

CS 235 - Homework 7

Deadline:

Due by **11:59 pm** on **Wednesday, October 21, 2015**.

How to submit:

Submit your files using `~st10/235submit` on nrs-projects, with a homework number of 7, by the deadline shown above.

Purpose

To provide practice using `Box` and `BoxLayout`, `GridBagLayout`, and no layout manager (absolute positioning), as well as providing more practice designing and building applications that combine the use of several layout managers amongst their panels/subpanels.

Important notes:

- **NOTE -->** You will be selecting the names for some of your Java code for this homework, so I need to know **which files** go with **which problem**. So, create a file `hw7-readme.txt` in which you let me know which file is being submitted for Problem 1, which file is being submitted for Problem 2, and which file is being submitted for Problem 3. (You are expected to use a particular name for your Java code in Problem 4.)
 - Submit your resulting `hw7-readme.txt` along with the source code files for this homework.
- 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

Create a Java application (and associated classes) that:

- displays your name somewhere on the displayed GUI,
- contains at least one visible `Box` that uses `BoxLayout`,
- includes at least two visible components within a `Box` using `BoxLayout`,
- includes at least one visible glue instance within a `Box` using `BoxLayout`, and
- includes at least one visible rigid area instance within a `Box` using `BoxLayout`.
 - (The glue(s) and rigid area(s) may be within different `Boxes` or the same `Box` – your choice.)

Your application must also meet the following additional specifications:

- Since the purpose of this question is to obtain `BoxLayout` experience, you may modify a previous class example or homework problem that currently does NOT use `BoxLayout` so that it now uses `BoxLayout`, **or** you may create something completely new. (Note: You may **not** use Problem 4's `Convert.java` for this Problem 1.)
- Buttons included should be sensitive to user input (should cause something noticeable to happen when they are pressed).
- [for example: you might create an application with a box of buttons in the south, that uses glue and rigid areas to "group" the buttons into subgroups within the box, some of which change the size of a label in the center, and/or some of which change the color of a label in the center, and/or some of which change the text of a label in the center... Or, you might do something completely different.]

Submit your resulting application's `.java` file(s). (Remember to include its name in your `hw7-readme.txt` file!)

Problem 2

Create a Java application (and associated classes) that:

- displays your name somewhere on the displayed GUI,
- contains at least one visible container that uses `GridBagLayout`,
 - ...whose grid contains at least **eight** cells in at least **two** rows and at least **two** columns,
 - ...in which at least **four** components are placed,
 - ...in which at least **one** component **noticeably** spans **more than one** cell, and
 - ...in which at least **one** component noticeably spans **exactly one** cell;
- uses an appropriate private method to help set up the/a `GridBagConstraints` object for each component in the `GridBagLayout` container.

Your application must also meet the following additional specifications:

- Since the purpose of this question is to obtain `GridBagLayout` experience, you may modify a previous class example or homework problem that currently does NOT use `GridBagLayout` so that it now uses `GridBagLayout`, **or** you may create something completely new. (Note: You **MAY** use Problem 4's `Convert.java` for this Problem 2, IF you wish -- IF you indicate this in your `hw7-readme.txt`, and meet this problem's requirements.)
- Buttons included should be sensitive to user input (should cause something noticeable to happen when they are pressed).
- [for example, you might create an application with an array of buttons placed interestingly in a grid (so that they meet at least the above specifications), that change color or label when they are pressed. Or, you could make more interestingly-placed tic-tac-toe squares/rectangles in `TicTac.java`, etc. Or, you might do something completely different.]

Submit your resulting application's `.java` file(s). (Remember to include its name in your `hw7-readme.txt` file!)

Problem 3

Create a Java application (and associated classes) that:

- displays your name somewhere on the displayed GUI,
- contains at least one visible container that uses no layout manager (that uses absolute positioning instead),
 - ...containing at least **five** visible components placed using absolute positioning.

Your application must also meet the following additional specifications:

- Since the purpose of this question is to obtain experience using no layout manager, you may modify a previous class example or homework problem that currently does NOT use absolute positioning so that it now does so, **or** you may create something completely new. (Note: You MAY use Problem 4's `Convert.java` for this Problem 3, IF you wish -- IF you indicate this in your `hw7-readme.txt`, and meet this problem's requirements.)
- Buttons included should be sensitive to user input (should cause something noticeable to happen when they are pressed).
- [for example, you might create an application with five absolutely-positioned buttons, and when one of these buttons is pressed, a label or textfield elsewhere in the application shows which button was pressed last. Or, you might do something completely different.]

Submit your resulting application's `.java` file(s). (Remember to include its name in your `hw7-readme.txt` file!)

Problem 4

Design a simple currency converter, whose classes are in a file `Convert.java`. It should meet the following specifications:

- You are required to include at least one `Box` for one of this application's "sub-containers" (that is, I call them sub-containers below, and using a `Box` rather than a `JPanel` for at least one of them is required for this problem).
 - this box should include an appropriate instance of at least one glue or rigid area (having both is fine, too...)
- Except for the above, the layouts used in the remaining "sub-containers" is up to you. You should attempt a reasonably attractive layout. IF you wish, you may use this for your Problem 2 and/or Problem 3, IF you meet that problem's specifications and you indicate you are doing so in your `hw7-readme.txt`.
- in the north/page_start of `ConvertPanel`: have a sub-container containing a label `U.S. Dollars: $` followed by an initially-blank textfield of a reasonable width that the user **cannot** directly type into. (When contents do appear later in this textfield, they should be **right-justified**.) We'll call this output textfield our dollars-textfield.
- in the south/page_end of `ConvertPanel`: have a sub-container containing a label showing "=", then a textfield of the same length as the dollars-textfield that the user also **cannot** directly type into, that is also initially blank, then a label that **initially** says by `<your last name>`. (When contents appear later in

this textfield, they, too, should be **right-justified**.) We'll call this textfield our results-textfield, and the label containing your name our results-label. (You'll see why in a moment.)

- in the west/line_begin: have a sub-container containing 4 buttons: three that specify currencies of your choice to convert to ("to Euros", etc.), and one that is a Clear button.
- in the center: have a sub-container containing 4 rows and 3 columns of buttons, labeled as:

1	2	3
4	5	6
7	8	9
.	0	

- (note the decimal-point button to the **left** of the 0! BUT the relative spacing of these buttons can vary...)
- We'll call these buttons our digit-buttons (even though one IS a decimal point... 8-))
- The user cannot type in either textfield! So, what can he/she do? Well, when he/she clicks a digit-button, that digit (or decimal point) is concatenated to the right of whatever is currently in the dollars-textfield, and the results-field and results-label should be made blank.
 - NOTE: what if someone tries to add in more than one decimal point? Then a `JOptionPane` should pop up telling them that only one decimal point is allowed, and not concatenate the "excess" decimal point.
 - (Hint -- might there be a `String` method that you could use to tell if you currently have a certain character within a `String` instance?)
- When the user clicks the Clear button, both the dollars-field and the results-field should be cleared (made blank), as should the results-label.
- When the user clicks one of the "to <some currency>" buttons, then the program should compute how many of that currency would be equivalent to the number of dollars currently entered in the textfield (look up the current rate using Google, document in a comment the date that you grabbed that rate...), then display that amount to **2 fractional places** in the results-textfield, and change the results-label to say the currency of the result (for example, euros).
 - (note: be sure to code those conversion rates as descriptively-named **named constants**!)
- Colors and font-sizes and font-styles are up to you, as long as the result is readable and attractive. You can even add more currency conversions if you would like...

Submit your resulting `Convert.java`.