CS 235 - Homework 7

Deadline:

11:59 pm on **Friday, November 5, 2021.**

Purpose

To provide practice with creating and controlling Java threads, and to practice with simple graphics (painting/drawing) in Swing.

How to submit:

You complete **Problems 1 and 2** on the course Canvas site (short-answer questions related to Java threads, and to painting on a Swing container).

For **Problems 3 onward**, you will create the specified `.java` files, and then submit those to the course Canvas site. (You'll be creating `.class` files, also, but you do not submit those.)

Important notes:

- Follow the Java coding standards mentioned in previous homework handouts and discussed in class.
- Be sure to submit copies of all `.jpg`, `.gif`, or `.png` files used in your homework problems.
- Note that Java applications with graphical user interfaces are expected to be structured as demonstrated in the in-class example `ButtonTest.java`
  - (that is, with an application class that creates and displays a `JFrame` subclass instance within the event dispatch thread,
  - and a `JFrame` subclass that creates and adds a `JPanel` subclass instance to itself in its constructor,
  - and a `JPanel` subclass whose constructor creates and adds appropriate components to itself)
- Using appropriate additional helper methods is fine, as is using additional classes. IF you use additional public classes, be sure to submit those as well!
- Because graphical user interfaces are involved in some of this homework's problems, **the CS50 IDE will NOT work for all of this homework's problems**. (Running in a browser, on the cloud, it cannot access your screen to display a `JFrame`.)

  If you have a Terminal or bash shell, you can compile and run Java as you do from the CS50 IDE Terminal.

  AND -- I have verified that Java works -- compiles and runs -- from the **Command Prompt** on `vlab.humboldt.edu` as it does from the CS50 IDE, also.

  That is:

  - Log into `vlab.humboldt.edu`

- In the search bar on the lower left, search for "command prompt", and click on the "Command Prompt" app that comes up.
- Even though this is a Windows Command Prompt window and not a bash shell, commands such as `mkdir` and `cd` and `ls` work here.
- I found that if I saved a `.java` file on the vlab desktop, then from the Command Prompt I could do the following:

  ```
  C:\Users\st10> cd Desktop
  C:\Users\st10\Desktop> javac MyGuiApp.java
  C:\Users\st10\Desktop> java MyGuiApp
  ```

  ...and my application would compile and run!
- (But save your `.java` files to your Google Drive for safer, longer-term storage that can be more easily accessed than the vlab Desktop!)

Problem 1 - 12 points

Problem 1 is correctly answering the "HW 7 - Problem 1 - short-answer questions thread basics" on the course Canvas site.

Problem 2 - 10 points

Problem 2 is correctly answering the "HW 7 - Problem 2 - short-answer questions on basics of painting on a Swing container" on the course Canvas site.

Problem 3

Copy `GameDie.java` into your directory where you are putting this problem's Java files. (This can be the Week 1 Lab's version of `GameDie`, or your Week 4 Lab Exercise version of `GameDie`.)

As a warmup, and to practice a bit more with threads, create a small Java application `RollThreads.java` that expects (at least) one command line argument, expected to be a positive integer, and it tries to create that many threads. (It should complain and exit if not given a positive integer command-line argument.)

And, it should somehow give the threads it creates at least some time (let's say at least 20 seconds) to run before it tries to interrupt them. (You can have the `main` method do more, also, if you'd like, but make sure:

- it gives the threads at least 20 seconds to run, and
- it does attempt to interrupt each thread it creates.)

What should your threads do? Each should at least meet the following requirements:

- Each thread's output should somehow include the name of the thread creating that output (so we can see that there really *are* multiple threads running at the same time.)
- Each thread should be set up such to catch an `InterruptedException` and exit gracefully -- also include some output when it exits so we can see it was interrupted and ended.

- Each thread should create a game die instance (this can be the same number of sides every time, or the game die's number of sides can be determined by a second command-line argument, or the game die's number of sides can be interactively requested and entered by the user for each thread, your choice!)
- Then each thread should do something with its game die instance -- for example:
  - each thread could loop, rolling its game die and printing its results to the screen and then sleeping; (but still include the thread's name in what it prints, so we can see that multiple threads are really running)
  - each thread could put up a `JOptionPane` (somehow including its name) when it starts, and then, perhaps, roll a die, sleep, roll a die, sleep, etc., until it is interrupted, and then it could put up a `JOptionPane` (again somehow including its name) summarizing, say, how many times that thread's die was rolled (and more or different stats if you are interested -- the average dice roll, number of times a 3 was rolled, etc.)
  - each thread could "silently" roll its die, sleep, roll, sleep, until it is asked to stop, and then it prints a final message including the thread's name and saying how many times it was rolled (and more if you wish -- average die roll? number of times a 6 was rolled? etc.)
  - or some other task you'd like to have each thread do with its game die.

Submit your resulting `RollThreads.java` and the `GameDie.java` that you used with it.

Problem 4

Note that the class `Graphics` is in the package `java.awt`, as is its more modern subclass, `Graphics2D`. You can read about the many methods they include in the Java 16 API.

Remember that you can draw/paint an image stored in a `.jpg`, `.png`, or `.gif` file as well, as you did in the Week 10 Lab Exercise.

To provide some additional light practice with painting/drawing in Swing, modify the Week 10 Lecture example `DrawApp1.java` into `DrawPlay.java`, meeting the following requirements:

- add another `@author` line to its opening Javadoc comment, indicating that you have adapted this
- change the `@version` line to the date that you last modified this
- (you may change the frame's width and height as you would like, making sure everything you draw/paint below is still visible!)
- within `DrawPlayPanel`'s method `paintComponent`, remove everything AFTER the first statement (everything AFTER the call to `super.paintComponent(g)`), and replace it as follows:
  - using font and color of your choice, visibly draw/paint a string including your name on the panel
  - using a color/colors other than the default color, visibly draw/paint at least one shape or line, using at least one method from either class `Graphics` or class `Graphics2D`
  - obtain at least one reasonably small image of type `.jpg` or `.png` or `.gif`, and save it/them in the same directory as `DrawPlay.java` (and its resulting `.class` files). Then, visibly paint it/them on the panel, in location(s) of your choice.
  - you may paint/draw any additional items that you would like, as long as the above are also visibly

included.

Submit your resulting `DrawPlay.java`, AND ALSO all `.jpg` or `.png` or `.gif` files that it uses!

Problem 5

For a little more practice with threads and painting/drawing in Swing:

Create an application `AnimateTwo.java` that animates TWO "things". (Optionally, if you'd like to animate **more** than two "things", that is fine -- just change the name of this class accordingly!)

- These two "things" should be properly painted onto a sub-panel, and they should be visibly different, but otherwise you can paint images, strings, lines, rectangles, shapes, some combination of these, or you can go further afield if you would like.
- YOU CHOOSE if both will be controlled by a **single** begin and a single end button (both controlled by a single thread),

  OR if each is controlled by its **OWN** begin and end button (4 buttons total) (each controlled by its OWN individual thread).

  - tastefully place these two or four buttons on top, bottom, left, or right, your choice;
  - You are required to create your thread(s) appropriately from `Runnable` instances as has been discussed in class and in the course text.
  - You may NOT use deprecated methods to control your thread(s) – you are required to use the `interrupt()` method appropriately to cause each thread's `run()` method to terminate gracefully.
  - NEITHER "thing" can move horizontally left-to-right across the screen -- (they can move up or down, right-to-left instead of left-to-right, bouncing, diagonally, sinusoidally, etc. -- how they move is your choice, as long as neither is moving left-to-right horizontally.)
  - As long as neither is moving horizontally left-to-right across the screen, they can move similarly or very differently.
- IF you choose to control both "things" with a single pair of begin-end buttons, then clicking the begin button should start both in their movements, and clicking end should stop both of their movements.
- IF you choose to control each "thing" with its own pair of begin-end buttons, then clicking its begin button starts JUST that one "thing" in its movement, and clicking its end button ends JUST that one "thing"'s movement.
  - Note, then, that it should be possible for both items to be moving at the same time, or for either one to be moving while the other is still, or for both to be still, depending on whose buttons have been pushed.
  - As in `ThreadImage1.java`, pushing the Start button for a thread while that thread is running shouldn't have any effect; and, pushing the Stop button for a thread while that thread is not running shouldn't have any effect, either. But if there currently isn't a running thread for that item, then pushing its Start button should start it moving on the screen, and if there is currently a running thread for that item, then pushing its Stop button should stop its thread (and its moving on the screen).)

Submit your resulting `AnimateTwo.java`, and also any `.jpg/.png/.gif` files that it uses (if any).