CS 235 - Homework 4

Deadline:

11:59 pm on **Friday, October 1, 2021**.

Purpose

To practice with `JFrame`, `JPanel`, `JLabel`, `JButton`, `JOptionPanel`, and simple event-handling, in the context of simple Java applications with Swing-based graphical user interfaces.

How to submit:

Submit your `.java` files for this homework to the course Canvas site. (You'll be creating `.class` files, also, but you do not submit those.)

Important notes:

- 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)
- Note that you can change the text within a `JLabel` (and indeed, a number of Swing components) by using its **`setText`** method, which expects a `String` and doesn't return anything, but has the side-effect of changing the `JLabel`'s text to that `String`.
- Note that `JButton` and `JLabel` have a method **`setForeground`** that expects a `Color` and doesn't return anything, but has the side-effect of changing the color of the text in that component (considered to be that component's foreground).
  - They also have a method `setBackground` which, at least on Max OS X, seems to have no effect on those components' background color! (as it turns out additional steps are needed to make this visible)
  - (although we know from `ButtonTest.java` that `setBackground` does work as expected for a `JPanel` subclass).
- Because graphical user interfaces are involved, **the CS50 IDE will NOT work on 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!)

- Follow the class Java coding standards mentioned in class and demonstrated in posted in-class examples -- some of these include:
  - Follow the Java naming standards that have been discussed in class.
  - Attempt "javadoc-style" comments for **each** Java class and method, in the same style as you see in posted in-class examples.
  - Everything inside a set of { } must be indented by AT LEAST 3 SPACES -- and the beginnings of statements that are sequential should be indented the SAME NUMBER of spaces. (That is, sequential statements should line up.)
  - { and } should each go on their OWN line, with { lined up evenly with the preceding line, with the { }'s contents indented by at least 3 spaces, and with } lined up with the opening {. That is, handle the curly braces as you see in all posted class examples!
- ASK ME if any of these are unclear to you!

Problem 1

This problem does not involve event handling -- but it does give you a chance to make use of `JOptionPane`'s `static` method `showInputDialog` to do something with input from a user in a simple graphical application.

Consider the posted examples from Week 5.

- `SimpleLabelTest.java` sets up an application that makes use of a `JFrame` subclass, and also makes use of a `JPanel` subclass that is placed on a `JFrame` subclass instance.

- `SimpleOptionPaneTest.java` includes an example of using `JOptionPane`'s static method `showInputDialog`.

Write a Java application `DisplayMsg.java` that that uses `JFrame` and `JPanel` subclasses to display a `JLabel` containing your name, and a separate `JLabel` to display the result from prompting the user to enter a message using a `JOptionPane` input dialog.

(Where should you call `JOptionPane`'s `static` method `showInputDialog`? There are actually several reasonable possibilities, I think, depending on how you organize this.)

You may use whatever (visible) colors you wish, and whatever frame size you think is reasonable that allows at least all of your name and at least part of a reasonable-length user message to show.

All of the classes involved should be designed to have their source code in the single file `DisplayMsg.java`.

Submit your resulting `DisplayMsg.java`.

Problem 2

Now for some event-handling!

Consider `CountClicksTest.java`, posted along with this homework handout. Read over this, try to determine what this does, and then compile and run it and see it in action.

Make your own copy of this class, and make the following changes/additions:

- Add another `@author` line in its opening javadoc comment, noting that this class has been adapted by you.
- Change the `@version` line's date appropriately
- Add a `JLabel` containing your name in text (foreground) that is some visible color other than blue or black **directly underneath** the `Button-Click Counter` label.
  - (You'll have to size the frame appropriately to achieve this using the default `FlowLayout` layout manager.)
  - (Yes, we *will* have a better way to lay this out when we discuss Java layout manager classes!)
- add an additional `JButton` labeled `Clear` **directly underneath** the `#-of-clicks` label which, when pushed, resets the appropriate label (and counter) so that `# of clicks: 0` is displayed (and starts recounting from 0 at the next "`Click Me`" button click).
- add a separate class implementing `ActionListener` to accomplish the new Clear button's desired actions.
- (be sure to resize the frame as needed so that your new components are nicely visible, and oriented as specified above)

Submit your resulting `CountClicksTest.java`.

Problem 3

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

Write a Java application `DieRoller.java` that makes use of `JFrame` and `JPanel` subclasses along with a class implementing an `ActionListener` to roll a `GameDie` instance and report the result.

(You will declare and make use of an instance of `GameDie` within this, but you should **not** include `GameDie`'s code within `DieRoller.java` – having `GameDie.java` in the same directory as `DieRoller.java` should suffice for it to be able to make use of that class.)

Your solution must also meet the following additional specifications:

- you get to decide how the number of sides of this game die is to be determined -- you can make it a named constant (it can always be the same), or you can allow the user to somehow choose this, your choice!
- it should include a `JLabel` containing your name (and whatever other text you'd like to include to result in an, ah, convenient length with regard to the default `FlowLayout` layout manager...)
- that name label should have, directly beneath it, a `JButton` instance, labeled `roll die` (or some descriptive button text that "fits"), and
- that button should have directly beneath them a `JLabel` instance initially showing `roll: 1` (or something similar that "fits"). (Note that `GameDie` does initially set the top of a die to 1.)
- (you may use any colors and fonts for the above that are reasonably legible/readable)
- When the button is pushed, your application should:
  - roll the corresponding `GameDie` instance. (Note that you are required to appropriately use the `GameDie` class in your solution.)
  - the value for that roll should be appropriately displayed in the button's roll-label.
- Use a class implementing `ActionListener` to accomplish the button's action.

Submit your resulting `DieRoller.java`.