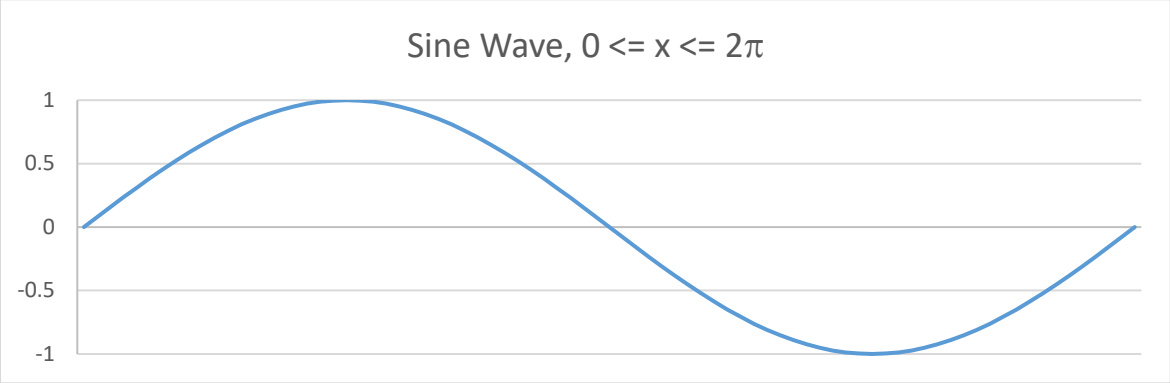
Project 3: Area Under Curve: Sine Wave

1 Objective

In addition to what you have already used in projects, this project requires indefinite loops to do its work. You will also get a chance to display information to the user and gather information from them. You will create multiple classes and make method calls across them. Finally, you will do some interesting math to estimate the area under the sine wave in a specific x domain.

2 Background

We will work with a sine wave, where x is measured in radians and is shown below in the domain $0 <= x <= 2\pi$. We'll focus on the first half, where y is positive (domain $0 <= x <= \pi$).



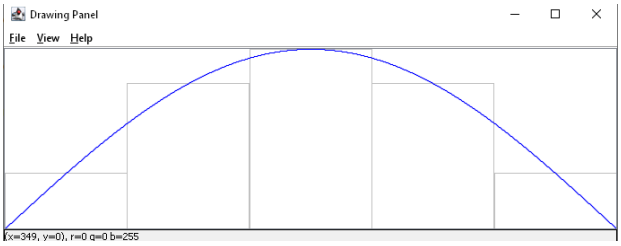
3 Area Estimation

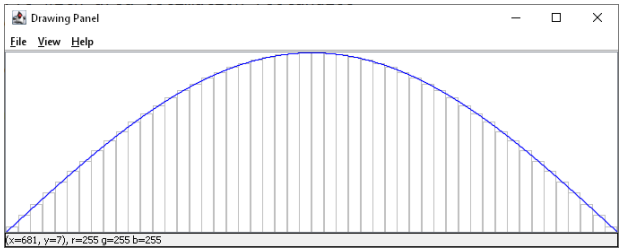
We will estimate area under the curve using a Riemann Sum (specifically a *Midpoint Riemann Sum*) which divides the area into a series of rectangles and calculates/sums their areas to estimate the area under a curve. As you might expect, the more rectangles we use, the better the estimate approximates the actual area. The actual area is 2 (Calculus can be used to calculate this), so we'll see how close we can get with our estimates. Here's are sample calculations using 5 rectangles:

Num. of rectangles:		5		
Width of each rect:		0.62831853	(π / 5)	
Actual area:		2		
Rect #	**Start**	**MidPoint**	**sin(MP)**	**Area**
1	0	0.31415927	0.309017	0.194161
2	0.628319	0.9424778	0.809017	0.50832
3	1.256637	1.57079633	1	0.628319
4	1.884956	2.19911486	0.809017	0.50832
5	2.513274	2.82743339	0.309017	0.194161
Total				**2.033281**

4 Graphical Output

Draw rectangles corresponding to the number of rectangles the user has requested (5 on the left example; 20 on the right). Then draw the sine wave (x domain 0 <= x <= π) so it is clearly visible.





5 User Interface

Display this menu:

```
-------------------------------------------
          Sine Wave Menu
-------------------------------------------

1.  Find y value for specified radians x
2.  Estimate area for 0 <= x <= PI
3.  Draw curve with area estimation rectangles

0.  EXIT the program

Enter your choice:
```

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 have made an error and make them try again.

Here are more details about the menu choices:

0. Exits gracefully, not artificially, from the program (i.e., a loop should end, you should not forcibly terminate the program or break out of the loop).
1. Allows user to enter a double x value (no range limitation), then calculates and displays a nicely labeled result showing 4 digits after the decimal point.
2. Asks the user for the number of rectangles to use, an integer in the range 1 to 500. Estimates the area under the curve with that number of rectangles and displays the result (again, with 4 digits after the decimal point).
3. Asks the user for the number of rectangles to use, an integer in the range 1 to 500. Draws the sine curve on a drawing panel that is 628x200 pixels (understand why?), also drawing number of filled rectangles requested by the user. Draw the curve *after* the rectangles so it is clearly visible.

5.1 Gathering User Input

Use a Scanner object to get keyboard input from the user. Use Scanner lookahead (not exceptions) to find data type issues and make the user fix those before continuing. We cover this type in class/videos.

6 Code Implementation

Create three separate classes, one that deals with the user, one that does the math computations, and one that draws. Our temperature conversion in-class code model multi-class interaction; follow that example. Follow the Course Style Guide. Use the specific class names shown below.

6.1 Class: SineCalc (does calculations)

- This class should have *no user communication nor main()*; it is a *supplier* of calculation functions.
- Write static methods for calculations, accepting parameters and returning results. Write methods for each of the specified calculations. Create other helper functions if you need them.
- Methods should implement precondition tests where appropriate, e.g., specifying a negative number of rectangles does not make sense and is not actionable.

6.2 Class: SineDraw (does drawing)

- This class should have *no user communication nor main()*; it is a *supplier* of drawing functionality.
- Write a static method for drawing, accepting parameters but returning no results. Create other helper functions if you need them.
- Write helper methods to turn cartesian coordinates to graphics coordinates, perhaps *cartesianXToGraphicX* (x parameter in radians) and *cartesianYToGraphicY*. This makes it easier to get drawings right. These should accept double parameter but return rounded integers.
- Create preconditions where they make sense.
- Draw the sine wave as a series of tiny lines; this will give the illusion of a curve.

6.3 Class: SineUI (handles user communication)

- 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 Calculations class.
- Never throw exceptions in classes that converse directly with the user; we want *validation*, here.
- This class should have a runnable main() method.
- Use no global variables. You may define class *constants*, of course.
- Implement generic (not problem-specific) methods to get in-range integers from the user. Pass in the expected minimum and maximum; the methods should ensure the user makes an entry in the specified range *and* using the right data type (using Scanner lookahead) and then passes back the user's choice. Hint: later projects may depend on this method, as well.
- Create *only* one Scanner object instance. If you do not understand why, please ask.

6.4 General

- Use procedural decomposition. Further decomposition is allowed if you think it helpful.
- Geometry requires precision; do not overuse integers here. In general, maintain precision. Then round at the last moment, just before the integer is required. This is good practice for graphics, too; the nearest pixel is usually the best one to choose.

7 Code Implementation

Follow our Course Style Guide found in Canvas.

8 Testing

- Test each function individually (unit testing), then the program holistically (integration testing). You do not need coded unit tests for this project; but be aware that you will do so in future projects.
- Do not 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. Here, user inputs should be bulletproof; nothing the user enters should blow up the program.

9 Extra Credit

Write tests for the methods in the SineCalc class. That includes all methods, preconditions, etc. The idea is that every statement in the class should have been hit by your tests.

10 Submitting Your Work

Use your OS to create a .ZIP file. Submit the compressed file.

11 Grading Matrix

Area	**Percent**
UI class	20%
UI class range & data type checking	15%
Calculation class	20%
Drawing class	15%
Drawn output: rectangles	10%
Drawn output: curve	5%
Preconditions	5%
Documentation and style	10%
Extra credit	3%
Total	**103%**