

CS 235 - Homework 6

Deadline:

Due by **11:59 pm** on **Wednesday, October 14, 2015**.

How to submit:

Submit your files using `~st10/235submit` on nrs-projects, with a homework number of 6, by the deadline shown above.

Purpose

To practice using `FlowLayout`, `BorderLayout`, and `GridLayout` layout managers.

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`
- Follow the class Java coding standards mentioned in class and demonstrated in posted in-class examples.

Problem 1

Consider `ColorPlay1.java` from Homework 5 - Problem 3 – remember that there is also a posted example solution, available from the course Moodle site, under "Some solutions", if you prefer.

Modify either your Homework 5 solution or the posted solution into a file `ColorPlay2.java`, whose modified classes meet the following specifications:

- `ColorPlay2Panel` should now use `BorderLayout`, and in its northern region should be a sub-panel using `FlowLayout` containing the `JLabel` including your name.
- `ColorPlay2Panel`'s center region should contain a sub-panel using `GridLayout`, with the grid arranged as you choose but with at least 6 cells, containing the red-green-blue labels and textfields.
- `ColorPlay2Panel`'s south region should contain a sub-panel (a button-panel) that uses `FlowLayout` and contains the button that is pressed to set the background color to the currently-entered red, green, and blue values.
- (if you would like to put anything in the east and west regions -- such as "dummy" labels or blanks for spacing -- that is fine.)
- Colors and font-sizes and font-styles are up to you, as long as the result is readable and attractive.

Submit your resulting `ColorPlay2.java`.

Problem 2

You made a GUI application, `DieRoller`, that rolled a single die in Homework 4 - Problem 4. But, now that we have `GridLayout` and `BorderLayout` available, writing a version that allows you to roll multiple dice should now be more reasonable (from a layout point of view...)

Decide: would you like to roll 2 dice? 3? 4? 5? Choose, make this size a named constant, and create a `DiceRoller` application that uses an array of `GameDie` of that size, and also meets the following specifications:

- decide if you want to use a named constant for the number of sides per die, or if you want to somehow allow the user to specify this (via, for example, an appropriate `JOptionPane`)
- the north should contain sub-panel with a centered, descriptive label including your name
- the center should contain a sub-panel with a two-row grid such that the top row contains buttons to roll their respective die, and the second row contains labels and/or output textfields - your choice - giving the value of the latest roll of that die
 - (whether there are any sub-panels within this sub-panel is up to you)
- the south should contain a sub-panel that has, centered, the sum of the *latest* rolls of all of the dice, using a label and/or output textfield, your choice.
 - (remember, you have an array of `GameDie` instances -- iterating through it and adding all of their current top values should be quite reasonable)
- use at least one visible border somewhere in your application
- (if you would like to put anything in the east and west regions -- such as "dummy" labels of blanks for spacing -- that is fine.)
- Colors and font-sizes and font-styles are up to you, as long as the result is readable and attractive.

Submit your resulting `DiceRoller.java`.

Problem 3

I mentioned during the Week 7 Lab that, within an `actionPerformed` method, you could actually grab a reference to the `SOURCE` of an event by using its `event` parameter.

The `ActionEvent` class includes a method `getSource`, which returns a value of type `Object`, a reference to the component that was acted upon (here, a reference to the button that was clicked).

But what if you want to call `JButton` methods on the `Object` returned? Well, if you know the `Object` is also of type `JButton`, you can cast it to `JButton`.

That is, in `LayoutTrio.java`, in inner class `NumButtonAction`, I could rewrite its `actionPerformed` method using this as follows:

```
public void actionPerformed(ActionEvent event)
{
    // grabbing the SOURCE of this event -- here, a button!
    //     (getSource() returns an Object -- I *know* it
    //     is a JButton, here, so it is safe for me to
```

```

        //      cast this returned result to a JButton)

        JButton clickedButton = (JButton) event.getSource();

        int currButtonVal =
            Integer.parseInt(clickedButton.getText());

        runningTotal += currButtonVal;
        resultsField.setText("" + runningTotal);
    }

```

And -- as you can see above, you can get the text on the top of a JButton using its `getText` method.

Interestingly enough, you can also **CHANGE** the text on a JButton using its `setText` method... So, with that noted, on to this problem!

Create a simple tic-tac-toe game **BOARD** program, whose classes are in a file `TicTac.java`. Since this problem is more about layout practice than about game logic, please note that it is **NOT** required to be very "smart"! It **JUST** needs to meet the following specifications:

- In the north, it needs to have a label stating "tic-tac-toe - by <your name here>" (You can choose if you want this label on a sub-panel or not.)
- In the south, it needs to have a JPanel using `FlowLayout`, containing a "Clear" button. (So, this button will be centered within a South sub-panel, rather than taking up the entire panel)
- In the center, it needs a JPanel containing a 3x3 grid of buttons using `GridLayout`. Let's call these the game buttons.
- When the application starts, all of the game buttons are **blank**.
 - If you click on a blank game button, it should change to show a large X. If you click on a game button with an X, it should change to show a large O. If you click on a game button with an O, it should change to be blank.
- If you click on the Clear button, all of the game buttons should change to be blank.
- Do NOT use a 9-way `if-else-if` (or 9 `if` statements, either...!) to handle the 9 buttons within your code!!
 - Use an array of buttons!
- use at least one visible border somewhere in your application
- (if you would like to put anything in the east and west regions -- such as "dummy" labels of blanks for spacing -- that is fine.)
- Colors and font-sizes and font-styles are up to you, as long as the result is readable and attractive. Make sure the X's and O's on the buttons are large and easy to see!

OPTIONALLY:

- IF you'd like, you can add more sophisticated logic to the above, IF you still meet the above requirements AND CLEARLY DOCUMENT those logic enhancements.

Submit your resulting `TicTac.java`.