COMP1406/1006 – Summer 2019

Submit a single file called `assignment1.zip` to the submission server
`http://bit.ly/COMP1406S19`

Your zip file must contain a directory called `comp1406a1` and all of your `.java` files must be in this directory. Do not include your `.class` files.

There are 100 marks possible marks.
Late submissions will be accepted up to midnight on Friday, July 19, but your grade will be computed out of 115 instead of 100. Your best grade (on time or late) will be recorded.

The objective of this assignment is to get you up to speed with basic Java syntax (control flow, branching, variables, methods, etc) and using arrays. This assignment will be graded for correctness only. The submission server will compile your code and run test cases to assess if your methods are correct or not. If one of your Java files does not compile, you will receive zero marks for that question. If your file does compile then your grade will be determined by the test cases used.

We are not testing your code for efficiency. However, it is expected that your code will terminate in a reasonable amount of time. If your code takes longer than (roughly) two or three seconds to run with our test cases on the server it will automatically terminate and you will receive a mark of zero for that question.

You may submit as many times as you wish before the submission deadline. We will use your *last* submission before the deadline for your grade.

You are NOT allowed to use any other Java classes for this assignment.
For example, you are not allowed to use `Arrays`, `ArrayList`, `System`, etc.
(You are always allowed to use `String` when needed.)
Using any other classes will result in a grade of zero for that problem.

1 Peak Analysis I [40 marks]

It is often important to identify local maxima in observed data. We will call these local maxima **peaks** in the data. Consider the following array:

3	6	7	5	3	-1	1

Notice that the value 7 is peak in the data since it is larger than both the value immediately before it (the value 6) and after it (the value 5) in the data. The value 1 will also be considered a peak since all values adjacent to it are smaller (in this case there is only adjacent value; the value -1 that comes before it). This data has TWO peaks.

The notion of a peak corresponds to a single location that is a local maximum. Sometimes there might be a local maximum with a value that is repeated in adjacent positions in the data. This kind of local maximum is called a **plateau**. A plateau is a local maximum that lies in two or more adjacent locations in the data. Consider the following array:

3	6	8	8	8	7	6	5	3	2

Here, there is a single plateau corresponding to array index positions 2, 3, 4 all with the value 8. This data has ONE plateau and ZERO peaks.

In the provided `Peaks.java` file, complete the `numberOfPeaks()` method. This method returns the number of peaks in a give array of input data. (This is worth 30 of the 40 marks for this problem.)

```
public static int numberOfPeaks(int[] data)
```

In the provided `Plateaus.java` file, complete the `numberOfPlateaus()` method. This method returns the number of plateaus in a given array of input data. (This is worth 10 of the 40 marks for this problem.)

```
public static int numberOfPlateaus(int[] data)
```

Be sure that you DO NOT alter the structure of the provided files. If you remove the `package` directive on the first line of the files, for example, your code will not compile on the server and you will receive zero for the problem. You may add a `main` method for your own testing code (we will not run the main method).

2 Peak Analysis II [30 marks]

The values and locations of the peaks (and plateaus) in data is usually much more important than simply knowing how many peaks there are. In this problem you will find the values and locations of all the peaks. Create a class called `FindPeaks` with the single static method

```
public static int[][] findPeaks(int[] data)
```

The method returns a 2-dimensional array of integers that correspond to the values and locations (in the data array) of all the peaks. The order of the peaks in the output array should correspond to the order of the peaks in the data. For example, the first peak in the input data will be the first peak in the output array. Consider the following code:

```
int[]   data = {3,6,7,5,3,-1,1};             // array from problem 1
int[][] peaks = FindPeaks.findPeaks(data);
```

The `peaks` variable, after calling `findPeaks(data)`, will be equivalent to the following array

```
int [][] answer = { {7, 1},       // peak values
                    {2, 6}        // peak locations
                  };
```

Here `peaks[0][0]` is the value of the first peak which is located at index position `peaks[1][0]` in the input array `data`. The second peak's value/location is contained in `peaks[0][1]`/`peaks[1][1]`. Here `peaks[0]` contains all the values and `peaks[1]` contains their corresponding index positions in the data.

Note that the size of the returned array depends on the actual input data. For example, the data `[3]` has a single peak, the data `[2,2,2]` has no peaks and the data `[3,2,3]` has two peaks.

Your class must begin with the line
`package comp1406a1;`

3 Grades [30 marks]

In the provided `Grades.java` files, complete the `finalLetterGrade()` method. This method takes five real numbers (`double`s) as input corresponding to a student's term work (assignment, tutorial, quiz, midterm and final exam grades, in this order) and returns the final letter grade (`String`) based on the grading scheme in the course outline, the university's standard grade conversion table shown below and another situation when a student missed the midterm but has a valid doctor's note (details are outlined in the `Grades.java` file).

In the following table, note that a square bracket indicates that the end value is included in the range (inclusive) and the parenthesis indicates that it is not included (exclusive). For example, the range $[80, 85)$ consists of all values x satisfying $80 \leq x < 85$.

Letter Grade	Grade Range	Grade Point
A+	[90, 100]	12
A	[85, 90)	11
A-	[80, 85)	10
B+	[77, 80)	9
B	[73, 77)	8
B-	[70, 73)	7
C+	[67, 70)	6
C	[63, 67)	5
C-	[60, 63)	4
D+	[57, 60)	3
D	[53, 57)	2
D-	[50, 53)	1
F	[0, 50)	0

Each grade passed as input will be a number between 0.0 and 100.0 (inclusive).

For example, calling `Grades.finalLetterGrade(79.1, 79.2, 79.3, 79.4, 79.5)` will output the string `"B+"`.

We are using floating point numbers! We need to be careful.

There is always a danger using floating point numbers because there are an infinite number number of real numbers and only a finite (fixed) number of `double`s or `float`s. This results in a representational error that is something that we need to be mindful of when using floating point numbers. For this problem, you will notice an `EPSILON` variable defined at the top of the `Grades` files. (DO NOT CHANGE THIS VALUE!) You will use this value as a tolerance for the resolution of numbers you use in calculations. If the difference between two numbers (in absolute value) is less than or equal to EPSILON then they will be considered the same value.

What does this mean for your code? If you calculate an overall final grade of 79.999999999979 then this will result in a `"B+"`. But, it is likely that if you computed the grade by hand you might actually get 80.0 (a different grade!). If we use `EPSILON=0.0001` in our code, then this computed grade is equivalent to 80 and a grade of `"A-"` would be returned. You will need to use `EPSILON` to determine the correct final letter grades when the computed grades are close to a boundary (like 50, 53, ..., 85, 90). When a computed grade is close a boundary, if it is less than the boundary value but within `EPSILON` of it, you will round the grade UP. If it is not within `EPSILON` of the boundary, then you use the grade as it is.

4 Extra Practice [0 marks]

Create a class called `Extra` that has the following static method

```
public static int[] widthAndHeight(int row, int column, String[][] grid)
```

The method returns an array with two integers. These integers are the width and height of a run of the same string that intersect at the specified row and column. A run is sequence of the same string (consecutive array elements) either in a row or column. For example, consider the following array of Strings

```
String[][] example = { { "cat", "dog", "eel", "cat" },
                       { "dog", "dog", "cat", "cat" }.
                       { "dog", "dog", "cat", "cat" }.
                       { "cat", "cat", "cat", "cow" }.
                       { "dog", "dog", "cat", "cat" } };
```

The string in row 3 and column 2 (rows and columns start with 0) is indicated below and highlighted in yellow. This will be the intersection point of the two runs.

```
String[][] example = { { "cat", "dog",  "eel" , "cat" },
                       { "dog", "dog",  "cat" , "cat" }.
                       { "dog", "dog",  "cat" , "cat" }.
                       { "cat", "cat",  "cat" , "cow" }.
                       { "dog", "dog",  "cat" , "cat" } };
```

The two runs are shown below. The width run (a run along a row) has length 3 and the height run (a run along a column) has length 4.

```
String[][] example = { {  "cat" ,  "dog" ,  "eel" ,  "cat"  },
                       {  "dog" ,  "dog" ,  "cat" ,  "cat"  }.
                       {  "dog" ,  "dog" ,  "cat" ,  "cat"  }.
                       {  "cat" ,  "cat" ,  "cat" ,  "cow"  }.
                       {  "dog" ,  "dog" ,  "cat" ,  "cat"  } };
```

The method would return the array `[3,4]` if called with `widthAndHeight(3,2,example)`.

Note that the output of the method is always an array of integers of length 2. Each of the two numbers is always a number that is greater than or equal to 1.

When testing for strings you should not use `==`.
Take care to use the string method `.equals()` to properly test for string equality.

Submission Recap

A complete assignment will consist of a single file (`assignment1.zip`) with the following four files included: `Peaks.java`, `Plateaus.java`, `FindPeaks.java` and `Grades.java`.

You may also include `Extra.java` for some basic testing but this will not have any marks.

Be sure to use the default compression program on your compute to create the zip file. Do NOT use 7zip, winrar, etc. The server may reject your submission if you do not use the default compression (archive) program.