

CS 235 - Homework 5

Deadline:

Due by **11:59 pm on Wednesday, September 30, 2015.**

How to submit:

Submit your files using `~st10/235submit` on nrs-projects, with a homework number of 5, by the deadline shown above.

Purpose

To practice exception-handling, and writing Java GUI applications that happen to include `JTextField` components as well as some borders and numeric formatting,.

Important notes:

- It is possible that some of your programs may be posted to the course Moodle site.
- 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)
- 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

Before we get GUI, this is a small command-line application to practice with exception-handling.

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 command-line Java application `ManyRolls` which expects to be called with exactly two command-line arguments: how many sides the die to be rolled should have, and how many times that die is to be rolled.

It should then create a game die instance with that many sides and roll it the specified number of times, outputting the result of each roll to the screen in a readable fashion of your choice.

- What if the user calls it without exactly 2 command line arguments?
 - It should complain to system output with an appropriate complaining message of your choice, and exit.
- What if either or both of the two command-line arguments are not positive integers?
 - It should also complain to system output with an appropriate complaining message of your choice, and exit.
 - (Hint: note that you are concerned about two issues here: is each an integer? AND is each positive? How can you know if a command-line argument is not an integer? Once you know if it is an integer, then you can check if it is positive...)

For example:

```
java ManyRolls 6 5
```

...should print to the screen something like:

```
Rolling a 6-sided die 5 times:
roll 1: 3
roll 2: 2
roll 3: 1
roll 4: 6
roll 5: 2
```

As another example,

```
java ManyRolls 20 3
```

...should print to the screen something like:

```
Rolling a 20-sided die 3 times:
roll 1: 13
roll 2: 2
roll 3: 7
```

But,

```
java ManyRolls
```

...should print to the screen something like:

Must call `ManyRolls` with exactly 2 integer arguments -- goodbye!

And, calls such as:

```
java ManyRolls 3 -7
```

...should print to the screen something like:

Must call `ManyRolls` with 2 positive integer arguments -- goodbye!

...as should calls such as:

```
java ManyRolls -4 7
```

```
java ManyRolls -8 -2
```

```
java ManyRolls 2.0 5
```

```
java ManyRolls moo oink
```

Submit your resulting `ManyRolls.java`.

Problem 2

Now, because I think few pairs had time to complete Problem 2 of the lab exercise, now you get a chance to do so and add a few additional enhancements.

As noted in the Week 5 Lab Exercise, there is a Java GUI application named `Mult2.java` available on the public course web page, under "In-class Examples" (and now also along with this homework handout). Read over it; most of it should look familiar after the Week 5 lecture.

Decide on at least one additional operation besides multiplication that you'd like your application to compute. (Addition is fine, as is subtraction or division or average or any operation of interest that can be done to two double values.)

For this homework problem's version, copy `Mult2.java` into a file `ComputeMore.java`, and change the class names accordingly. Then:

- ADD an `@author` line with "adapted by <your name>" to its javadoc comment
- Change the "Mult2" label to say "ComputeMore, adapted by <your name here>"
- Change the instructions label's text appropriately to "fit" the new functionality
- Add an additional button for each additional operation you decide to add
 - (you may change the label on the "Multiply Values" button, as long as its meaning is still clear, especially if that helps with layout or just allows it to look better with your new button(s))
- Right now, there's a label next to the textfield holding the multiplied result that says "And the product is:". That isn't going to work with two or more operations -- modify the logic so that the text for this label changes to describe the operation that was done.
 - (e.g., when the multiply button is clicked, it says "And the product is:" -- and if your other button was, say, an add button, when clicked that label would change to "And the sum is:")
- Modify it so that at least two components use a font or fonts visibly and obviously different than the default font.

- Make all the textfields right-aligned.
- Modify it so that, if the user enters a value that cannot be parsed as a `double`, a `JOptionPane` is displayed letting the user know that a number must be entered, and the textfields are set to contain 0.
- Modify it so that the computed result displayed appears in a non-editable textfield.
- Modify it so that the computed result displayed is formatted to precisely 3 fractional places.
- Give at least one component a noticeable, visible border of your choice.
- Resize, kluge as needed to get a reasonable/readable looking result. Colors and fonts are your choice, as long as the result is readable.

Submit your resulting `ComputeMore.java`.

Problem 3

Consider: the class `Color` in package `java.awt` includes a constructor that expects three integers, representing a red value, a green value, and a blue value, each an integer in $[0, 255]$, and produces a new `Color` instance with that RGB value.

For example, you can try out the following in DrJava's Interactions window:

```
import java.awt.*;
import javax.swing.*;
Color custom = new Color(13, 188, 233);

JLabel playLabel = new JLabel("LOOK AT MEEEE");
playLabel.setForeground(custom);

JPanel playPanel = new JPanel();
playPanel.add(playLabel);

JFrame playFrame = new JFrame();
playFrame.add(playPanel);
playFrame.setSize(200, 50);
playFrame.setVisible(true);
```

But, that's a clunky way to play with and see the colors you'd get with different red, green, and blue values -- here, you'll write a tool that makes color experiments more convenient.

Write a Java GUI application `ColorPlay1.java`, that allows one to enter different combinations of red, green, and blue values, and see the resulting color. These are the minimum requirements:

- Somewhere within it should be a `JLabel` including your name.
- Give at least one component a noticeable, visible border of your choice.
- It should include 3 `JTextField` instances whose contents will be right-justified, with accompanying `JLabel` instance(s) requesting that the user enter desired red, green, and blue values into these textfields. (You get to choose the relative orientation of these textfields and label(s).)
 - You may decide what the initial contents of these textfields should be.

- It should include a `JButton` with appropriate text on it, which, when pressed, will cause the color of the application panel's background to be changed to the color corresponding to the red, green, and blue values currently entered in the 3 textfields.
 - What if the user enters something that is not an integer in one or more of the textfields?
Then an appropriate `JOptionPane` dialog should appear letting the user know they've entered at least one "bad" non-integer value, and the textfields' contents should be reset to some "safe" default value of your choice.
 - What if the user enters all integers, BUT enters an integer not in $[0, 255]$ in one or more of the textfields?
Then another appropriate `JOptionPane` dialog should appear letting the user know they've entered a "bad" out-of-range integer value, and the textfields' contents should be reset to some "safe" default value of your choice.
 - (notice that, in either "bad" case above, you are changing all 3 textfields' contents, which is admittedly simpler. OPTIONALLY, you may choose to JUST change each "bad" textfield(s)'s contents.)
- Resize, kluge as needed to get a reasonable/readable looking result. Fonts, and other colors besides the panel's background color, are your choice, as long as the result is readable.

Submit your resulting `ColorPlay1.java`.