DUE: 1/24/2019

Project #1—Geometries with JTabbedPane

A number of exercises in our text inspired this project. Specifically, you will find the following exercises at end of Chapter 3 of particular interest: 3.1, 3.3, and 3.25. You may think of this project as three separate problems all solved within one GUI: hence the requirement for use of JTabbedPane with three separate JPanels for tabs positioned to the right in the GUI shown below. Each JPanel displays the relevant components for solving the particular problem. Note however, that the names for each component must be different across all JPanels. For example, there are three Calculate JButtons but all three must have different names, e.g., quadraticJButton, cramersJButton, and linesJButton (note also the naming convention followed here). Here are the statements for the three problems:

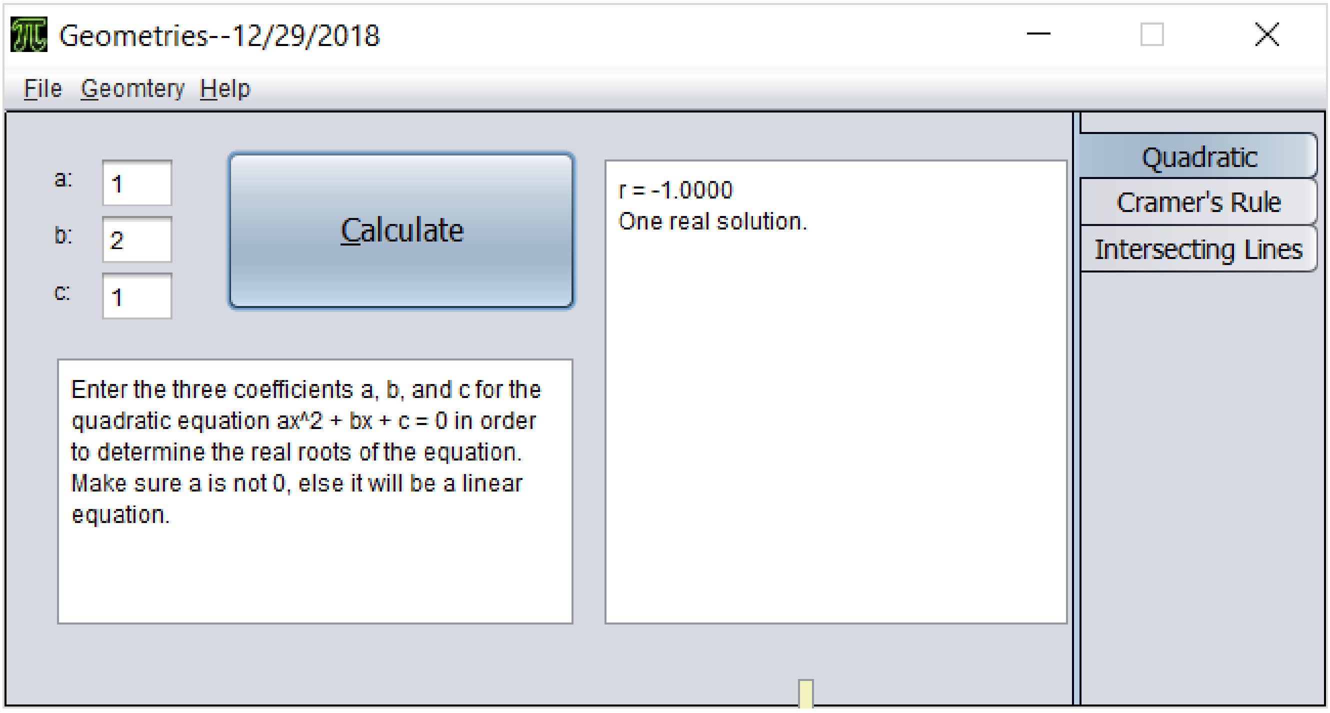
1. This is exactly the statement from programming exercise 3.1. (Algebra: solve quadratic equations) The two roots of a quadratic equation $ax^2 + bx + c = 0$ can be obtained using the following quadratic formula:

   $$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

   Note: $b^2 - 4ac$ is called the discriminant of the quadratic equation. If it is positive, the equation has two real roots. If it is zero, the equation has one root. If it is negative, the equation has no real roots.

   For this first part of the project you are to write a program that prompts the user to enter values for a, b, and c and displays the result based on the discriminant. If the discriminant is positive, it displays the two roots. If the discriminant is 0, it displays one root. Otherwise, it displays “The equation has no real roots”. The program should verify for valid input (the coefficients a, b, and c should be double type with $a \neq 0$) and that the equation is quadratic, i.e. $a \neq 0$.

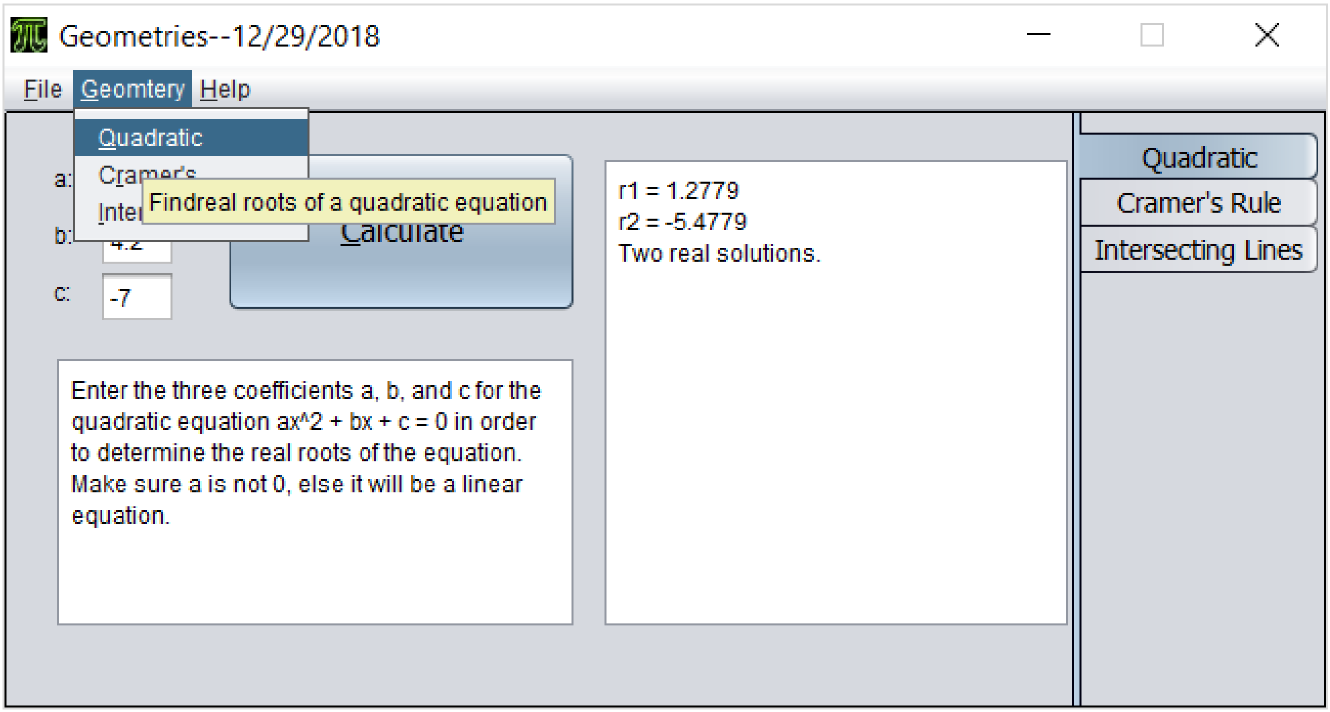
   Consider designing the first JPanel of the required JTabbedPane of the GUI to look like this (with sample runs):



a: 1
b: 2
c: 1

Enter the three coefficients a, b, and c for the quadratic equation ax^2 + bx + c = 0 in order to determine the real roots of the equation. Make sure a is not 0, else it will be a linear equation.

```
r = -1.0000
One real solution.
```



c: -7

Enter the three coefficients a, b, and c for the quadratic equation ax^2 + bx + c = 0 in order to determine the real roots of the equation. Make sure a is not 0, else it will be a linear equation.

```
r1 = 1.2779
r2 = -5.4779
Two real solutions.
```

2. The second tab of the JTabbedPane illustrates the second problem, similar to programming exercise 3.3 in your text: (Algebra: solve 2 x 2 system of linear equations) A [2x2 system of] linear equation[s] can be solved using Cramer’s rule given in Programming Exercise 1.13. Write a program that prompts the user to enter [the coefficients for the two lines,] a, b, c, d, e, and f and displays the result. If ad - bc is 0, report that “The equation has no solution.”

A 2x2 system of linear equations consists of two equations with two unknowns, typically written in the standard form as

$$\begin{aligned} a_1 x + b_1 y &= c_1 \\ a_2 x + b_2 y &= c_2 \end{aligned} \qquad (*)$$

where a1, b1, c1, a2, b2 and c2 are real numbers and x and y are the unknowns. In Euclidian plane, the system consists of two lines in the plane. The point of intersection, if there is one, is called a solution to the system.

There are a number of ways to find the solution of a 2x2 system: graph, substitution, elimination (addition), matrices, and others. For this project, the second tab solves the 2x2 system of linear equations via Cramer's Rule. The choice for Cramer's Rule is because it extends easily to a 3x3, 4x4, and so on systems.

It is important to realize that there are three (3) possibilities for such a system:

a. The system is ***consistent*** and has one solution. For example, the system

   2x – y = 5
   x + 6y = –4

   has the point (2, –1) as its only solution. The system is consistent if the two lines have different slopes or alternatively if $\frac{a_1}{a_2} \neq \frac{b_1}{b_2}$.

b. The system is ***inconsistent*** if it has no solution. For example, the following two lines are parallel and hence have no point of intersection.

   6x – 4y = 10
   6y – 9x = –7

   The system is inconsistent if the two lines have same slopes but different y-intercepts or alternatively if $\frac{a_1}{a_2} = \frac{b_1}{b_2} \neq \frac{c_1}{c_2}$.

c. The system is ***dependent*** if the lines are the same line in which case there are infinitely many solutions. For example, the following two are the same line.

   6x – 4y = 10
   6y – 9x = –15

The system is dependent if the two lines have same slopes and same y-intercepts or alternatively if $\frac{a_1}{a_2} = \frac{b_1}{b_2} = \frac{c_1}{c_2}$.

The solution of a 2x2 system of linear equations defined above (*) via Cramer's Rule is based on calculating three 2x2 determinants. A 2x2 determinant is defined as follows:

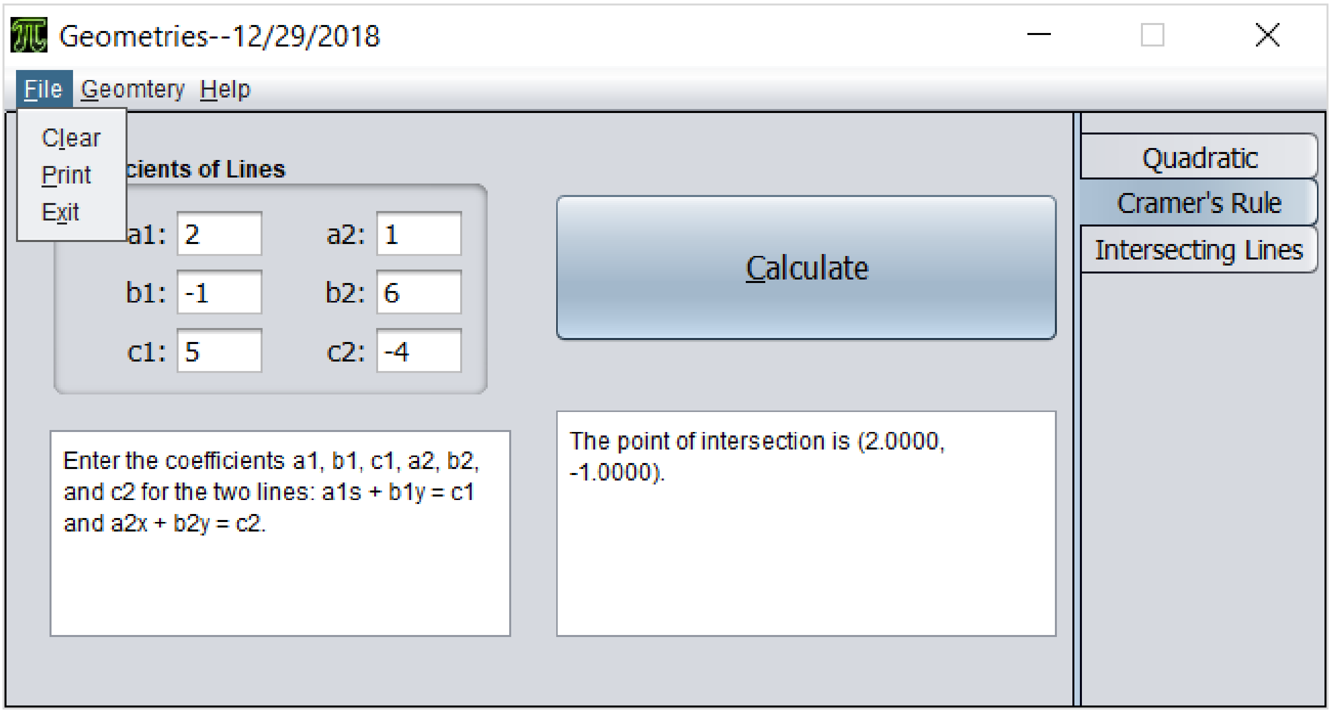
$$D = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

The solution, (x, y), if it exists, defined via Cramer's Rule is obtained by

$$x = \frac{D_x}{D}, \quad y = \frac{D_y}{D}, \text{ where } D_x = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}, \quad D_y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}, \quad and \quad D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

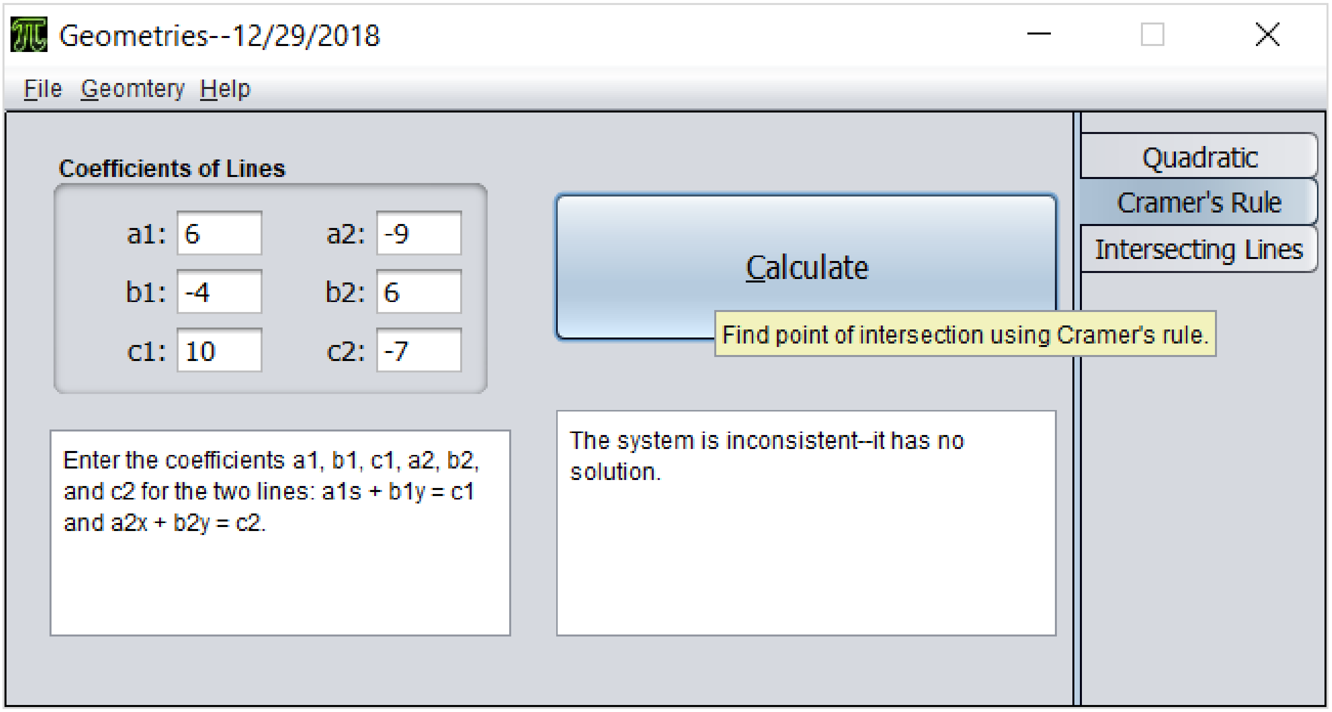
This part of the project solves a given 2x2 system of linear equations. Your program should allow the user to enter the six coefficients, a1, b1, c1, a2, b2 and c2 and find the solution using Cramer's formulas, if it exists. If the system is inconsistent or dependent (D = 0) it should be able to distinguish between the two.

Consider designing the second tab of the required JTabbedPane of the GUI to look like this and extend the solution to distinguish between inconsistent and dependent case:



Enter the coefficients a1, b1, c1, a2, b2, and c2 for the two lines: a1s + b1y = c1 and a2x + b2y = c2.

The point of intersection is (2.0000, -1.0000).

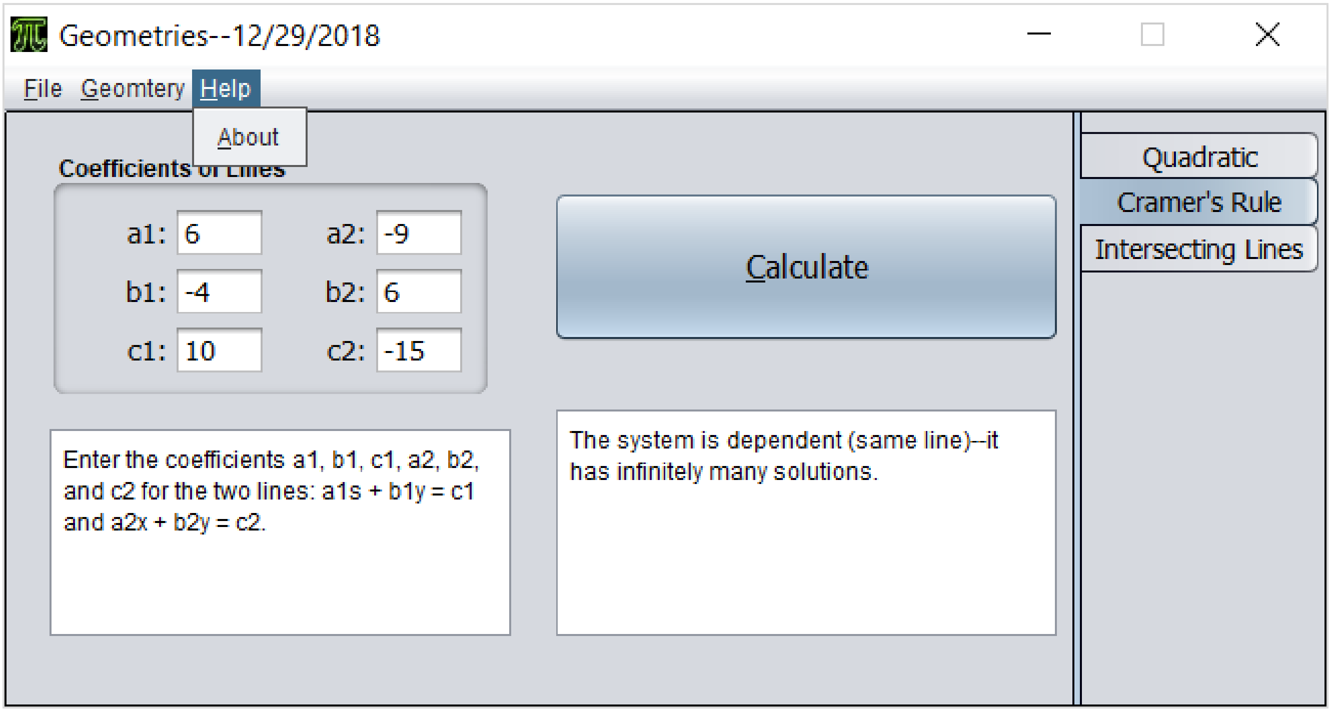


Coefficients of Lines

a1:	6	a2:	-9
b1:	-4	b2:	6
c1:	10	c2:	-7

Enter the coefficients a1, b1, c1, a2, b2, and c2 for the two lines: a1s + b1y = c1 and a2x + b2y = c2.

The system is inconsistent--it has no solution.



Coefficients of Lines

a1:	6	a2:	-9
b1:	-4	b2:	6
c1:	10	c2:	-15

Enter the coefficients a1, b1, c1, a2, b2, and c2 for the two lines: a1s + b1y = c1 and a2x + b2y = c2.

The system is dependent (same line)--it has infinitely many solutions.

3. The final third problem is exactly programming exercise 3.25 (Geometry: intersecting point) Two points on line 1 are given as (x1, y1) and (x2, y2) and on line 2 as (x3, y3) and (x4, y4), as shown in Figure 3.8a–b. The intersecting point of the two lines can be found by solving the following linear equation:

$$(y_1 - y_2)x - (x_1 - x_2)y = (y_1 - y_2)x_1 - (x_1 - x_2)y_1$$
$$(y_3 - y_4)x - (x_3 - x_4)y = (y_3 - y_4)x_3 - (x_3 - x_4)y_3$$

This linear equation can be solved using Cramer's rule (see Programming Exercise 3.3). If the equation has no solutions, the two lines are parallel (Figure 3.8c).

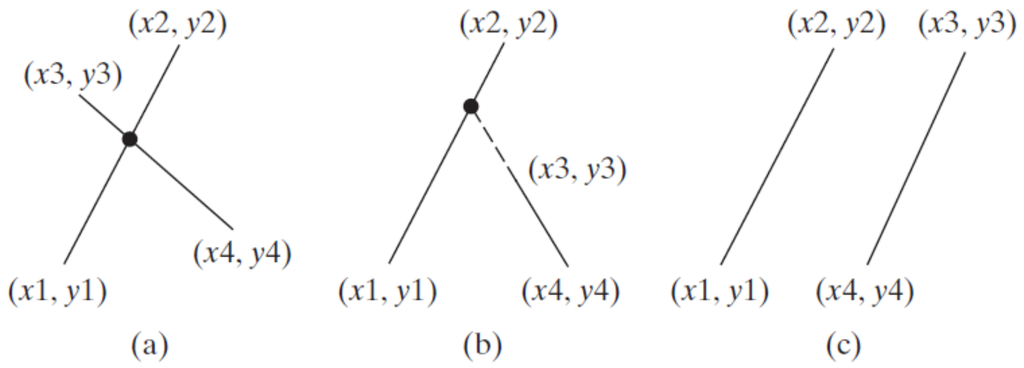
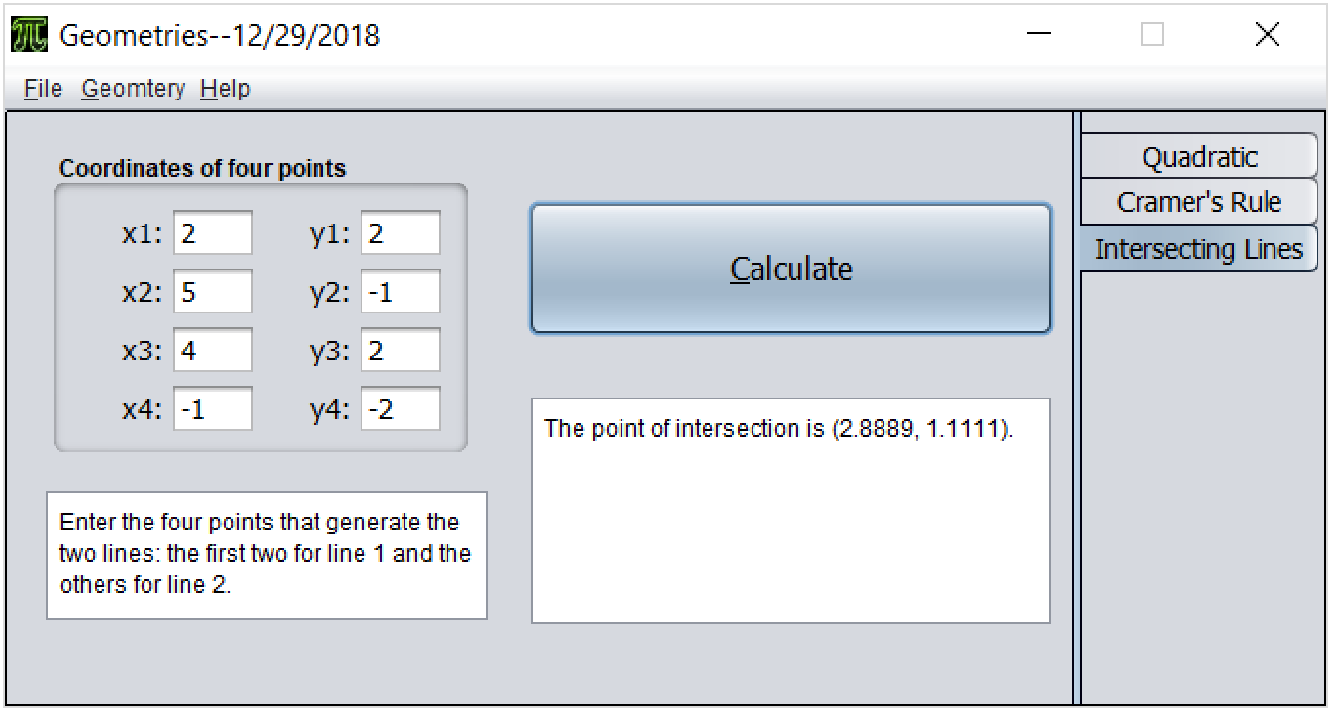


FIGURE 3.8 Two lines intersect in (a and b) and two lines are parallel in (c).

For this third part, write a Java program that prompts the user to enter four points and displays the intersecting point. Consider designing the third tab of the required JTabbedPane of the GUI to look like this:

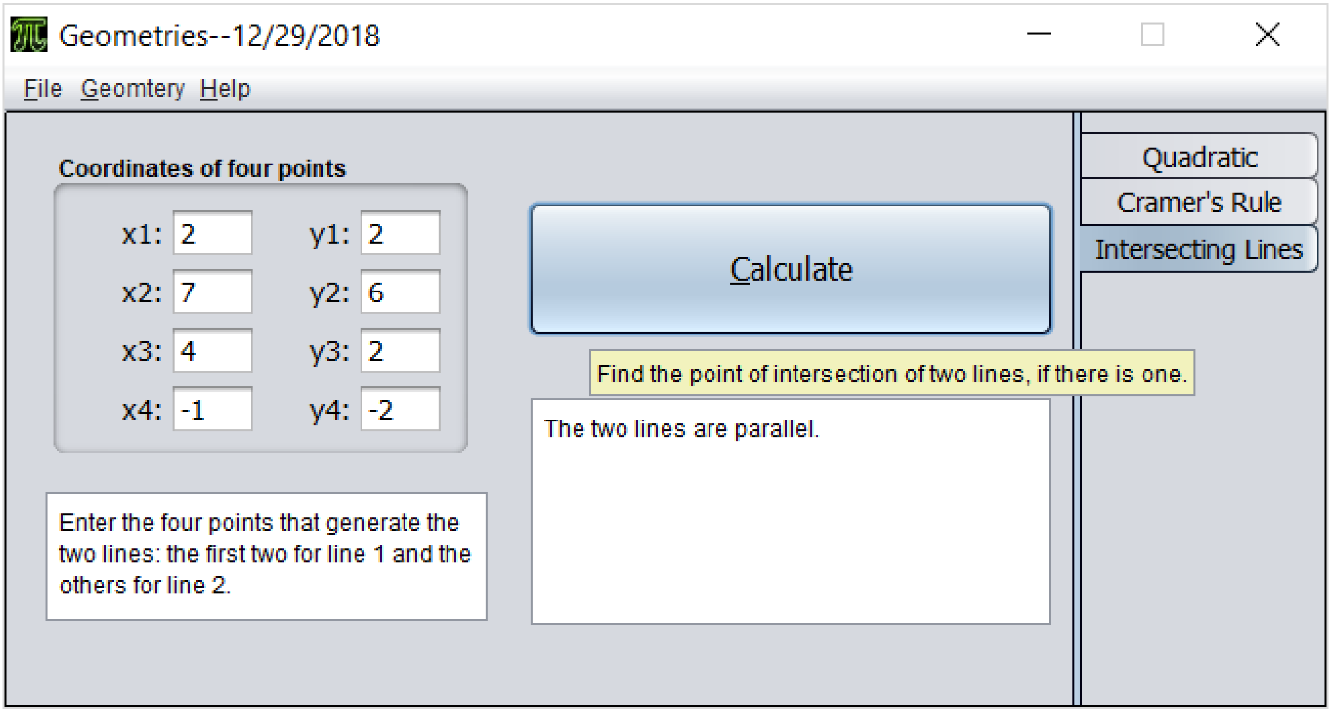


Coordinates of four points

x1: 2 y1: 2
x2: 5 y2: -1
x3: 4 y3: 2
x4: -1 y4: -2

Enter the four points that generate the two lines: the first two for line 1 and the others for line 2.

The point of intersection is (2.8889, 1.1111).


Geometries--12/29/2018

Coordinates of four points

x1: 2 y1: 2
x2: 7 y2: 6
x3: 4 y3: 2
x4: -1 y4: -2

Find the point of intersection of two lines, if there is one.

The two lines are parallel.

Enter the four points that generate the two lines: the first two for line 1 and the others for line 2.

Write a Java GUI-based program that uses a JTabbedPane nested in a JFrame with three JPanels for tabs to solve the three problems as described above.

Requirements for Project 1. Your project must contain/do:

1. The project is worth 30 points and it will be graded on correctness, completeness, presentation, comments, punctuality, and the like.
2. The following are required controls/features:
   a. The GUI should contain all of the controls and images as shown in the example GUI above. Specifically, you must use a JTabbedPane with three JPanels inside a JFrame. You are encouraged to find free art images (small) of your own, and your own logo and form icon images.
   b. Use JTextFields for input of number of sides and length of each side. Use JLabels for the prompts for each input value. Use JTextAreas to display results.
   c. Pressing the enter key should fire the calculate button within the selected JPanel which displays the calculated values (in non-editable controls—preferably JTextAreas).
   d. As the form loads, the current date should be displayed in the title bar of the form. The form should be centered as it is displayed and should not be resizable.
   e. The JMenuBar should contain JMenus for at least File (with JMenuItems for Clear, Print and Exit), Geometries (with JMenuItems for Quadratic, Cramer's Rule, and Intersecting Lines to select the corresponding tab), and Help (with JMenuItem for About form). They should have accelerator ("hot") key assigned to them (set

Mnemonic property to appropriate letter) in order to use the keyboard's ALT key with the appropriate letter assigned.

f. Use the Math class methods and constants (e.g., Math.PI, Math.sqrt, etc.).
g. Provide Tooltip to help the user navigate throughout the form.
h. When each JPanel in the JTabbedPane is shown, the corresponding Calculate JButton should become default (so that the Enter key fires it).

3. Follow the quality coding style and follow Java coding conventions. This includes but is not limited to:
   a. Avoid methods with more than approximately 30 lines.
   b. Avoid excessive global or class (static) variables.
   c. No magic numbers--use finals for constants.
   d. Upper/lowercase naming convention (ClassName, methodName, variableName, CONSTANT_NAME).
   e. White space and indentation ({} line up horizontally or vertically).
4. Make sure that your name, project number, and due date appear as comments in your code (preferably on top). Use Javadocs for this purpose.
5. Meaningful About form that describes the functionality of the project, warnings, copyright, etc.
6. Splash screen.
7. Two print capabilities: use the provided PrintUtilities class to print the entire form and print the content of the displayed JTextArea.
8. Validation of input. Catch errors generated for illegal type and range for inputs. I suggest you use try/catch blocks for this project.
9. Use Javadocs extensively--read the document named Javadocs and follow the example therein. Specifically,
   a. Each class and method must have documentation that includes description, purpose and appropriate tags.
   b. Each method must have comments for each parameter (if they have any).
   c. Each method must have comments for return value (unless it is void).
   d. Each method could have pre and post-condition comments.
   e. In general, the method comments have the following structure

```
/**
   name and description of method (mandatory)
   @author
   @param parameter_1 description of parameter 1
   @param parameter_2 description of parameter
    . . .
   @return description of return value if there is any
*/
```

10. Submit the zipped file containing all files and folders for the project via Blackboard (click on the Project 1 link to attach the zipped project file). A good file name might be Your_Last_Name_Project 1—Geometries_CS141A.zip.

EXTRA CREDIT (Optional—3 or more points). Add an additional tab or two to solve exercises 3.28 and 3.29. Graphing the points, lines, parabola should bring even more extra credit points. Use of additional classes (Line, Quadratic, Validation, etc.) can also bring extra credit.

Project Guidelines

The following list reviews the GUI design and programming guidelines you have learned so far. You can use this list to verify that the applications and projects you create adhere to the standards outlined herein.

- Information should flow either vertically or horizontally, with the most important information always located in the upper-left corner of the screen.
- Maintain a consistent margin of two or three dots from the edge of the window.
- Related controls should be grouped together using either white space or a JPanel.
- Position related controls with equal spacing.
- Buttons should either be centered along the bottom of the screen or stacked in either the upper-right or the lower-right corner.
- If the buttons are centered along the bottom of the screen, then each button should be the same height; their widths, however, may vary.
- If the buttons are stacked in either the upper-right or lower-right corner of the screen, then each button should be the same height and the same width.
- Use no more than six buttons on a form.
- The most commonly used button should be placed first and should be set as a default (the Enter key should fire it).
- Button captions should:
  - ✓ be meaningful
  - ✓ be from one to three words appear on one line
  - ✓ be entered using book title capitalization
  - ✓ contain mnemonic settings “hot” keys activation
- Use labels to identify the controls in the interface.
- Identifying labels should:
  - ✓ be from one to three words
  - ✓ appear on one line
  - ✓ be aligned by their left borders
  - ✓ be positioned either above or to the left of the control that they identify
  - ✓ end with a colon (:)
  - ✓ be entered using sentence capitalization
- Align labels and controls to minimize the number of different margins.
- If you use a graphic in the interface, use a small one and place it in a location that will not distract the user.
- Use no more than two different font sizes, which should be 10, 12 or 14 points.
- Use only one font type, which should be a sans serif font, in the interface.
- Avoid using italics and underlining.

- Use light colors such as white, off-white, light gray, pale blue, or pale yellow for an application's background, and very dark color such as black for the text. Alternately, use dark colors for the form's background with light colors for the text—contrast is the name of the game.
- Use color sparingly and do not use it as the only means of identification for an element in the interface.
- Document the program internally--use Javadocs extensively.
- Use variables to control the preciseness of numbers in the interface.
- Test the application with both valid and invalid data (test the application without entering any data; also test it by entering letters where numbers are expected). Test the application for correct results with known data.
- Center a form by including the appropriate formulas in the form's Load event.
- Remove excess or duplicated code from the application.
- Do not allow the form to maximize or resize.
- Provide icon for the form.

Grading Guidelines: Your project will be graded on the following:

- Correct solution to the proposed problem.
- GUI design (as outlined above).
- Elegance, simplicity, style, and readability of the algorithms and code.
- Grading rubric (given below).
- Comments:
    - ✓ Use of Javadocs.
    - ✓ Description of project.
    - ✓ Description of methods.
    - ✓ Description of complicated code.

Hints for Project #1: *Given that your GUI resembles the given one, follow these guidelines:*

- ✓ Consider the steps for this project:
    - Understand the problem.
    - Draw a flowchart or pseudocode and have an example completely worked out on a piece of paper following your flowchart.
    - Design the GUI in NetBeans—use JTabbedPane and JPanels to group controls with similar functionality.
    - Write the code following the IPO (Input, Process, and Output) protocol.
    - Make sure you have a declaration section for the constants and variables, an input section, a calculation section and an output section.
    - Perform calculations on "paper" with known results against which you calculations with the program should be checked.
- ✓ You may use your own images or free images from the Internet but follow these restrictions:
    - The images should be small, included in an Images folder in the src subfolder.

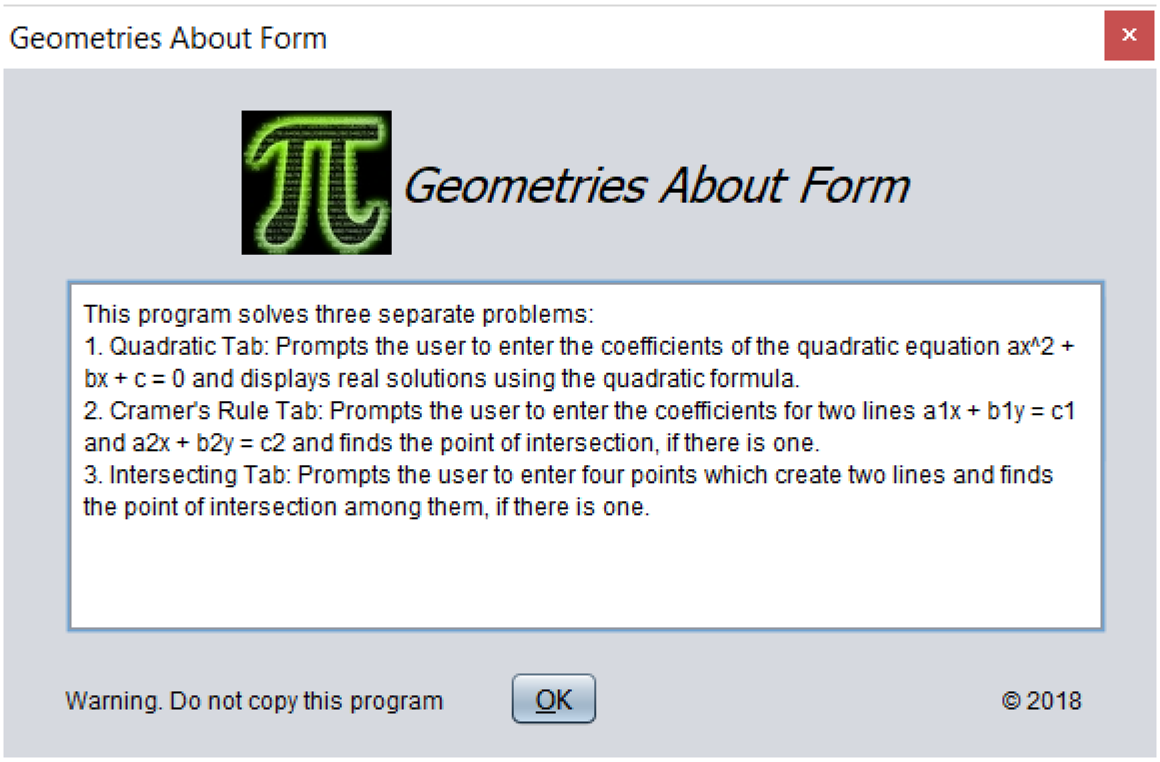
   - The images should be free—it takes a simple request with an email for permission to use if obtained through the Internet. Consider including a comment of where you obtained the images.
   - The images should be only in jpeg, gif or png format.
- ✓ The current date should appear as the form loads in the form's title.
- ✓ Make certain the calculated values are not editable.
- ✓ Make sure the Calculate JButton is default (it "fires" on Enter) for each JPanel.
- ✓ Comments, comments, comments all over. Make sure you use Javadocs guidelines and you generate the Javadoc html files. Read the document on Javadocs and follow the Java coding convention.
- ✓ Chose variables of appropriate type and name them using the established Java naming convention. Do the same with controls and constants. Read the document on Java naming convention and follows its guidelines.
- ✓ Indent code and make it easy to ready.
- ✓ Make sure you use constant declarations (final) for all constants.
- ✓ Provide meaningful menus that synchronize with the buttons but do not duplicate code.
- ✓ Provide ToolTipText with appropriate description for each control that needs it.
- ✓ The output JTextAreas should not be editable—the user should not be allowed to change the calculated values.
- ✓ Consider using the DecimalFormat or (better) NumberFormat class for formatting the output to desirable (four) number of significant digits.

Pay attention to the following criteria for grading:

'~~~~~~

'Good effort. Here are suggestions for improving:
'1. Use Javadoc comments through the program, not just at the top heading.
'2. Convert entries in text fields from strings to numbers. You get type mismatch run-time error otherwise.
'3. Make the Show button default for each tab—pressing the Enter should fire it.
'4. Correct the calculations—results are incorrect.
'5. Follow Java's naming convention for class, methods, variables and constants.
'6. Use final definitions for constants and use the Math class methods.
'8. It is good style to tab code inside each method.
'9. Use JTabbedPane with three tabs (JPanels).
'10. Use System.exit(0); to exit program.
'11. Clear should clear all text fields, all text areas, and class variables and reset focus to first tab.
'12. Change the tab index with menu choices as well.
'13. Name the project and classes appropriately.
'14. All input and output values should be double type.
'15. Clear should clear all text fields and reset the form to its original state.
'16. Provide current date in the form's title.
'17. Provide icon for the form.

'18. Validate inputs for correct type and range—use try/catch blocks.
'19. Avoid excessive number of class variables-declare variables inside methods.
'20. Clean up the empty procedures code.
'21. Show the form centered as it starts.
'22. Add menus to the form with meaningful choices.
'23. Synchronize menus with tab selections and controls.
'24. Set Editable property to False for all output text areas.
'25. Disable form from maximization or resizing.
'26. Provide ToolTip with meaningful message for the needed controls.
'27. Display results with four decimal places--use the DecimalFormat or NumberFormat class.
'28. Distinguish between inconsistent and dependent cases in Cramer's Rule.
'29. Missing required calculations.
'30. Missing meaningful menus without duplicating code.
'31. Provide ability to print invoice—use provided PrintUtilities class.
'32. Provide a meaningful and detailed About form and a Splash screen.
'33. Program crashes on no input or illegal input.
'34. Incomplete.
'35. Late.
'
'The ones that apply to your project are:
'1, 3, 6, 15, 28
'
'25+1=26/30
'~~~~~~



This program solves three separate problems:
1. Quadratic Tab: Prompts the user to enter the coefficients of the quadratic equation ax^2 + bx + c = 0 and displays real solutions using the quadratic formula.
2. Cramer's Rule Tab: Prompts the user to enter the coefficients for two lines a1x + b1y = c1 and a2x + b2y = c2 and finds the point of intersection, if there is one.
3. Intersecting Tab: Prompts the user to enter four points which create two lines and finds the point of intersection among them, if there is one.

Warning. Do not copy this program

© 2018