CS1073 Assignment #3

Fall 2012

Due: Friday, October 12, 2012 by 12:00 NOON in the appropriate assignment bin on E level of Head Hall (near the entrance to Gillin Hall).

The purpose of this assignment is to:

- introduce the `boolean` data type and decision statements,

---

1. Programming Exercise

Write a program that reads in four integers and displays `alternating` if the numbers alternate in terms of their relationships with each other. For instance, if the second number is less than the first, then the third must be greater than the second, and the fourth must be less than the third. Of if the second number is greater than the first, then the third must be less than the second, and the fourth must be greater than the third. Display `not alternating` otherwise. Any two adjacent numbers that are equal means not alternating.

Here is sample output from four runs (note: user input is shown here in italics):

```
Enter four integers: 1 5 7 4
not alternating

enter four integers: 3 8 -1 2
alternating

enter four integers: 9 8 10 4
alternating

enter four integers: -3 5 5 6
not alternating
```

For this question you only need to write one class. You may place all of your code in the main method of that class.

After you have tested your application and you're sure that it works properly, print a copy of your code. Also print sample output from running the program using the four test cases shown above.

2. A LineSegment Class

Download the `CartesianPoint.java` file available with this assignment on Desire2Learn. You do not need to change this file. You will use this class to help create a `LineSegment` class and an associated test driver. **We highly recommend you use incremental programming to complete this program, where you code a little, try to compile and run what you have so far, and repeat until done.**

Write a complete `LineSegment` class. A `LineSegment` is defined by its two endpoints, each of which is represented by a `CartesianPoint` object.

Create two constructors – one that receives two `CartesianPoint` objects, and the other that receives the four x and y values required for the constructor to create two `CartesianPoint` objects.

Include an accessor method for each of the two endpoints.

Also include the following three methods:

`isParallelTo` – Returns true if this line segment is parallel to another line segment. The other line segment is given as a parameter to this method. Two line segments are parallel if they have the same slope. Be careful with vertical lines, where the slope is undefined because calculating such a slope would require division by zero. Your method must handle vertical lines. (Hint: One way to do this is to perform your calculations in such a way that no divisions are necessary. Another possible solution would be to use an if statement to detect when you have a vertical line, prior to attempting any slope calculations.) Can't remember how to calculate the slope of a line segment? Google it!

`length` – Returns the length of this line segment. You can use the `distance` method in the `CartesianPoint` class to help with this calculation.

`midpoint` – Returns a `CartesianPoint` representing the midpoint of this line segment.

If you need information on how to find the midpoint of a line segment, you can check here: www.mash.dept.shef.ac.uk/Resources/StraightLines1A.pdf

Add javadoc comments to the `LineSegment` class. Include a javadoc comment for the class, and for each of its instance variables and methods (including the constructors). Include `@author`, `@param`, and `@return` tags where appropriate. You may wish to run `javadoc` to check that your comments are correctly formatted. However, you do not need to submit the `html` files obtained by running the `javadoc` command.

In a separate `LineTest` class, create a test driver that exercises each of the methods in your `LineSegment` class, including the accessors for the two endpoints, as well as both constructors.

Use the following line segments as test data:

Line Segment 1: (0.0, 0.0) to (0.0, 5.0)
Line Segment 2: (2.0, 5.0) to (2.0, 7.0)
Line Segment 3: (1.0, 5.0) to (6.0, 5.0)
Line Segment 4: (2.0, 5.0) to (7.3, 10.1)
Line Segment 5: (0.0, -3.0) to (10.6, 7.2)
Line Segment 6: (0.0, -3.0) to (10.6, 3.0)
Line Segment 7: (2.0, 5.0) to (2.0, 5.0)

For the `isParallelTo` method, include test cases for each of the following:

- Line segments 1 and 2.
- Line segments 1 and 3.
- Line segments 4 and 5.
- Line segments 4 and 6.
- Line segment 4 compared with itself (yes, this is considered parallel).

For each test case for the `isParallelTo` method, use an if-else statement in your test driver to print whether or not the two line segments were found to be parallel. All output should include enough information so you can identify which line segments are involved in each test case.

For both the `midpoint` and `length` methods, include test cases for each of the following:

- Line segment 1
- Line segment 3
- Line segment 4
- Line segment 7

It is okay to use the CartesianPoint toString() method for displaying midpoint results.

For each test case, you **must** print sufficient information to allow you to confirm that the test case performed as expected.

Once your program is complete, print a copy of the two classes you wrote (i.e. the `LineSegment` class and the test driver; you do not need to print `CartesianPoint.java`). Also print your test driver output.

Once the assignment is complete: Assemble your printed pages **in the same order as the questions**. **Staple** together at the top left corner. Be sure to sign the statement of authorship on your cover page. Submit your assignment to the correct bin.

What to hand in (a checklist):

1. The completed assignment cover page (available in Desire2Learn).
2. The application that you wrote for question 1,
3. The sample output for question 1,
4. The two classes you wrote for question 2 (you do NOT need to hand in `CartesianPoint.java`),
5. The test driver output for question 2.

End of Assignment 3