PROJECT SPECIFICATION

Problem Selection

Create a visual graphing calculator

Problem Description

- Create a visual graphing calculator application
- Create a canvas 400 wide by 400 pixels high (use a jPanel).
- Scale the canvas to show the effective interval from -2 to +2
- Provide radio buttons to select a function X, X2, X3, Sqrt(X), and Log(X).
- Provide input text boxes for scalar value (m) and an offset value (b).

Member Designation

We approve the project as described above, and authorize the team to proceed.

Member Name	Member Role	Date / Initial
Darah Backal	Programmer	12/03/2020
Ozzie Rodriguez	Tester	12/03/2020
Mouaz Ali	Designer	12/03/2020

High-Level Requirements

Members must work in their specific roles in a timely fashion.

- The algorithm designer solves the problem and creates the design documentation used by the programmer and tester
- The programmer writes commented Java source code after interpreting the documentation provided by the designer
- The tester creates and implements test cases and works with the designer and programmer to alleviate errors
- The tester is also responsible for producing the final report, including the original design documents, source code, and test case results

Deliverables

Use this report as a guide for what to include in your project submission. The following items must be included.

- Design documentation (digital white-board, algorithm pseudo-code, program flowchart)
- Java Source Code (fully commented using JavaDoc, implements the design specification)
- Test Cases (establish test cases and document the results, any code or design issues should be documented and resolved)

Algorithm Designer Documentation Checklist

•	The whiteboard should feature a conceptual overview of how the problem is solved. PowerPoint may be used as the whiteboard medium, however there are a host of possible ways to submit design documentation (see the professor for details).	6 Points
•	Flowchart featuring program flow and decision trees. Use PowerPoint to generate these diagrams unless you're familiar with a UML modeler that you prefer to use.	6 Points
•	Create plain language instructions (pseudo—code) for use by the programmer and quality assurance analyst during the implementation and testing phases.	6 Points
•	Identify input / output requirements. These should be shown on the flowchart.	2 Points

Programmer Task Checklist

•	Uses the documentation provided by the designer to implement Java Source Code to solve the problem.	10 Points
•	Uses resources such as pre-existing classes in the Java API to aide in development.	2 Points
•	Uses JavaDoc format to comment all classes, methods, and variables.	6 Points
•	Exports the project with eclipse for use by the quality assurance tester.	2 Points

Quality Assurance Tester Task Checklist

•	Uses resources such as the designer's flowchart and the plain language document to understand how the algorithm works.	4 Points
•	Analyzes the classes provided by the programmer and compares them to the designer's documentation. If any discrepancies are discovered, they should be documented and resolved with the programmer and designer.	8 Points
•	Create test cases based on the application implementation. Provide the results of the test cases as part of the final report.	5 Points
•	Create the final report including the original documentation, the source code, and the quality assurance results observed.	3 Points

CREATING A GRAPHIC CALCULATOR CLASS

To create a graphing calculator, understand how a graphene calculator works first

User input > Read Input > Instructions > Decisions > Perform Calculations > Display Results (output).

Parameters:

- JFrame Frame;
-

JTextField txtScalar, txtOffset;
JRadioButton rdbtnX, rdbtnX2, rdbtnX3, rdbtnLog, rdbtnSqrt;
JButton btnCheck;
JPanel canvas;
JLabel lblScalar, lblOffset, lblNewLabel

GRAPH METHOD ACCESSORS:

frame.setBounds(number 1, number 2, number 3, number 4)
canvas.setBounds(number 1, number 2, number 3, number 4)
lblScalar.setBounds(number 1, number 2, number 3, number 4)
txtScalar.setBounds(number 1, number 2, number 3, number 4)
lblOffset.setBounds(number 1, number 2, number 3, number 4)
textOffset.setBounds(number 1, number 2, number 3, number 4)
rdbtnX.setBounds(number 1, number 2, number 3, number 4)
rdbtnX2.setBounds(number 1, number 2, number 3, number 4)
rdbtnLog.setBounds(number 1, number 2, number 3, number 4)
rdbtnX3.setBounds(number 1, number 2, number 3, number 4)
rdbtnSqrt.setBounds(number 1, number 2, number 3, number 4)
btnCheck.setBounds(number 1, number 2, number 3, number 4)
lblNewLabel.setBounds(number 1, number 2, number 3, number 4)
g.drawLine(number 1, number 2, number 3, number 4)

Application()

txtScalar = new JTextField();
textOffset = new JTextField();
lblScalar = new JLabel("Scalar (m)");
lblOffset = new JLabel("Offset (b)");
rdbtnX = new JRadioButton(“X”);
rdbtnX2 = new JRadioButton("X^2");
rdbtnLog = new JRadioButton("Log(X)");
rdbtnX3 = new JRadioButton("X^3");
rdbtnSqrt = new JRadioButton("Sqrt(X)");
btnCheck = new JButton("Graph");
lblNewLabel = new JLabel("y = m * f (x) + b");

Frame.setLayout(null);
Frame.setVisible(true);
Frame.setSize(x,y);
Frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
Frame.setResizable(false);

rdbtnX.addActionListener();
rdbtnX2.addActionListener();
rdbtnX3.addActionListener();
rdbtnLog.addActionListener();
rdbtnSqrt.addActionListener();

Frame.add(canvas); → adds frame to canvas
Frame.add(lblScalar); → adds frame to lblScalar
Frame.add(txtScalar); → adds frame to txtScalar
Frame.add(lblOffset); → adds frame to lblOffset
Frame.add(textOffset); → adds frame to textOffset
Frame.add(rdbtnX); → adds frame to X button
Frame.add(rdbtnX2); → adds frame to X^2 button
Frame.add(rdbtnLog); → adds frame to Log(X) button
Frame.add(rdbtnX3): → adds frame to X^3 button
Frame.add(rdbtnSqrt); → adds frame to Sqrt(X) button
Frame.add(lblNewLabel); → adds frame to y=m*f(x) label

paintComponent method → to draw on JPanel
isNumeric Method → Returns a boolean code.

drawGrid(Graphics g) {
drawX(Graphics g) {

```
drawX2(Graphics g) {
drawX3(Graphics g) {
drawLogX(Graphics g) {
drawSqrtX(Graphics g) {

if (operation==1) {
	drawX(g);
double dx = 0.001;
for(double x=-2; x<=2; x=x+dx) {
double y = scalar*x+offset;
int cX = (int) Math.round((x*100)+200);
int cY = (int) Math.round((-y*100)+200);

} else if (operation==2) {
	drawX2(g);
double dx = 0.001;
for(double x=-2; x<=2; x=x+dx) {
double y = scalar*(x*x)+offset;
int cX = (int) Math.round((x*100)+200);
int cY = (int) Math.round((-y*100)+200);

} else if (operation==3) {
	drawX3(g);
double dx = 0.001;
for(double x=-2; x<=2; x=x+dx) {
double y = scalar*(x*x*x)+offset;
int cX = (int) Math.round((x*100)+200);
int cY = (int) Math.round((-y*100)+200);

} else if (operation==4) {
	drawLogX(g);
double dx = 0.001;
for(double x=-2; x<=2; x=x+dx) {
double y = scalar*(Math.log(x))+offset;
int cX = (int) Math.round((x*100)+200);
int cY = (int) Math.round((-y*100)+200);

} else if (operation==5) {
	drawSqrtX(g);
double dx = 0.001;
for(double x=-2; x<=2; x=x+dx) {
double y = scalar*(Math.sqrt(x))+offset;
int cX = (int) Math.round((x*100)+200);
int cY = (int) Math.round((-y*100)+200);
}
if (isNumeric(txtScalar.getText())) {
scalar = Double.parseDouble(txtScalar.getText());
} else {
```

```
        scalar = 1.0;
}

if(isNumeric(textOffset.getText())) {
. . .
} else {
. . .

if (rdbtnX.isSelected()) {
Set all others to false
Operation = 1;

if (rdbtnX2.isSelected()) {
. . .
Operation = 2;

if (rdbtnX3.isSelected()) {
. . .
Operation = 3;

if (rdbtnLog.isSelected()) {
. . .
Operation = 4;

if (rdbtnSqrt.isSelected()) {
. . .
Operation = 5;


canvas.repaint()
```

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.PaintEvent;
import java.awt.geom.Line2D;
import java.util.Scanner;
import java.awt.event.ActionEvent;
import javax.swing.JRadioButton;
import javax.swing.JPanel;
import java.awt.GridLayout;
import javax.swing.JSplitPane;

public class Application extends JPanel
```

implements ActionListener{

```
private JFrame frame;
private JTextField txtScalar;
private JTextField textOffset;
private JRadioButton rdbtnX;
private JRadioButton rdbtnX2;
private JRadioButton rdbtnX3;
private JRadioButton rdbtnLog;
private JRadioButton rdbtnSqrt;
private int operation;

private double scalar = 1.0;
private double offset = 0.0;

/**
 * Launch the application.
 */
public static void main(String[] args) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				Application window = new Application();
				window.frame.setVisible(true);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	});
```

```
}

/**
 * Constructs an application for graphing.
 */
public Application() {
	initialize();
}

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
	frame = new JFrame();
	frame.setBounds(100, 100, 459, 637);
	frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	frame.getContentPane().setLayout(null);

	JPanel canvas = new JPanel() {
		@Override

		protected void paintComponent(Graphics g) {
			super.paintComponent(g);
			drawGrid(g);

			if (operation==1) {
				drawX(g);
			} else if (operation==2) {
```

```
                drawX2(g);
            } else if (operation==3) {
                drawX3(g);
            } else if (operation==4) {
                drawLogX(g);
            } else if (operation==5) {
                drawSqrtX(g);
            }

        }
};
canvas.setBackground(Color.WHITE);
canvas.setBounds(30,200,400,400);
frame.getContentPane().add(canvas);

JLabel lblScalar = new JLabel("Scalar (m)");
lblScalar.setForeground(Color.BLACK);
lblScalar.setFont(new Font("Lucida Grande", Font.PLAIN, 13));
lblScalar.setBackground(Color.BLACK);
lblScalar.setBounds(17, 17, 81, 43);
frame.getContentPane().add(lblScalar);

txtScalar = new JTextField();
txtScalar.setBounds(89, 25, 121, 29);
frame.getContentPane().add(txtScalar);
txtScalar.setColumns(10);

JLabel lblOffset = new JLabel("Offset (b)");
lblOffset.setBounds(231, 30, 61, 16);
```

```
frame.getContentPane().add(lblOffset);

textOffset = new JTextField();
textOffset.setColumns(10);
textOffset.setBounds(304, 24, 126, 29);
frame.getContentPane().add(textOffset);

JRadioButton rdbtnX = new JRadioButton("X");
rdbtnX.setBounds(17, 74, 51, 23);
frame.getContentPane().add(rdbtnX);

JRadioButton rdbtnX2 = new JRadioButton("X^2");
rdbtnX2.setBounds(17, 109, 61, 23);
frame.getContentPane().add(rdbtnX2);

JRadioButton rdbtnLog = new JRadioButton("Log(X)");
rdbtnLog.setBounds(17, 144, 81, 23);
frame.getContentPane().add(rdbtnLog);

JRadioButton rdbtnX3 = new JRadioButton("X^3");
rdbtnX3.setBounds(133, 109, 75, 23);
frame.getContentPane().add(rdbtnX3);

JRadioButton rdbtnSqrt = new JRadioButton("Sqrt(X)");
rdbtnSqrt.setBounds(133, 144, 87, 23);
frame.getContentPane().add(rdbtnSqrt);

rdbtnX.addActionListener(this);
rdbtnX2.addActionListener(this);
```

```
rdbtnX3.addActionListener(this);
rdbtnLog.addActionListener(this);
rdbtnSqrt.addActionListener(this);



JButton btnCheck = new JButton("Graph");
btnCheck.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {

		if (isNumeric(txtScalar.getText())) {
			scalar = Double.parseDouble(txtScalar.getText());
		} else {
			scalar = 1.0;
		}

		if(isNumeric(textOffset.getText())) {
			offset = Double.parseDouble(textOffset.getText());
		} else {
			offset = 0.0;
		}

		if (rdbtnX.isSelected()) {
			rdbtnX2.setSelected(false);
			rdbtnX3.setSelected(false);
			rdbtnLog.setSelected(false);
			rdbtnSqrt.setSelected(false);
		operation = 1;
	}
```

```
if (rdbtnX2.isSelected()) {
        rdbtnX.setSelected(false);
        rdbtnX3.setSelected(false);
        rdbtnLog.setSelected(false);
        rdbtnSqrt.setSelected(false);
operation = 2;
}

if (rdbtnX3.isSelected()) {
        rdbtnX.setSelected(false);
        rdbtnX2.setSelected(false);
        rdbtnLog.setSelected(false);
        rdbtnSqrt.setSelected(false);
operation = 3;
}

if (rdbtnLog.isSelected()) {
        rdbtnX.setSelected(false);
        rdbtnX2.setSelected(false);
        rdbtnX3.setSelected(false);
        rdbtnSqrt.setSelected(false);
operation = 4;
}

if (rdbtnSqrt.isSelected()) {
        rdbtnX.setSelected(false);
        rdbtnX2.setSelected(false);
        rdbtnX3.setSelected(false);
```

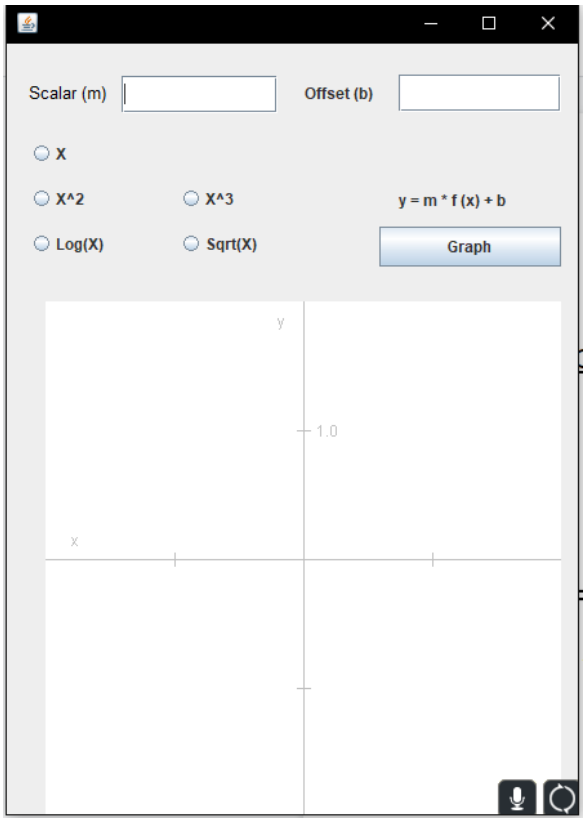
```
                    rdbtnLog.setSelected(false);
                operation = 5;
                }

                canvas.repaint();

            }
        });

        btnCheck.setBounds(289, 142, 141, 31);
        frame.getContentPane().add(btnCheck);

        JLabel lblNewLabel = new JLabel("y = m * f (x) + b");
        lblNewLabel.setBounds(304, 113, 113, 16);
        frame.getContentPane().add(lblNewLabel);

        }

private void drawGrid(Graphics g) {
        g.setColor(Color.LIGHT_GRAY);
        g.drawString("x", 20, 190);
        g.drawString("y",180, 20);
        g.drawString("1.0", 210, 105);
        g.drawLine(200,0,200,400);
        g.drawLine(0,200,400,200);
        g.drawLine(195,300,205,300);
        g.drawLine(195,100,205,100);
        g.drawLine(100,195,100,205);
        g.drawLine(300,195,300,205);
```

```
}


public static boolean isNumeric(String string) {
   if (string == null) {
      return false;
   }
   try {
      double d = Double.parseDouble(string);
   } catch (NumberFormatException nfe) {
      return false;
   }
   return true;
}

@Override
public void actionPerformed(ActionEvent e) {

}

private void drawX(Graphics g) {
        double dx = 0.001;
        for(double x=-2; x<=2; x=x+dx) {
                double y = scalar*x+offset;
                int cX = (int) Math.round((x*100)+200);
                int cY = (int) Math.round((-y*100)+200);
                g.setColor(Color.RED);
                g.drawLine(cX, cY, cX, cY);
        }
```

```
	}

	private void drawX2(Graphics g) {
		double dx = 0.001;                                    // whatever is set as x increment
		for(double x=-2; x<=2; x=x+dx) { // loop that sets X from -2 to 2
			double y = scalar*(x*x)+offset;                    // conceptual Y (solve based on X)
			int cX = (int) Math.round((x*100)+200);
			int cY = (int) Math.round((-y*100)+200);
			g.setColor(Color.RED);
			g.drawLine(cX,  cY, cX, cY);
		}
	}

	private void drawX3(Graphics g) {
		double dx = 0.001;
		for(double x=-2; x<=2; x=x+dx) {
			double y = scalar*(x*x*x)+offset;
			int cX = (int) Math.round((x*100)+200);
			int cY = (int) Math.round((-y*100)+200);
			g.setColor(Color.RED);
			g.drawLine(cX,  cY, cX, cY);
		}
	}


	private void drawLogX(Graphics g) {
		double dx = 0.001;
		for(double x=-2; x<=2; x=x+dx) {
			double y = scalar*(Math.log(x))+offset;
```

```
                int cX = (int) Math.round((x*100)+200);
                int cY = (int) Math.round((-y*100)+200);
                g.setColor(Color.RED);
                g.drawLine(cX,  cY, cX, cY);
            }
        }

        private void drawSqrtX(Graphics g) {
            double dx = 0.001;
            for(double x=-2; x<=2; x=x+dx) {
                double y = scalar*(Math.sqrt(x))+offset;
                int cX = (int) Math.round((x*100)+200);
                int cY = (int) Math.round((-y*100)+200);
                g.setColor(Color.RED);
                g.drawLine(cX,  cY, cX, cY);
            }
        }

}
```

Slope-Intercept Form (y=mx+b) Test.

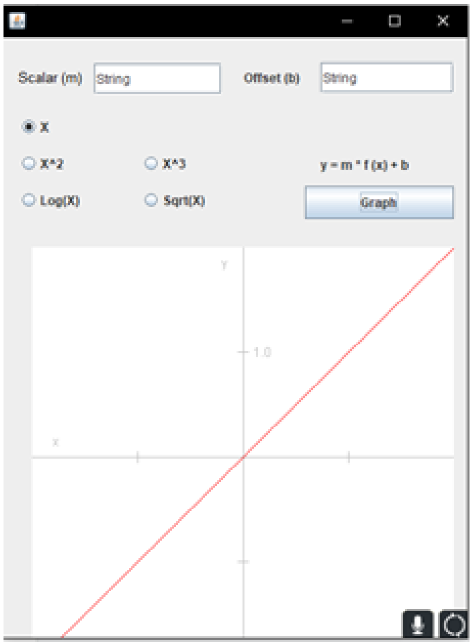


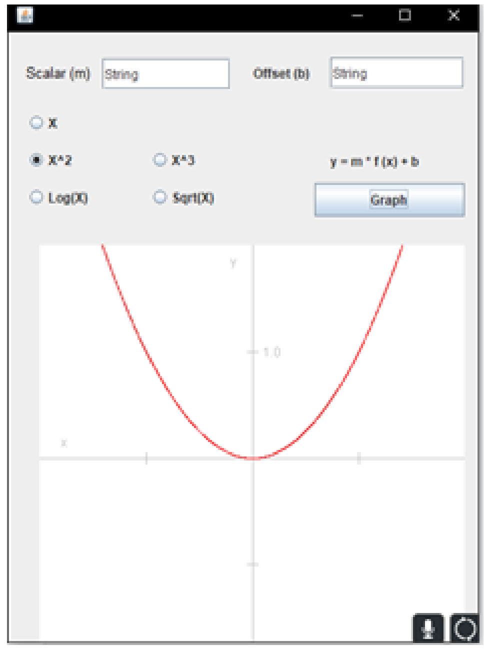
M = Slope of the graph.

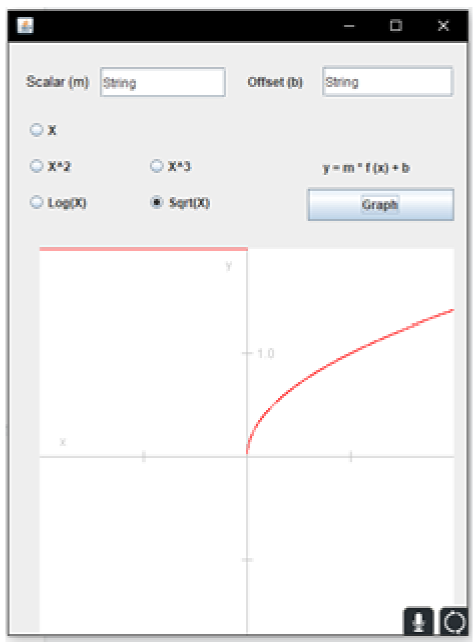
B = Y-intercept of the graph.

X = X-intercept of the graph.

The Graphing calculator does NOT accept any type of string or input other than numbers. When the user does try to input a string and computes it into the calculator, it presents the user with the standard graph of the function.







Section 1: X

Test Case: Basic Graphical Functions With Whole Numbers and floating point numbers

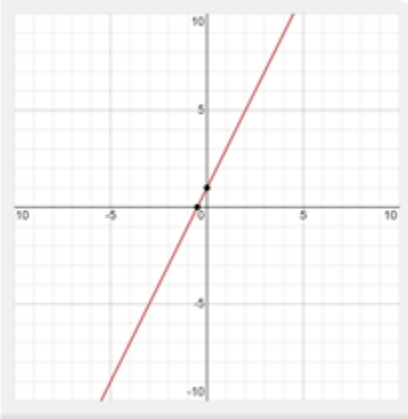
Test1 A:

Objective: Test the calculator with small whole numbers that do not exceed the canvas and negative numbers.

Graph: y = 2x + 1

Expected Graph:

X Axis	Y Axis
-0.5	1



Actual Graph:

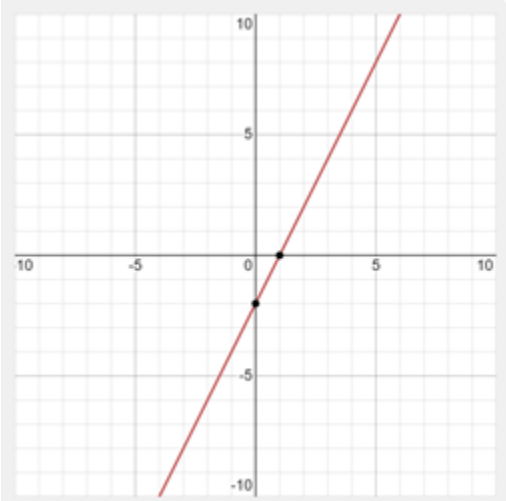
X Axis	Y Axis
-0.5	1

Test 1 B

Graph: y = 2x - 2

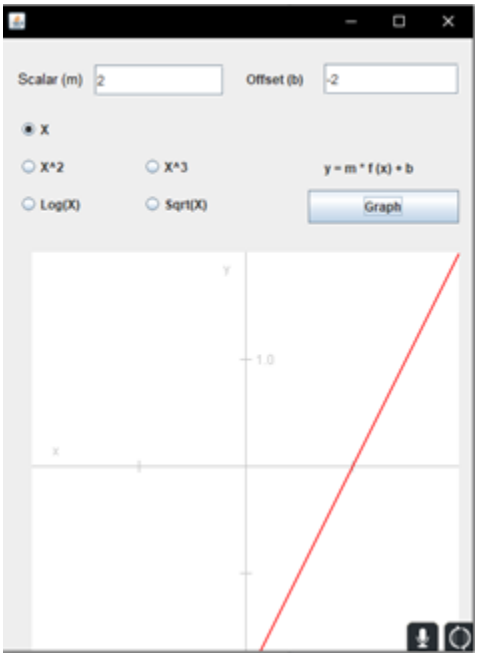
Expected Graph:

X Axis	Y Axis
1	-2



Actual Graph:

X Axis	Y Axis
1	-2



Test Case: Test numbers higher than the canvas limit

Test 2 A

Graph: y= 18x + 23

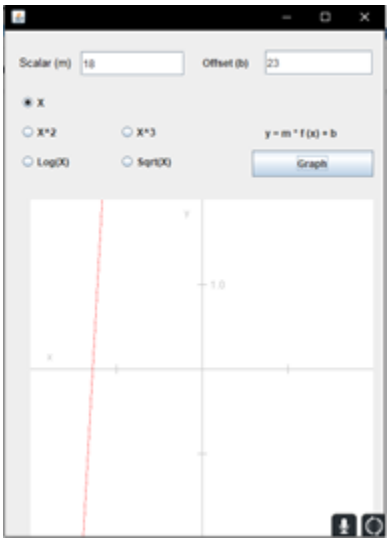
Objective: Test the calculator with higher whole numbers that **do** exceed the canvas.

Expected Graph:

X Axis	Y Axis
-1.27	23

Actual Graph:

X Axis	Y Axis
-1.27	??

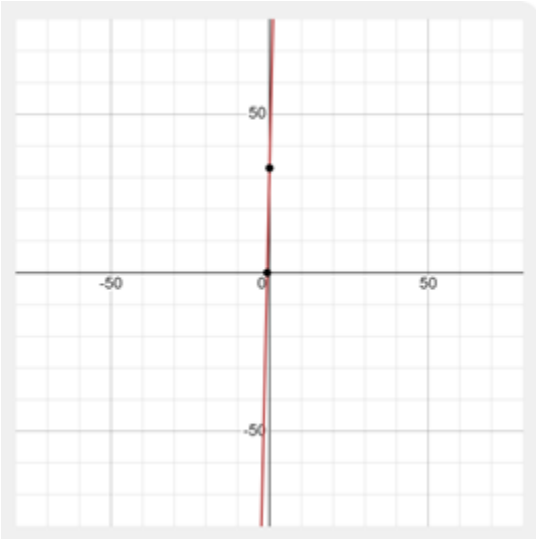


Test 2 B

Graph: y= 44x + 33

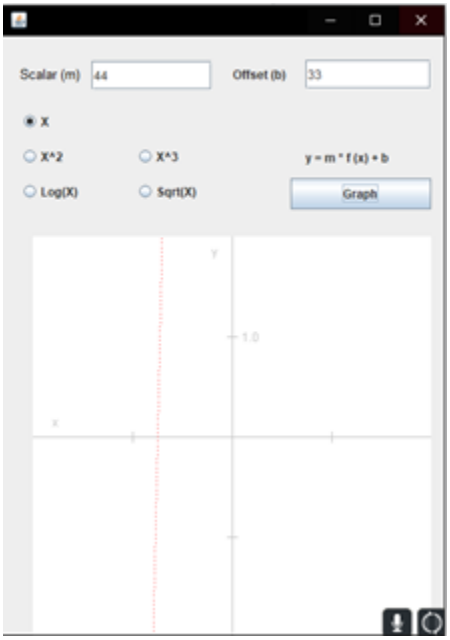
Expected Graph:

X Axis	Y Axis
-0.75	33



Actual Graph:

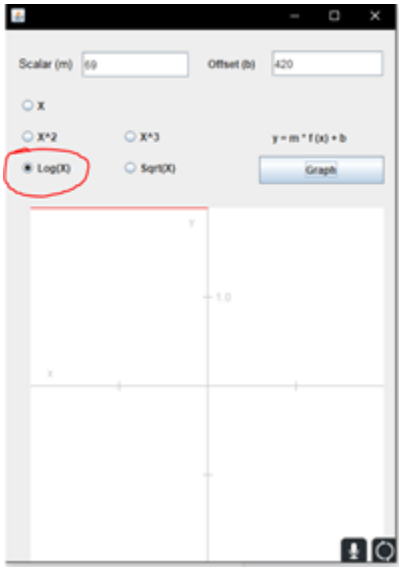
X Axis	Y Axis
-0.75	??



Observations: When using small whole positive and negative numbers as well as floating-point numbers that do not exceed the canvas, the calculator graphs the linear equations accordingly with the same x axis and y axis. However, something to keep in mind is that the calculator does not adapt to the Y axis when it exceeds the canvas because the canvas is only allowed to show the effective intervals from -2 to +2 and when trying to input a number greater than 2, it results in the Y and X axis not being represented as it should.

Errors: None, the graphing calculator was designed to be simple in that it only scales the canvas to show the effective interval from -2 to +2 and it was not designed to adapt its Y axis depending on the numbers that the user inputs.

Discovery: When inputting very large numbers in the equation form of y = 69x + 420 for example, the graphing calculator stops working completely in all of the mathematical choices BUT Log(x) which does not really do anything but show a little line on the top left.



When trying to graph an equation with or without a negative fractional slope, the Graphing calculator cannot seem to be able to graph it. For example:

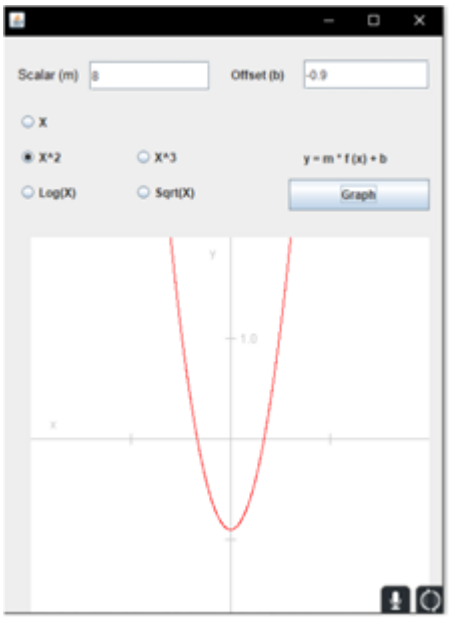
The equation: y = -7/3x + 11 should graph the following ….

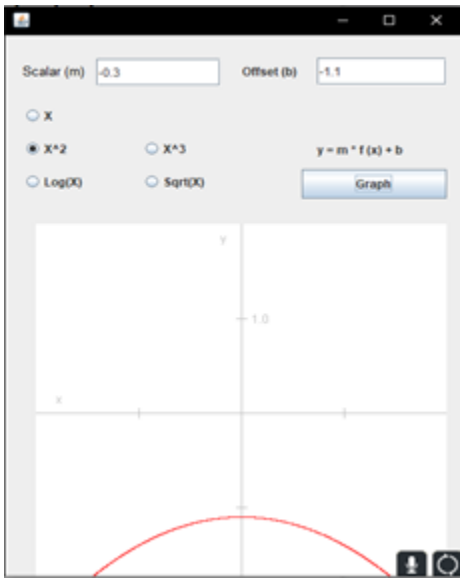
While it actually doesn't really show anything in the application itself:

Section 2: x^2

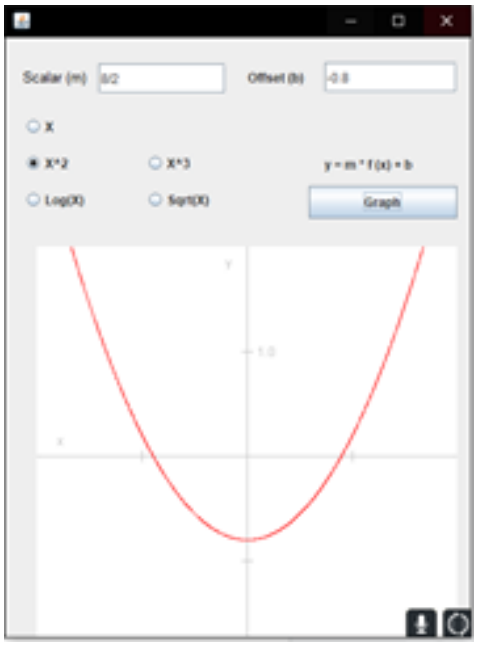
Test Case: Basic Graphical Functions With Whole Numbers and floating point numbers

When inputting negative and positive floating-point numbers and whole numbers with the x^2 function, the function works together to form a parabola. The m representing how wide the parabola should be and b representing the position of the parabola on the Y axis:

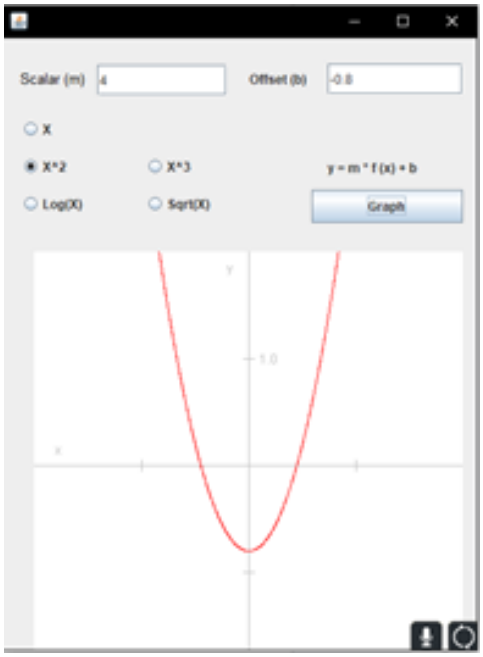




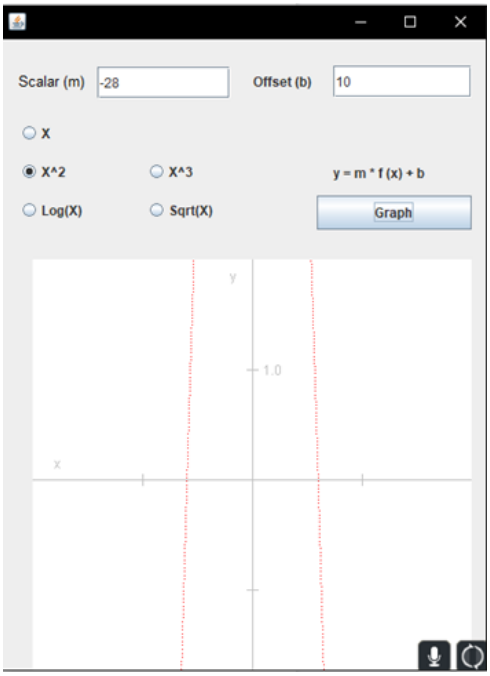
When trying to implement fractions into the ring, it does not seem to be able to calculate them and what it does instead is to put the parabola into the function's standard mode regardless of the input of the fraction.



When instead it should be like this:



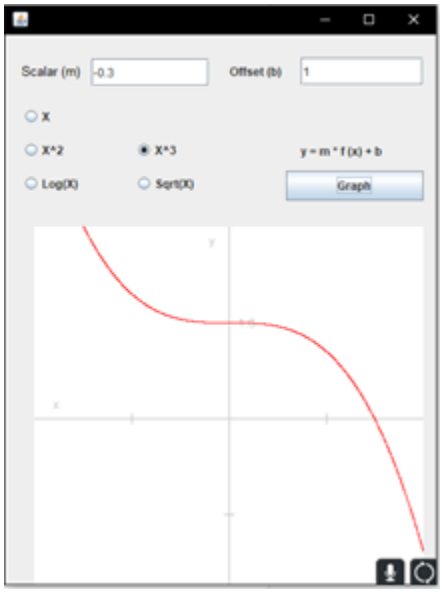
Unlike the section with the x, after testing with numbers higher than that of the canvas, it proves that it still works regardless of how high the number is. As long as the M is set to where the viewer can actually see the parabola it works.

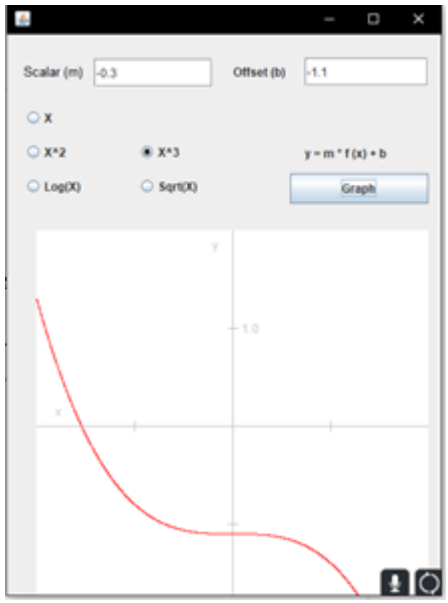


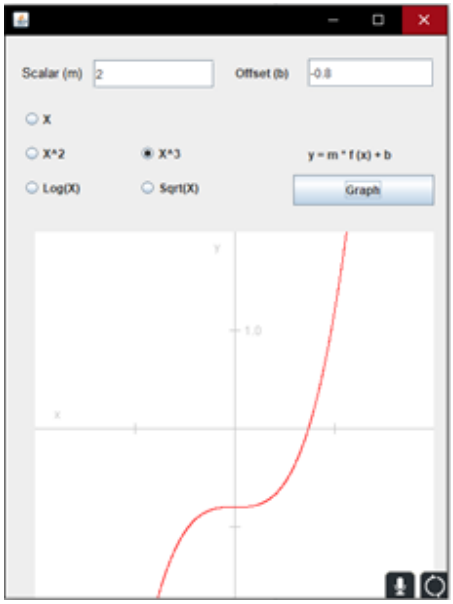
Section 3: x^3

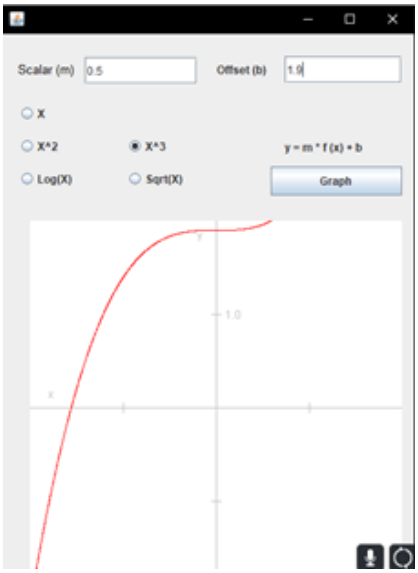
Test Case: Basic Graphical Functions With Whole Numbers and floating point numbers

When inputting negative and positive floating-point numbers as well as whole numbers with the x^3 function they all work together to form the line. The m in this instance representing how straight the line should be and b representing the position of the parabola on the Y axis:





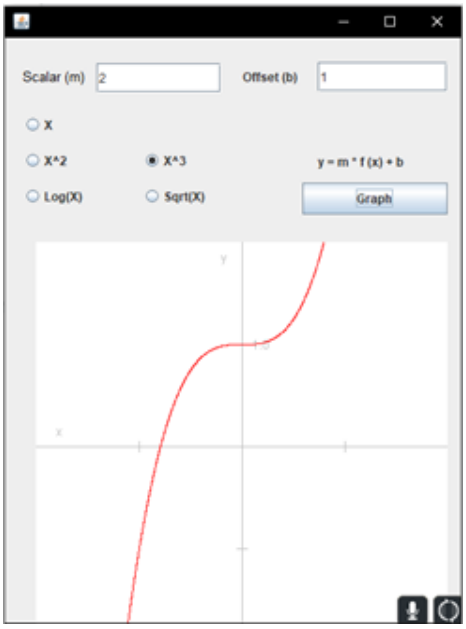




Fractions:

After testing fractions, the same type of error that occurred with the parabola occurs with the x^3 line.

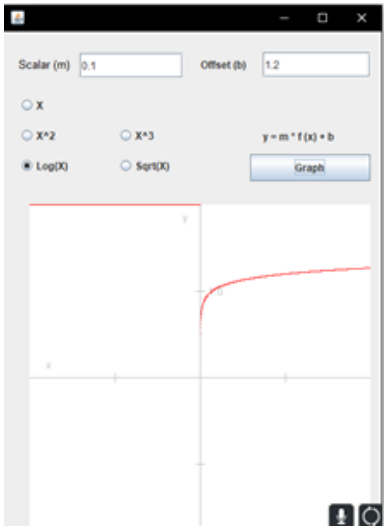
Instead of the line being represented as this:

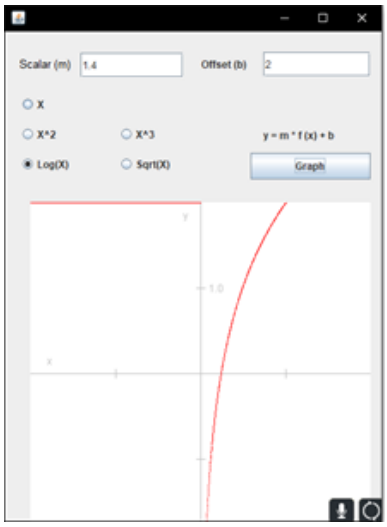


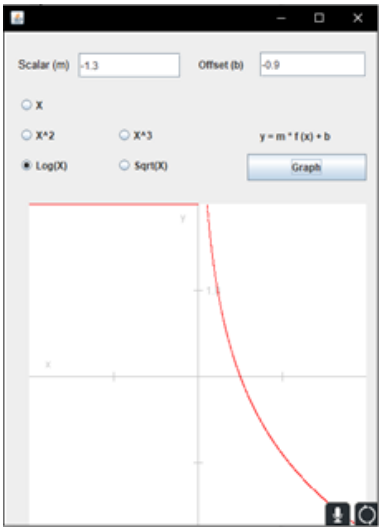
It goes back to the standard form of the X^3:

Lastly after testing the Log and Square root functions no errors were found and when it came to fractions, the same error was present in that it would reverse itself to standard graph.

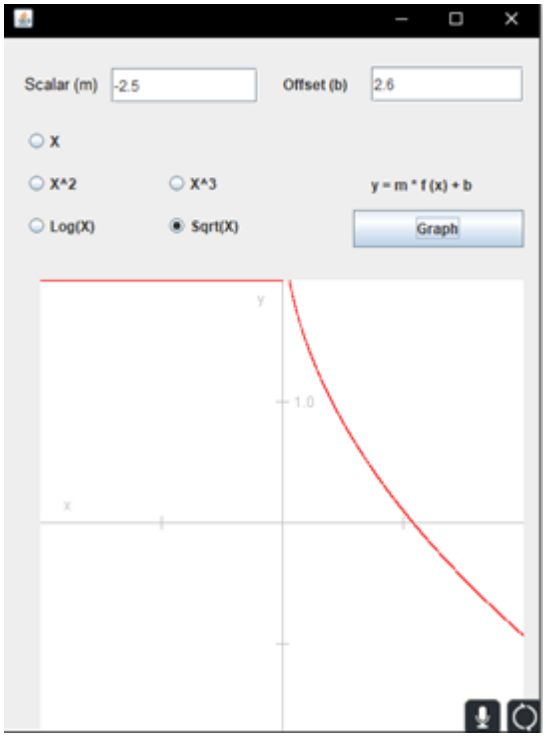
Log Tests:

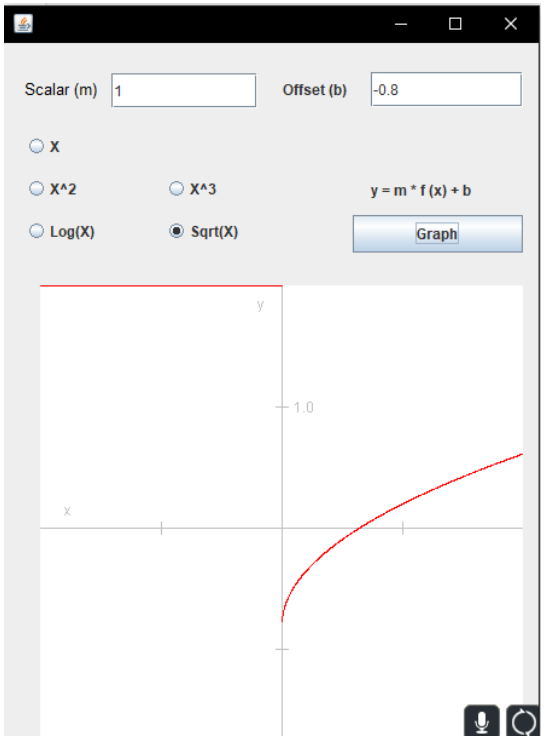






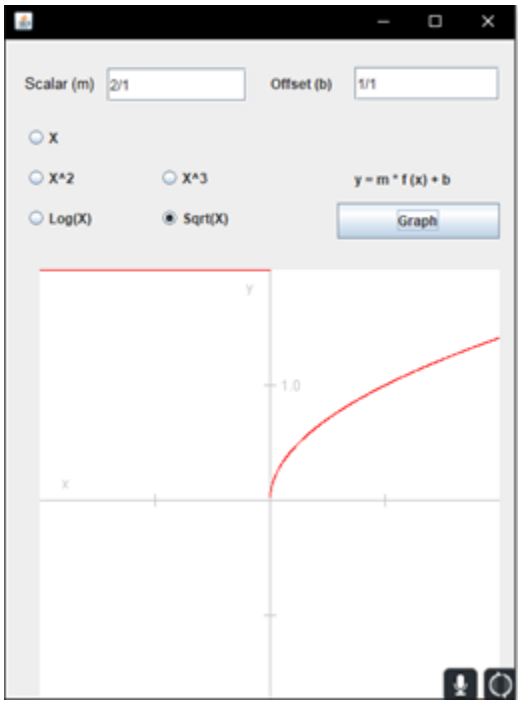
Square root tests:



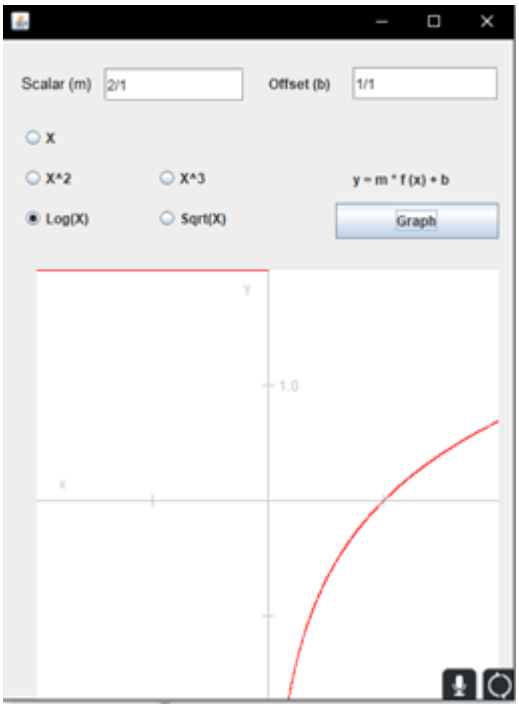


Fractions:

Square root:



Log:



Fractions have no effect on either of them.

In conclusion, close to zero errors within the program and the calculator itself. Only major problem was the inability to properly do fractions.