Project 5: Fractals & GUIs (Triangle-ation!)

Contents

1 Understand

1.1 Know the Objectives

After completing this project, you should have a solid grasp of these topics and how to code them:

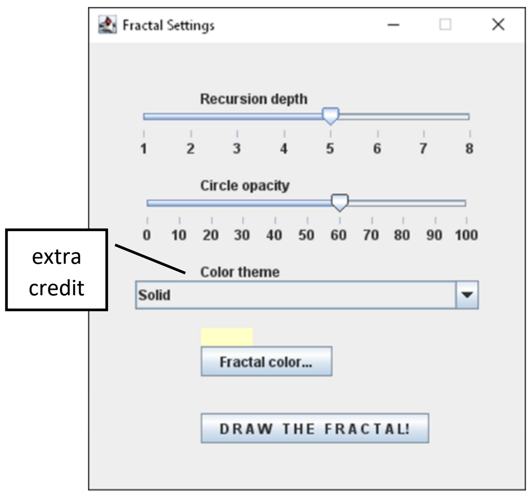
- Fractals (created using recursion)
- Observer design pattern

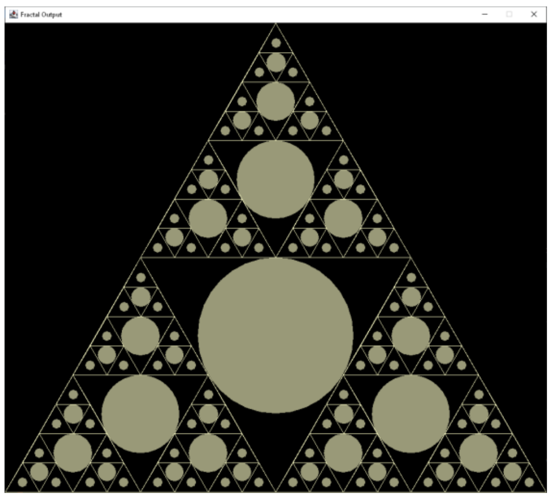
1.2 Understand the Problem: Fractal

Fractals are often drawn using recursion, starting with one drawn pattern, then branching off to child patterns (usually smaller and offset); this continues until the desired recursion depth has been reached.

Our fractals will start with a large equilateral triangle with a circle at the circumcenter of a triangle formed from the original triangle's side midpoints. As recursion progresses, drawing proceeds inward, with children created in each of the three corners of the parent.

The GUI lets the user control many aspects of fractal drawing. The expected GUI[1] with defaults[2] is shown below, at left, with the corresponding output shown at right. Output must be drawn on a *single* separate window (i.e., not a new window each time the fractal is drawn), with a black background. When the GUI appears, it should have default set, with the default image shown on the drawing area[3]. Note that the drawing area is not square; it needs to house an equilateral triangle. When the user clicks the bottom button, the corresponding drawing should be shown.





2 Design

2.1 Design Documentation

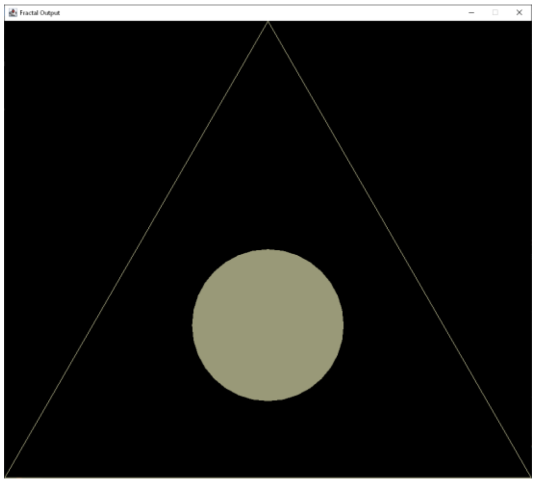
We will work through this project's design together; as such, *no preliminary designs are due*. Please prepare to actively participate in the design discussion. We will also look at a UML Sequence Diagram that will be useful in understanding the chain of events that result in the updated drawing.

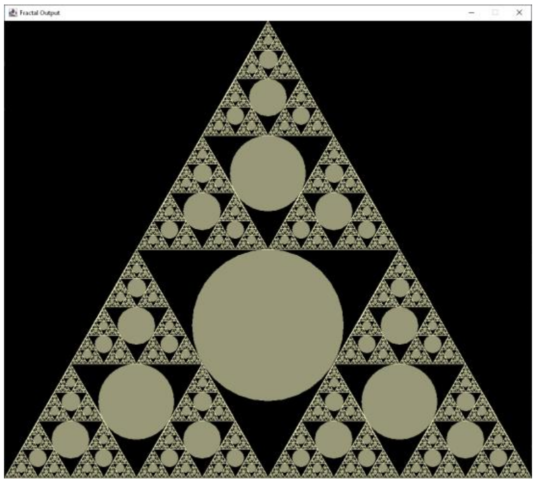
[1] Your GUI should look very similar to the one shown; carefully note the widgets, labels, layout, and spacing.
[2] Your defaults should be exactly the ones shown in the GUI, except for the default color; there, choose one that suits you and look good on a black background.
[3] Don't show an empty black window, please; that's not visually enticing to the user.

2.2 GUI Settings

2.2.1 Recursion Depth

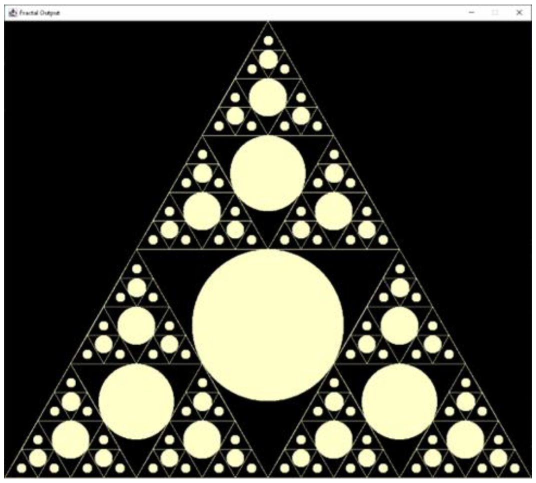
This slider controls how many levels of recursion are performed. Below left is a sample of output with a recursion depth of 1 (minimum); at right, output with a recursion depth of 8 (maximum).

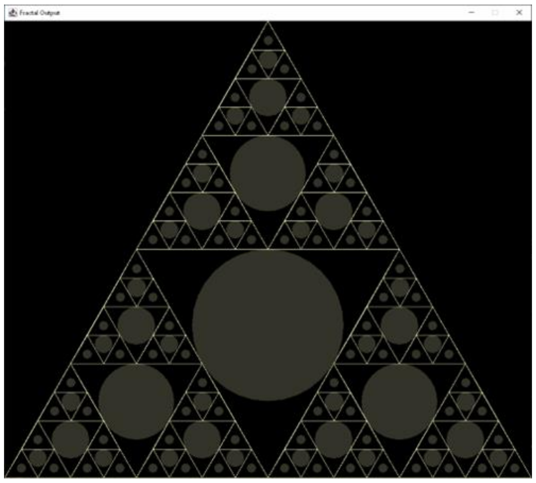




2.2.2 Circle Opacity

Circles are drawn at the circumcenter[4] of a triangle created from the midpoints of the parent triangle's sides. Circles are transparent, with their transparency being set by the opacity slider in the GUI. Research the setColor() overloads that take four parameters (red, green, blue, alpha). The sample image at left shows output with opacity set to 100 (alpha = 1.0); the image at right, 20 (alpha = 0.2).





3 Code

3.1 Understand the Details

3.1.1 Core Classes

- FractalGui is responsible for *gathering user data* via the settings dialog.
- FractalGenerator is responsible for *generating fractal data* and storing it in an ArrayList.

[4] You'll find background on finding the circumcenter in this video, and sample code on this page.

- FractalDrawing is responsible for *drawing the fractal* using the fractal data in an ArrayList. This must be a separate class from the GUI. But to make this more universal, have the fractal elements *draw themselves* when asked[5].
- Main: since this is a full-featured application, you will need a Main class separate from the others. But main here has little code in it; it should consist of only three code statements.

3.1.2 Observer

We will use the Observer design pattern; the FractalGenerator will be the concrete Subject; the FractalDrawing, the concrete Observer. The FractalGui won't participate in an observer/subject model; it will simply send data to its FractalSubject (via setData) when the user clicks the Draw button.

Remember that in Observer, we can implement incredibly *loose coupling*; none of the three classes mentioned above should hardcode the class names of the other two. Only main needs to know and use these, to do the introductions/handshakes.

We will use a lazy pull model. "Pull" means that the Observer update message won't be laden with data; it is just an announcement that data is available. The concrete observer will then ask for data (via getData), which will retrieve the ArrayList of FractalElement objects to draw. "Lazy" means that you shouldn't generate data unless it is requested; only when getData is called should you generate it.

3.1.3 FractalElement Interface

Your fractal generator will be creating shape objects. To support and group those, we'll create a FractalElement interface that has one method: draw. This method will need a Graphics reference parameter. Implementing classes are responsible for drawing themselves and should require minimal calculations as the most interesting calculation work is done in the fractal generator class.

3.1.4 Triangle Record

This class will implement the FractalElement interface. Triangles can be drawn as three lines (drawLine), or perhaps as an unfilled polygon (drawPolygon).

3.1.5 Circle Record

This class will implement the FractalElement interface. Circles are drawn with fillOval().

3.1.6 Use These

- GUI dialog created in Java Swing
- Graphics drawn on a JPanel (for the display)
- Observer design pattern
- Recursion

3.1.7 Don't Use These

- A monolithic approach where everything is in one class (or only a couple of classes).
- The DrawingPanel class provided by the textbook authors.

[5] You would not want the whole application to work only with specific shapes; it should be flexible enough to work with other shapes, though you are not required to do that work.

3.2 Use the Interface(s)

Use the provided interface(s) for the project. Your class(es) should implement these and must exactly follow the signatures in the interfaces. Add helper methods to your classes, of course, but do not alter the interface in any way.

3.3 Implement Other Requirements: Hints

3.3.1 Remember the Basics

- Build code incrementally. Check your work at each step.
- Remember the standard mathematical practice of carrying around the maximum precision for as long as possible, then *round* at the last moment; that is good advice, here.

3.3.2 Drawing Circles

- Circles are drawn from the upper left-hand corner of their bounding box, not their centers. That is a detail that you can put off until the last minute. Instead, focus on the *center* in earlier thinking.

3.3.3 Overall Approach

It is tempting, and not unrealistic, to first build the program in a monolithic fashion. For example, you could write code that gathers setting information from the console/keyboard, generates the data, and draws the fractal on a DrawingPanel. But then this code will need to be split apart to fit into the model we will design; that is not a small feat, requiring some mental paradigm shifts. Another approach would be to start with the model and make sure your signaling mechanisms are working properly; then write code in stages and check carefully that the state is right at each step.

4 Document

4.1 Follow the Style Guide

Follow the Course Style Guide, which is linked in the Reference section of the Modules list in Canvas. Failure to do so will result in the loss of points.

4.2 Write JavaDoc

Write complete JavaDoc notation for all classes in this project. This means that an -Xdoclint:all comes back with no errors or warnings. If there are errors or warnings, you will lose points.

5 Test

~~5.1 Write Unit Tests~~

Take a break from testing on this project. It's prudent to test basic building blocks you create, though.

5.2 Verify Code Meets Acceptance Criteria

Work will be returned to you for rework (with a deduction for late work) if it doesn't meet the acceptance criteria listed here:

- …compiles with all required Xlint and Xdoclint command line additions.
- …runs without throwing any exceptions.
- …allows the user to choose fractal options.
- …draws the resulting fractal upon request.
- …has results that are at least a vague facsimile of what is expected.

6 Demonstrate

A working GUI with fractal output is the only demonstration necessary.

7 Earn Extra Credit

7.1 Real-time Interaction (1%)

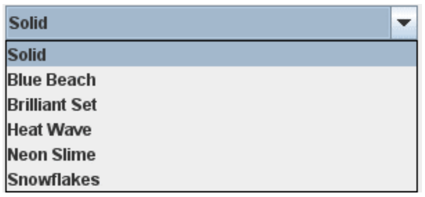
GUI changes should immediately be reflected in the drawing (i.e., it should implement real-time updates).

7.2 Theme Selector in GUI (1%)

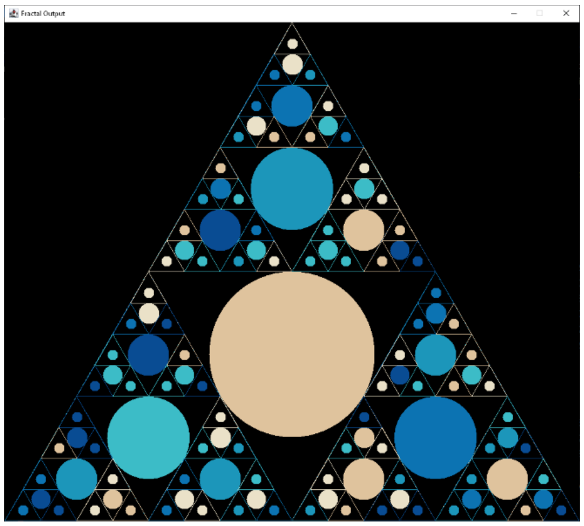
7.2.1 Themes

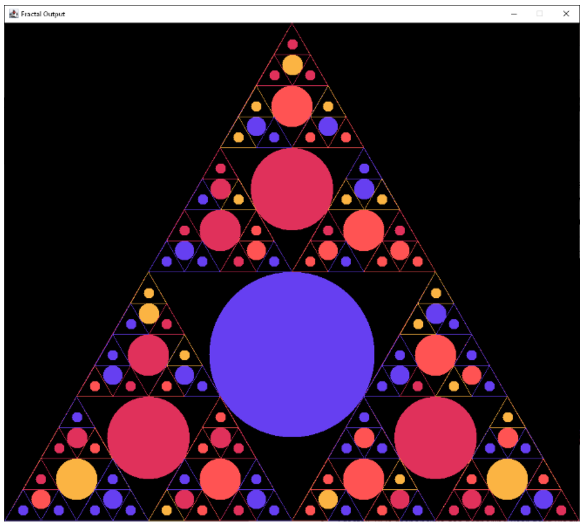
The default theme should be "Solid", allowing color choice via the Fractal Color button (which launches a Swing color picker). When this theme is not selected, the Fractal Color button should be disabled. Start with the themes shown below; add a few others you like.

Other themes are colorful ones selected from this site: https://www.schemecolor.com/. Each triangle and circle should be a random color from the theme. After "Solid", themes should be listed alphabetically.



Below left is sample output from Blue Beach; below right, Brilliant Set. Samples have opacity at 100% to clearly show theme colors.





All theme work should be encapsulated in the GUI; send an array of colors when sending data to the Subject. That works for Solid, too; the array can have one element in that case.

7.3 Themes Stored in File (1%)

- Store the theme color names and color data in a file.
- Mark the data structure(s) that hold these *static*.
- Write *static* code to read the file and fill in the data structures at class load time.

8 Measure Success

Area	**Value**	**Evaluation**
Observer/Subject implementation	15%	Was the Observer model properly and successfully used? Were all the handshakes properly coded and used? Was loose coupling implemented as we designed?
GUI layout	10%	Does the GUI look like the example? Do widgets send correct data?
GUI class implementation	15%	Was the class set up as requested? Was it successfully coded using good practices?
Fractal drawn/colored properly	15%	Was the fractal drawn correctly, given the settings? Did subsequent drawings look good? Were screen sizes appropriate and friendly?
Fractal generation class	20%	Was the class set up as requested? Was it successfully coded using good practices?
Drawing class implementation	15%	Was the class set up as requested? Was it successfully coded using good practices?
JavaDoc	5%	Did you use JavaDoc notation where requested, and use it properly? Did the Xdoclint run come back clean?
Style/internal documentation	5%	Were other elements of style (including the Style Guide) followed?
Extra credit	3%	Was the extra credit implemented per instructions?
Total	**103%**	

9 Submit Work

Follow the course Submission Guide when submitting your work.