CS 235 - Homework 5

Deadline:

11:59 pm on **Friday, October 8, 2021**.

Purpose

To practice more exception-handling (both not involving, and involving, `JTextField` objects!), and writing Java GUI applications that happen to include `JTextField` components as well as some borders and numeric formatting.

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)
- 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

Before we get GUI, this is a small command-line application to practice some more with exception-handling (and, *if you choose*, a bit more with numeric formatting).

Copy `GameDie.java` into your directory where you are putting this problem's Java files. (For our purposes here, either the posted Lab 1 version or your Week 4 Lab Exercise version should work.) You will be using a `GameDie` instance in this problem.

Write a command-line Java application `ManyRolls` that 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`.

Optional variation

Fun fact: for `System.out.printf` and `String.format`, "`%`*val*`d`" outputs an `int` value right-justified in a field of size *val* (although it ignores this field size if it has more than *val* digits.)

Use numeric formatting to "line up" your printed roll-results.

Optional variation 1: Do so assuming a reasonable maximum number of die-sides (such as 99) and a maximum number of requested-rolls (such as 99).

Optional variation 2: Do so *not* assuming such a maximum (and determining the possible maximum field-width dynamically).

For example:

```
java ManyRolls 20 10
```

...could print to the screen something like:

```
Rolling a 20-sided die 10 times:
roll  1:  1
roll  2:  7
roll  3: 14
roll  4: 19
roll  5:  1
roll  6: 13
roll  7:  8
roll  8: 20
roll  9:  8
roll 10: 13
```

Problem 2

Consider `Mult2.java` from the Week 6 Lab Exercise. What if it supported more than just multiplication?

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.)

Then, for this homework problem, copy your Week 6 Lab Exercise version of `Mult2.java` into a file `ComputeMore.java`, and change the class names accordingly. Then:

- ADD a *third* `@author` line with "adapted by <your name>" to its javadoc comment.
- Change the `@version` comment appropriately.
- Change the "Mult2" *frame title* to say "ComputeMore".
- Change the "Mult2" *label* to say "ComputeMore, adapted by <your name here>"
- Change the `JLabel` you added containing your and your Week 6 Lab Exercise partner's name(s) to contain just your name. (Note that your partner(s) will still be acknowledged in the 2nd `@author` comment for this class, which you are leaving in this version.)
- Change the instructions label's text appropriately to "fit" the new functionality you decided upon
- 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:")
- Change the size of the `ComputeMoreFrame` as desired (as long as all of the components are visible).
- As long as the results are *reasonably* visible and readable, the colors used are up to you.
- *Keep* the Week 6 Lab Exercise requirement that at least two components use a font or fonts visibly and obviously different than the default font --

  but otherwise, as long as the results are *reasonably* visible and readable, you get to choose the fonts used.
- Make sure to also *keep* these Week 6 Lab Exercise requirements in your version of `ComputeMore.java`:
  - All the textfields should be right-aligned.
  - At least two components should be given noticeable, visible border of your choice.
  - `try-catch` blocks are appropriately used so that, if the user enters a value that cannot be parsed as a `double`, a `JOptionPane` will be displayed letting the user know that a number must be entered, and all 3 textfields will be set to contain `0`.
  - The `JTextField` displaying the computations' results should be non-editable.
  - The computations' results' display should be formatted to precisely 3 fractional places.

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 `jshell`:

```
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 (*first* version of 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`.