main method must use the getAverage method to calculate the average of all the Ratings, and must output this value.

7. Hint: Consider using the `split` function in the `String` class to process the lines. An example of its use is shown below.

```
String str = "red-blue-green";
String[] vals = str.split("-");
System.out.println(val[1]); //prints "blue"
```

The code above will split the String `str` every time it sees a - character. The resulting array will look like `["red", "blue", "green"]` (For instance, `vals[0]` will be `"red"`)

Note that you can use the Java API documentation to find the methods offered by the ArrayList and String classes.
http://java.sun.com/javase/7/docs/api/