

CS 235 - Homework 4

Deadline:

Due by **11:59 pm on Wednesday, September 23, 2015.**

How to submit:

Submit your files using `~st10/235submit` on nrs-projects, with a homework number of 4, by the deadline shown above.

Reminder: Instructions for using the tool `~st10/235submit`:

- If you did not create your files (using, for example, `nano`) on nrs-projects, then first transfer your files to be submitted to a directory on nrs-projects.
 - On Linux or Mac OS X, you can use `sftp` (secure file transfer) to connect to `nrs-projects.humboldt.edu`, and then transfer them
 - In a Windows lab on campus, you can use the **WinSCP** program, which is a graphical `sftp` program.
 - The free program **FileZilla** is a graphical `sftp` program that works on Windows and Mac OS X (and maybe also Linux?)

- Once your files to be submitted are in a directory on nrs-projects, then use `ssh` (or PuTTY in a Windows lab on campus) to connect to `nrs-projects.humboldt.edu`.

- Use `cd` to change to the directory containing the files to be submitted -- for example,

```
cd 235hw04
```

- Type the command:

```
~st10/235submit
```

...and give the number of the homework being submitted (or whatever number you have been asked to do for lab-related files) when asked, and answer that, `y`, you do want to submit all of the files with of-interest-to-235 suffixes in the current directory. (Note that I don't mind if a few extraneous files get submitted as well -- I'd rather receive too many files than too few, and typing in all of the file names for some assignment is just too error-prone...)

- You are expected to carefully check the list of files that the tool believes have been submitted, and make sure all of the files you hoped to submit were indeed submitted! (The most common error is to try to run `~st10/235submit` while in a different directory than where your files are...)

Purpose

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

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).
- 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 is a bit of a warm-up, to get you started.

You will find the source code file `ButtonTest.java` from the Week 4 Lab on the CS 235 public course web site. 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` to this containing your name (this can be before, after, or among the buttons, whichever you prefer)
- Add an active `JButton`, including appropriate additional code as needed, for a fourth color of your choice, different from yellow, blue, or red; clicking this should cause the panel's background to change to this fourth color.
- Change the fonts from what they are in the posted version for all 4 buttons and for the new label.

Submit your resulting `ButtonTest.java`.

Problem 2

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 4. `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.

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 3

Now, back to 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.)

- 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 4

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.

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`.