AP COMPUTER SCIENCE A UNIT 2 PROJECT

20 points major assessment

Due XXX

In this project, the goal is to create an interactive program that allows a user to enter a pairs of coordinate points, e.g. (4, 13) and (-3, -8), and your program will calculate and output for the user:

- The distance between the pair of points.
- The slope and *y*-intercept of the line passing through those points.
- The complete linear equation that passes through the points, e.g. $y = 3x + 1.0$

The user will then be asked to enter an *x* value and your program will "solve" for the corresponding *y* value and tell the user the coordinate of the point on the line with that *x*.

Here is a video demonstrating the program

For this project, you should implement a complete `LinearEquation` class with instance variables and methods outlined on the next page. You should also write a runner class (`LinearEquationRunner`) in which you write the logic for getting the user's input via `Scanner`, etc. and calling the appropriate methods of your `LinearEquation` class.

Feel free to use or edit other classes or code that you have already developed in previous labs for this project. The string methods will be useful!

Submission

- Submit this document in Google Classroom with the self-check column on the last pages completed by you (should be all X's)

- Complete your program in IntelliJ, push to GitHub, and share a link to your project below:

Program Technical Specifications

Your `LinearEquationRunner` class should hold the main "logic" of your program in the `main` method, including input and printed output (see examples here), and should include the following:

- Use the `Scanner` class to welcome the user and ask the user to enter each coordinate point; both *x* and *y* values should be *integers*, i.e. your program should support user-provided points like (3, -7), but does *not* need to support decimal values like (3.5, -7.2).

- The user should be able to type in each **in coordinate form**, like this: `(5, -12)` with opening and closing parentheses, and a comma followed by a space in between the *x* and *y* coordinates.

- Your program should be able to take each coordinate (scanned in as a `String`) and "parse out" both the *x* and *y* values. You should use the `Integer.parseInt` static method (see slides for example) and various String methods to help with this.
- Once you have the *x* and *y* values for each of the two points parsed out of the coordinate strings provided by the user, use them to create a `LinearEquation` object.
- Your program should then call the `lineInfo()` method of the `LinearEquation` object to obtain and print a string that includes the following:
    - The two original points
    - The equation of the line through the two points in $y = mx + b$ format
    - The slope of the line
    - The *y*-intercept of the line
    - The distance between the two points
- Lastly, after printing the line information through the two points, ask the user to enter an *x* value and have your program print the corresponding coordinate point (*x*, *y*) on the line; for example, if the equation is $y = 3x + 1.0$ and user enters an *x* value of 3.5, the program prints out (3.5, 11.5). Note that although the user was required to provide the initial coordinates using all integer values, the user *should* be able to provide a `double` value for *x* in this step (and your program should print a coordinate with `doubles`).
- See Full Program Examples 1 and 2 for examples of the program's output.
- **Important Note!** If the user enters two coordinates with the *same* ***x*** value, e.g. (4, 5) and (4, -6), the points fall on a *vertical* line, the slope is undefined, and there is *no valid linear equation* in the form $y = mx + b$ to model this. You should check for this situation after the user enters their points, but *before* creating a `LinearEquation` object with the *x* and *y* values. If the user *does* do this, inform your user that they have entered coordinates that result in a vertical line and provide an equation in the form $x =$ __, e.g. $x = 4$ for points (4, 5) and (4, -6), then end the program (do not print out any other information). Your `LinearEquation` class does *not* need to handle `x1` and `x2` being equal. See Full Program Example 3 for an example of the program's output for vertical lines.

Your `LinearEquation` class should support the following behaviors:

- Creating a new `LinearEquation` from points (x_1, y_1) and (x_2, y_2).
- Calculating and returning the distance between the two points, rounded to the *nearest hundredth*.
- Calculating and returning the slope and *y*-intercept of the linear equation containing the two (*x*, *y*) points, each rounded to the *nearest hundredth*.
- Generating and returning a printable `String` that represents the linear equation of the line in slope-intercept ($y = mx + b$) form, where *m* is the slope and *b* is the *y*-intercept; e.g. `"y = 3x + 1.5"`
    - Print *m* as a *fraction* rather than a decimal, e.g. `"y = 1/2x + 1.5"`

- If the fraction reduces to a whole number, print the whole number rather than a fraction: e.g. print `"y = 2x + 1.5"` rather than `"y = 8/4x + 1.5"`
    - One exception: A slope of "1" should not print, e.g. `"y = x + 1.5"`, and a slope of "-1" should print as "-", e.g. `"y = -x + 1.5"`
    - Aside from whole numbers, you do ***not*** need to "reduce fractions" otherwise; e.g. `"y = 6/8x + 1.5"` is totally fine (you do *not* need to print this as `"y = 3/4x + 1.5"`)
- If the slope is *negative*, print the negative sign to the left, e.g. print `"y = -2/3x + 1.5"` rather than `"y = 2/-3x + 1.5"`
- Do not print double negatives in the fraction, e.g. print `"y = 2/3x + 1.5"` rather than `"y = -2/-3x + 1.5"`
- *b* values should be printed as decimals, e.g. `"y = 4x + 1.5"`
- If the *y*-intercept is 0, omit it; e.g. print `"y = 4x"` rather than `"y = 4x + 0"`
- If the *y*-intercept is *negative*, print it as subtraction, rather than plus a negative, e.g. print `"y = 4x - 1.5"` rather than `"y = 4x + -1.5"`
- Handle *horizontal* lines (when $m = 0$, i.e. if `y1` and `y2` are equal) by printing $y = b$; for example, the line through points (2, 5) and (7, 5) should print as `"y = 5"` rather than `"y = 0x + 5"`
- **Note:** As mentioned in the `LinearEquationRunner` specifications above, your `LinearEquation` class does ***not*** need to handle vertical lines, i.e. when `x1 == x2` (and slope is thus undefined); this is noted as a precondition of the constructor.

- Generating and returning a printable `String` containing *all* of the `LinearEquation` object's information, including the two points, the slope, the *y*-intercept, the equation of the line through the two points in $y = mx + b$ format, and the distance between the two points.
- Calculating the coordinate point corresponding to a given *x* value using `doubles`, and returning it as a `String` in the format (*x*, *y*), e.g. `"(2.0, 1.5)"` or `"(-4.5, 16.85)"`
    - Your `LinearEquation` class should be able to calculate a coordinate for values provided by the user for *x* that is a `double`
- Rounding *all printed decimal values* to the nearest hundredth (including distance, slope, *y*-intercept, and x and y values in returned coordinates); the `Math.round` method can help with this. You should write a `roundedToHundredth` method that performs this rounding for you and then use that method everywhere else in your class where rounding is needed.

Suggestions for organizing your work flow:

1. Write initial code in your `LinearEquationRunner` class to obtain the coordinates from the user and figure out how to parse the two `int` *x* and *y* values from each coordinates entered as `(x, y)` and stored as a `String`; once you have *two* x's and two y's as `int`s, move on to developing the `LinearEquation` class and writing its methods.

2. Once you begin working on the `LinearEquation` class, you will quickly discover that the most challenging part of this task is the `equation()` method, which returns the equation as a String in the format $y = mx + b$ with various constraints on how the printing looks; you might consider getting all other methods working (and tested!) before tackling the `equation()` method.

3. You might do the `lineInfo()` method last, since that requires all other methods to be completed.

Below is the `LinearEquation` class with all required methods that you should implement. Make sure your implementation matches all technical specifications. Various test cases are provided on the pages that follow.

```
public class LinearEquation
/* Instance Variables */
private int x1;
private int y1;
private int x2;
private int y2;


/* Creates a LinearEquation object */
/* PRECONDITION: x1 and x2 are NOT equal (client programs are responsible for ensuring
   this precondition is not violated)
public LinearEquation(int x1, int y1, int x2, int y2)


/* Calculates and returns distance between (x1, y1) and (x2, y2), rounded to
   the nearest hundredth */
public double distance()



/* Calculates and returns the y-intercept of the line between (x1, y1) and
   (x2, y2), rounded to the nearest hundredth */
public double yIntercept()



/* Calculates and returns the slope of the line between (x1, y1) and
   (x2, y2), rounded to the nearest hundredth */
public double slope()



/* Returns a String that represents the linear equation of the line through points
   (x1, y1) and (x2, y2) in slope-intercept (y = mx + b) form, e.g. "y = 3x + 1.5".

    When generating the m value (slope), here are examples of "printable" slopes:
       5, -5, 1/2, 6/8 (reducing not required), 8/5, -2/3, -18/7

    Here are non-examples of "printable" slopes:
       1/-2 (should be -1/2), -4/-3 (should be 4/3), 8/4 (should be reduced to 2),
       -12/3 (should be -4), 3/3 (should be 1), -6/6 (should be -1)

    HINT: Be sure to check if the line is horizontal and return an appropriate string,
    e.g. y = 6
    HINT: Absolute value might be helpful for ensuring proper placement of negative sign!


    When generating the b value, here are some examples of "printable" y-intercepts:
       + 1.0      - 2.35           + 12.5           - 8.0       + 17.19
```

```
    Here are non-examples of "printable" y-intercepts:
       - -1.0      + -2.35      - -12.5      + -8.0      - -17.19      + 0    - 0

    HINT: Absolute value might be helpful for printing negative y-intercepts as
          subtraction!
 */
public String equation()



/* Returns a String of the coordinate point on the line that has the given x value, with
   both x and y coordinates as decimals to the nearest hundredth, e.g (-5.0, 6.75) */
public String coordinateForX(double xValue)



/* "Helper" method for use elsewhere in your methods; returns the value toRound rounded
    to the nearest hundredth

    HINT:  the Math.round method can help with this!
 */
public double roundedToHundredth(double toRound)



/* Returns a string that includes all information about the linear equation, each on
   separate lines:
     - The original points: (x1, y1) and (x2, y2)
     - The equation of the line in y = mx + b format (using equation() method)
     - The slope of the line, as a decimal (using slope() method)
     - The y-intercept of the line (using yIntercept() method)
     - The distance between the two points (using distance() method)

  This method should call all other appropriate methods to get the info it needs:
  equation(), slope(), yIntercept(), distance().

  */
public String lineInfo()
```

Examples of full program flow with input and printed output:

Full Program Example 1 (shown in the demo video):

```
Welcome!
Enter coordinate 1: (-1, 5)
Enter coordinate 2: (3, 10)

The two points are: (-1, 5) and (3, 10)
The equation of the line between these points is: y = 5/4x + 6.25
The slope of this line is: 1.25
The y-intercept of the line is: 6.25
The distance between the two points is: 6.4

Enter a value for x: 4

The point on the line is (4.0, 11.25)
```

Full Program Example 2: User enters decimal value for x

```
Welcome!
Enter coordinate 1: (1, -10)
Enter coordinate 2: (-3, 2)

The two points are: (1, -10) and (-3, 2)
The equation of the line between these points is: y = -3x - 7.0
The slope of this line is: -3.0
The y-intercept of the line is: -7.0
The distance between the two points is: 12.65

Enter a value for x: 2.5

The point on the line is (2.5, -14.5)
```

Full Program Example 3: User enters two points that are on a vertical line

```
Welcome!
Enter coordinate 1: (5, 7)
Enter coordinate 2: (5, -3)

These points are on a vertical line: x = 5
```

Test values to use for testing the `equation()`, `slope()`, `yIntercept()`, `distance()`, `lineInfo()`, and `coordinateForX(double)` methods of your `LinearEquation` class:

Sample test code to help you test your LinearEquation methods:

```
int x1 = -1;
int y1 = 5;
int x2 = 3;
int y2 = 10;
LinearEquation equation = new LinearEquation(x1, y1, x2, y2);
System.out.println("Equation: " + equation.equation());
System.out.println("Slope: " + equation.slope());
System.out.println("y-intercept: " + equation.yIntercept());
System.out.println("Distance: " + equation.distance());
System.out.println();
System.out.println("----- Line info -----");
System.out.println(equation.lineInfo());
System.out.println();
double testX = 4;
System.out.println("Coordinate for x: " + equation.coordinateForX(testX));
```

OUTPUT for the test case above:

```
Equation: y = 5/4x + 6.25
Slope: 1.25
y-intercept: 6.25
Distance: 6.4

----- Line info -----
The two points are: (-1, 5) and (3, 10)
The equation of the line between these points is: y = 5/4x + 6.25
The slope of this line is: 1.25
The y-intercept of the line is: 6.25
The distance between the two points is: 6.4

Coordinate for x: (4.0, 11.25)
```

Additional test points and the expected returned String from `lineInfo()` for each:

Test Case 1: Positive fractional slope (this test case shown above)

```
The two points are: (-1, 5) and (3, 10)
The equation of the line between these points is: y = 5/4x + 6.25
The slope of this line is: 1.25
The y-intercept of the line is: 6.25
The distance between the two points is: 6.4
```

Test Case 2: Positive fractional slope

```
The two points are: (2, 10) and (-1, 3)
The equation of the line between these points is: y = 7/3x + 5.34
The slope of this line is: 2.33
The y-intercept of the line is: 5.34
The distance between the two points is: 7.62
```

Test Case 3: Positive fractional slope, negative y-intercept

```
The two points are: (-6, -2) and (1, -1)
The equation of the line between these points is: y = 1/7x - 1.16
The slope of this line is: 0.14
The y-intercept of the line is: -1.16
The distance between the two points is: 7.07
```

Test Case 4: Positive fractional slope

```
The two points are: (1, 4) and (7, 12)
The equation of the line between these points is: y = 8/6x + 2.67
The slope of this line is: 1.33
The y-intercept of the line is: 2.67
The distance between the two points is: 10.0
```

Test Case 5: Positive fractional slope, y-intercept = 0 (not printed)

```
The two points are: (0, 0) and (4, 5)
The equation of the line between these points is: y = 5/4x
The slope of this line is: 1.25
The y-intercept of the line is: 0.0
The distance between the two points is: 6.4
```

Test Case 6: Negative fractional slope

```
The two points are: (-1, 4) and (-7, 12)
The equation of the line between these points is: y = -8/6x + 2.67
The slope of this line is: -1.33
The y-intercept of the line is: 2.67
The distance between the two points is: 10.0
```

Test Case 7: Negative fractional slope

```
The two points are: (6, 3) and (-1, 4)
The equation of the line between these points is: y = -1/7x + 3.84
The slope of this line is: -0.14
The y-intercept of the line is: 3.84
The distance between the two points is: 7.07
```

Test Case 8: Whole number positive slope, negative y-intercept

```
The two points are: (4, 0) and (6, 10)
The equation of the line between these points is: y = 5x - 20.0
The slope of this line is: 5.0
The y-intercept of the line is: -20.0
The distance between the two points is: 10.2
```

Test Case 9: Whole number negative slope

```
The two points are: (6, 2) and (8, -12)
The equation of the line between these points is: y = -7x + 44.0
The slope of this line is: -7.0
The y-intercept of the line is: 44.0
The distance between the two points is: 14.14
```

Test Case 10: Whole number negative slope, negative y-intercept

```
The two points are: (1, -10) and (-3, 2)
The equation of the line between these points is: y = -3x - 7.0
The slope of this line is: -3.0
The y-intercept of the line is: -7.0
The distance between the two points is: 12.65
```

Test Case 11: Whole number positive slope, y-intercept = 0 (not printed)

```
The two points are: (7, 14) and (5, 10)
The equation of the line between these points is: y = 2x
The slope of this line is: 2.0
The y-intercept of the line is: 0.0
The distance between the two points is: 4.47
```

Test Case 12: Slope of +1 (the 1 not printed)

```
The two points are: (-1, 3) and (2, 6)
The equation of the line between these points is: y = x + 4.0
The slope of this line is: 1.0
The y-intercept of the line is: 4.0
The distance between the two points is: 4.24
```

Test Case 13: Slope of -1 (only a negative sign is printed)

```
The two points are: (-1, 2) and (-3, 4)
The equation of the line between these points is: y = -x + 1.0
The slope of this line is: -1.0
The y-intercept of the line is: 1.0
The distance between the two points is: 2.83
```

Test Case 14: Line with slope of 1 through origin (y = x)

```
The two points are: (-2, -2) and (4, 4)
The equation of the line between these points is: y = x
The slope of this line is: 1.0
The y-intercept of the line is: 0.0
The distance between the two points is: 8.49
```

Test Case 15: Horizontal line (slope = 0), positive y-intercept

```
The two points are: (7, 12) and (3, 12)
The equation of the line between these points is: y = 12
The slope of this line is: 0.0
The y-intercept of the line is: 12.0
The distance between the two points is: 4.0
```

Test Case 16: Horizontal line (slope = 0), negative y-intercept

```
The two points are: (16, -2) and (3, -2)
The equation of the line between these points is: y = -2
The slope of this line is: 0.0
The y-intercept of the line is: -2.0
The distance between the two points is: 13.0
```

Checklist & Rubric (20 points; half point for each row)

#	Program Technical Requirement	Self check X	Mr. Miller's check
Design			
1	All method names and signatures match those in the specifications; i.e. don't change method names, parameter types, or return values from the requirements		
2	Code has proper indentation, use of well-named variables, blank spaces as appropriate to break up sections of code, and comments to explain complex design.		
The `LinearEquationRunner` client class			
3	In the `main()` method, the program welcomes the user.		
4	Program uses the `Scanner` class to accept two different coordinates and store them as `String` variables.		
5	Both coordinates are accepted and saved in the format "`(x, y)`"		
6	Successfully use string methods to extract x and y from each point		
7	Successfully use `Integer.parseInt` to convert string values to int values		
8	Program checks to see if x1 == x2 before creating a `LinearEquation` object.		
9	If user enters two points with the same x value (i.e. a vertical line), the program informs the user a message that the line is a vertical and outputs the equation of the line in x = ___ format, e.g. "`x = 4`" and the program ends.		
10	Program generates a `LinearEquation` object from the x and y values parsed from the coordinates, i.e. x1, y1, x2, y2		
11	Program uses `System.out.println(obj.lineInfo())`, where `obj` is your `LinearEquation` object, to print the `LinearEquation` object's info.		
12	After line's information is printed, the program asks user to enter an x value		
13	The program accepts an x value from the user as a `double`		
14	The program uses the `LinearEquation` object's `coordinateForX` method to obtain the `String` coordinate.		
15	The program prints the resulting `String` coordinate for the user.		
17	**All** printed decimal values are rounded to the nearest *hundredth*.		
The `LinearEquation` class			
17	`public LinearEquation(int x1, int y1, int x2, int y2)` constructor creates a LinearEquation object.		
18	`public double distance()` correctly calculates and returns distance between		

	`(x1, y1)` and `(x2, y2)`		
19	`public double yIntercept()` correctly calculates and returns the y-intercept of the line between `(x1, y1)` and `(x2, y2)`		
20	`public double slope()` correctly calculates and returns the slope of the line between `(x1, y1)` and `(x2, y2)`		
21	`public String equation()` correctly returns a `String` that represents the linear equation of the line through points `(x1, y1)` and `(x2, y2)` in slope-intercept (y = mx + b) form, where m is the "printable" slope and b is the y-intercept; e.g. `"y = 3x + 1.5"`		
22	• printed slope is a whole number when the fraction reduces to a whole number (e.g. `"4"` instead of `"8/2"`)		
23	• printed slope is a fraction when it can't be reduced to a whole number (e.g. `"4/3"`)		
24	• printed negative slopes have the negative sign in the correct place (e.g. `"-4/3"`, ***not*** `"4/-3"`)		
25	• no "double negatives" appear in the printed slop (e.g. `"4/3"`, ***not*** `"-4/-3"`)		
26	• correctly prints slopes of +1 and -1 (e.g. `"y = x + 4.0"` and `"y = -x + 4.0"`, ***not*** `y = "1x + 4.0"` or `"y = -1x + 4.0"`		
27	• printed y-intercepts are rounded to two decimal places.		
28	• printed y-intercepts are not shown for a y-intercept of 0.		
29	• printed negative y-intercepts appear as subtraction rather than plus a negative (e.g. `"y = 2x - 4.0"` and `"y = 1/2x - 7.25"`, ***not*** `y = "2x + -4.0"` or `"y = 1/2x + -7.25"`		
30	`public String equation()` handles horizontal lines (when *m* = 0, i.e. when `y1` and `y2` are equal) by returning an equation in the form *y* = *b*; for example, the line through points (2, 5) and (7, 5) should be returned as `"y = 5"` rather than `"y = 0x + 5"`		
31	`public String lineInfo()` correctly generates and returns a printable `String` outputting all required information.		
32	• Includes the original points: (x1, y1) and (x2, y2)		
33	• The equation of the line in y = mx + b format (generated by calling the `equation()` method)		
34	• The slope of the line, as a decimal (using `slope()` method)		
35	• The y-intercept of the line (using `yIntercept()` method)		
36	• The distance between the two points (using `distance()` method)		
37	`public String coordinateForX(double yValue)` correctly computes		

	the corresponding y value for the point on the line and returns a string representing the coordinate for the point `(x, y)` with values rounded to the nearest hundredth, e.g. `(4.5, -1.27)`		
38	`public double roundedToHundredth(double toRound)` correctly returns the `toRound` value rounded to the nearest *hundredth.*		
39	`roundedToHundredth(double toRound)` are used appropriately in your other methods so as to reduce redundancy		
Testing and submission			
40	All 16 test cases are tested (including others you think up) to ensure accuracy in your method development.		
	Rubric Row Total (out of 40)		
	Total Points (out of 20; 0.5 points per row)		