Homework 08 – Fractal Museum

Problem Description:

We're getting to the end of the semester! For this last homework, you will explore your creative side with JavaFX. Please make sure to read all parts of this document carefully!

You are a contemporary modern artistic gamemaster and your new favorite tool is JavaFX. You must invent a game in which Two Players, just by gazing at a Fractal Creation, attempt to guess how many distinct parts it contains. In this assignment you will be using your knowledge of:

- GUIs
- Event-Driven Programming
- Recursion.

There are going to be some basic requirements, but the UI and design of the game is largely up to you! Feel free to explore some fun aspects of JavaFX! This homework is intentionally more open-ended, and we grade most of it manually.

Remember to test for Checkstyle errors and include Javadoc comments!

JavaFX Installation Directions:

JavaFX installation instructions can be found on Canvas.

Solution Description:

Requirements

Write a JavaFX Class called `FractalMuseum`. The class must meet the following requirements:

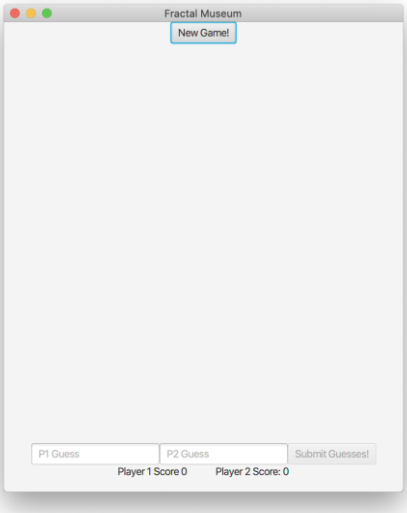
- The title of the window must be "Fractal Museum." The window must not cut off anything.
- Make sure to use at least one Anonymous Inner Class and one Lambda Expression in your implementation. (This is required for full points.)
- A button to Generate the Fractal
  - Clicking the Fractal Generation Button should re-enable the Guess Submit button
  - Clicking the Fractal Generation Button should generate a Fractal
- An area to display the Fractal Creation. We have provided a class, `FractalFactory`, with a static helper, `drawFractal(...)`, which draws the Fractal inside a `Pane` you provide:
  - A fractal is a never-ending pattern, but for this homework the pattern will end when there is no more resolution (i.e. <1 pixel in size). The generated Fractal could be up to 500x500 pixels in size.
  - The fractal will be created by recursively creating squares that are a fraction of the size of the previous square(s) and positioned at the corners of the previous square(s)
  - `drawFractal(...)` returns an integer representing the number of distinct squares drawn
- An area for Players to guess how many distinct squares are present in a Fractal Creation.
  - Two user input fields, which validate their input type appropriately
    - Players cannot submit empty guesses
    - Players can only guess non-negative integers
  - A button which Submits the guesses of both Players
- An area to display each Player's number of wins
- When the submit button is clicked:

- The input fields should clear and the Submit button should become disabled
- The winner must be indicated. Whichever player guessed the closest number of squares wins. Make sure to not arbitrarily select a winner if there is a tie; neither player's score should update.
- The correct answer should be displayed.

Screenshots

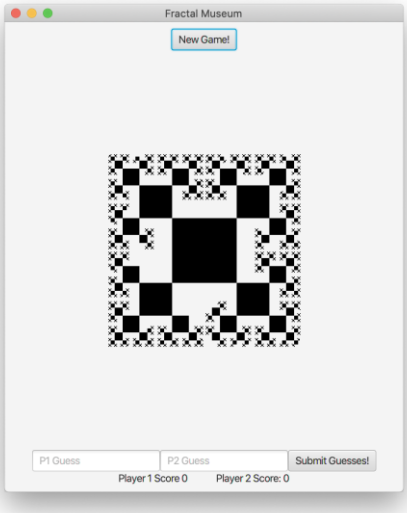
Here are images of a `FractalMuseum` which correctly meets the requirements above. You may style your game just like this, or differently, but make sure to fulfil the requirements above.

1. Startup



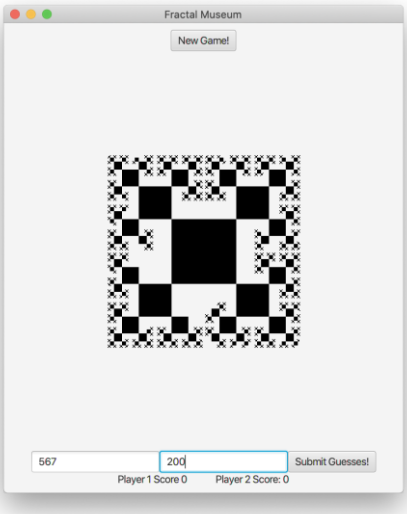
Launching the game presents this view.

2. New Game



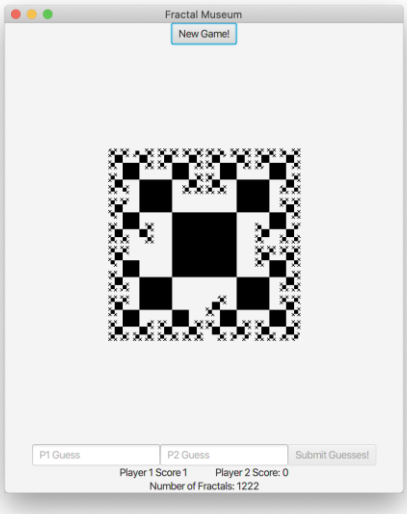
Generating a new Fractal presents this view.

3. Guess Input



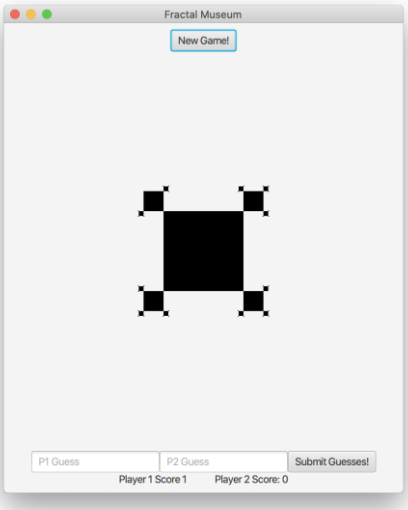
Both Players have entered their guesses, but have not yet submitted them.

4. Guesses Submitted



The Players have submitted their guesses. The Submit button is now disabled, the answer displayed, and score updated.

5. New Game



Clicking on New Game presents a new Fractal and allows players to guess once more.

Tips and Tricks

- The Java API is your friend. Use it to your advantage.
- Make sure you are properly able to run JavaFX programs before starting this assignment. DO NOT wait until the last moment to configure your JavaFX!
- Work incrementally. Get a basic UI up and running. Add buttons, cells for display, etc., piece by piece. Think of which type of layout manager(s) you want to use

Extra Credit Options:

As this homework is relatively open-ended, there will be up to **30 bonus points available** for expressing creativity through your implementation. **In order to be eligible for extra credit**, you must first earn the first 100 points by meeting all of the requirements.

All of these items are subjectively graded. Additions will be considered under the following categories. You may earn points under a certain category multiple times:

- 5 – Non-trivial enhancement to the graphics of the program
- 5 to 15 – Non-trivial enhancement to the controls of the program
- 5 to 15 – Non-trivial new feature added

Some examples include, but are certainly not limited to:

- Custom Profile (Photos, names, lifetime wins, etc.) for the Players
- An option to select from one of multiple Fractal shapes (including a square, as required)
- Different parts of the Fractal have different colors
- Ability to zoom in to the Fractal Creation
- Add more Players
- A Configuration file which is saves data between game runs

Rubric:

- [100] FractalMuseum
    - [10] Anonymous inner class
    - [10] Lambda function
    - [5] window title
    - [20] Logical Layout & Function
    - [25] Guess Entry Fields & Submission
    - [10] Fractal Generation Button
    - [10] Winner Selection
    - [10] Score Tracking

Allowed Imports

There are no disallowed imports for this assignment. *Feel the full power* of the Java API. (It's over 9000)

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `assert` (the reserved keyword)

Javadoc

For this assignment, you will be commenting your code with Javadoc. Javadoc comments are a clean and useful way to document your code's functionality. For more information on what Javadoc comments are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the Javadoc overview for your code using the command below, which will put all the files into a folder named "javadoc". Note you should execute this after adding Javadoc comments to your code:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 *This class represents a Dog object.
 *@author George P.Burdell
 *@version 1.0
 */
 public class Dog {

    /**
     *Creates an awesome dog(NOT a dawg!)
     */
     public Dog() {
     ...
     }
```

```
    /**
     *This method takes in two ints and returns their sum
     *@param a first number
     *@param b second number
     *@return sum of a and b
     */
     public int add(int a,int b) {
     ...
     }
}
```

A more thorough tutorial for Javadoc Comments can be found here.

Take note of a few things:

1. Javadoc comments begin with `/**` and end with `*/`.
2. Every class you write must be Javadoc'd and the @author and @verion tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an @param tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadoc comments using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here.

To run Checkstyle, put the jar file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **35 points**. This means that up to 35 points can be lost from Checkstyle errors.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one .java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload all source files you created to Gradescope. You will also need to include the `FractalFactory` source code whether or not you've chosen to modify it.

Make sure you see the message stating "HW08 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

Due to the open-ended nature of this homework, Gradescope will only check for compilation, Checkstyle, and a correct collaboration statement.

The remainder of your assignment will be graded by a TA once the submission deadline has passed, so the output on Gradescope will not reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications