Project 2: Fun with Curve Math

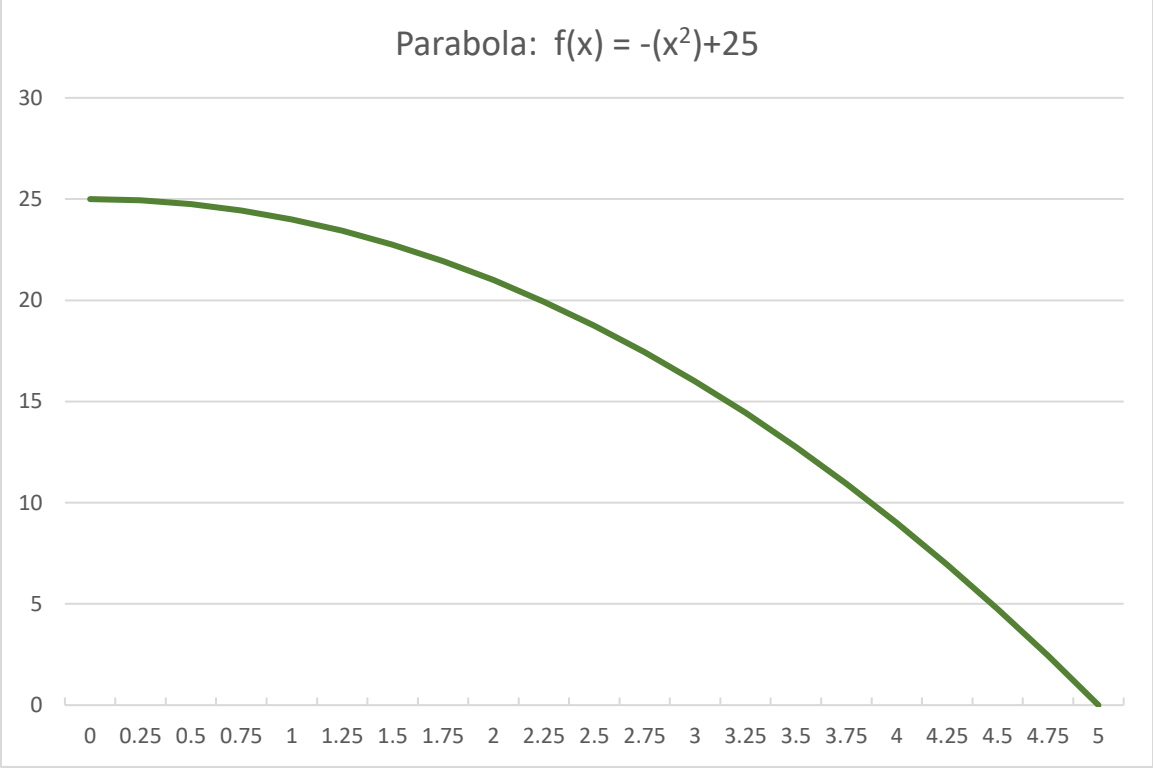
Carefully read this *entire document* before beginning your work.

1 Objective

This project will require indefinite loops to do its work. You'll also get a chance to display information to the user and gather information from her. The project's subject matter ties in nicely with the math that you have studied. It will also test your ability to design your approach, develop incrementally, and test your code.

2 Background

We're working with a parabola specified by this equation: $f(x) = -(x^2) + 25$ with *x*'s domain in the range 0 to 5 (inclusive). The partial parabola looks like this:



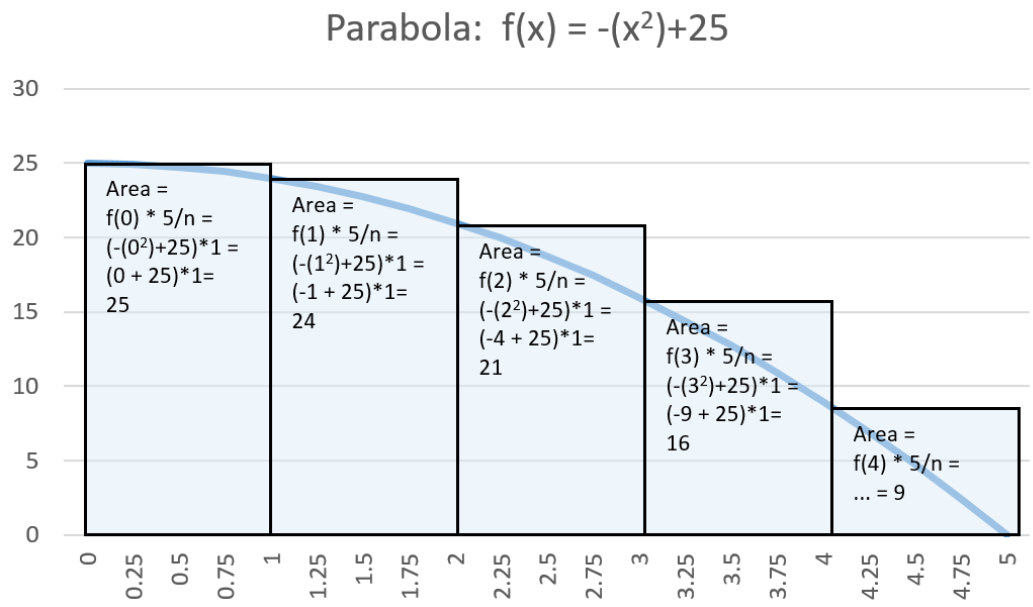
For example, when the *x* coordinate is 4, the y coordinate is $-(4^2) + 25 = -16 + 25 = 9$

The slope of the line at a given point is given by: $f'(x) = -2x$ (determined using limits or calculus), so for example when the *x* coordinate is 2.5, the slope of the line is $-2 \cdot 2.5 = -5$.

3 Area Estimation

The area under the curve can be estimated by using a series of *n* rectangles, each of which starts at a calculated value of *x*, with a width specified by 5/*n* (x's maximum domain divided by how many rectangles we're using) and the rectangle's left-side *y* determined by the parabola's defining equation[1].

Were we to visualize the estimation for five rectangles, it would look like this:



The total area is the sum of the areas of all the rectangles, in this case 25 + 24 + 21 + 16 + 9 = 95. The more rectangles one uses, the closer the resulting estimation to the actual area.

The actual area as determined by limits or calculus is: $\left(125 - \frac{125}{3}\right) = 83.\overline{33}$. So, after we calculate the area using rectangles, we can tell how far off the estimate is, in this case it's 14% too big; the overestimation is apparent from the graph, in fact.

4 User Interface

Display this menu:

```
Parabola Calculations Menu

1. Calculate area estimates
2. Find the slope of the tangent line at a given x
3. Find y for a given x
4. Exit the program

Enter your choice:
```

[1] This is called a Left Riemann Sum.

Keep the user within the menu system until they choose to exit, i.e., they can do more than activity while they are there. If the user enters an integer outside the range of valid menu choices, or enters an invalid data type, tell them they've made an error. The menu should then be displayed again.

Here are more details about the menu choices:

1. Asks the user for a starting and ending number of rectangles, plus an increment (and ensures the data entered is reasonable, e.g., end must be greater than start and maximum number of rectangles is no greater than 5000). For example, the user might supply 50, 80, and 10 to those questions, respectively, indicating s/he wants to start at 50 rectangles, end at 80 rectangles, and increment each time by 10 rectangles. Uses the rectangle estimation method to estimate the area for each number of rectangles specified by the user (e.g., 50, 60, 70, and 80). For *each* calculation shows the calculated area and the calculated delta from the actual area.
2. Asks the user for an x coordinate (0<=x<=5), then reports the slope of the tangent line.
3. Asks the user for an x coordinate (0<=x<=5), then reports the corresponding y value.
4. Exits gracefully (not artificially) from the program (i.e., a loop should end, you shouldn't forcible terminate the program or break out of the loop).

4.1 Getting User Input

Use a Scanner object to get keyboard input from the user.

5 Code Implementation

Create two separate classes, one that deals with the user, and one that does the math computations. Examples of this type of code division can be found in MyMath.java and TestMyMath.java. Follow the Course Style Guide.

5.1 ParaCalc

- This class should have *no user communication*; it is purely a *supplier* of calculation functions.
- There should be no main() method here.
- Write static methods for calculations that accept parameters and return results. Provide three public methods, one for each of the specified calculations. Other helper functions may be created.
- Functions must implement precondition tests on parameter values and throw exceptions when appropriate.

5.2 ParaApp

- This class should contain *all* the user communication, consisting of console display and user input via the Scanner class. This class is a *client* of calculation functions from the ParabolaCalc class.
- This class should have a main() method.
- You may define class constants, but no other class-level variables.
- Implement generic (not problem-specific) functions to get in-range integers and floating-point values from the user. Pass in the expected minimum and maximum; the functions should ensure the user makes an entry in the specified range *and* using the right data type (using Scanner look-ahead) and then passes back the user's choice. Important: you'll make of use of such functions in later projects!
- Create *only* one Scanner object instance. If you don't understand why, think and ask! It's a common misconception that more are needed.

5.3 General

- Use procedural decomposition. Further decomposition is allowed if you think it helpful.

6 Testing

- Test each function individually (unit testing), then the program holistically (integration testing). You don't need this to be in code for *this* project; but be aware that you must code it in future projects.
- Don't test only "happy path" inputs; you want to know what happens when bad inputs enter the system, then modify your code to handle them if you know enough to do so.

7 Submitting Your Work

You'll be creating two java files; zip them and submit them as a single file (e.g., in a .zip file). BlueJ can also create .jar files; be sure and specify "include source" if you use this method.

8 Extra Credit

For 5% extra credit, draw the resulting rectangles on a DrawingPanel (when the user chooses area estimation from the menu). This should work with any valid number of rectangles the user enters.

9 Grading Matrix and Achievement Levels

Achievement	**Max**
Compile/runtime errors	50%
Basic with non-looping menu	60%
Basic with looping menu	65%
Input range checking	70%
Functions to get in-range integers and floating-point values	80%
Data type checking added to "get" functions	85%
Function preconditions added	90%
Documentation and style	100%
Rectangles drawn	105%