**CSCI 470/502 Assignment 4 – Tip Calculator Summer 2019**

**150 points**

In this assignment, you will write a graphical user interface for the tip calculator application that you wrote for Assignment 1.

*Initial screen shot of the application*

![A screenshot of a cell phone

Description automatically generated]()

**Class TipCalculator**

This class from Assignment 1 can be re-used with no changes. The get methods for the three data members will not be used and should therefore be deleted.

**Class TipApp**

This is a new class that will encapsulate the user interface of the application. It should extend JFrame and implement the interfaces ActionListener and ChangeListener.

*Data Members*

* A TipCalculator object. (This could be a local variable in the calculateTips() method if you prefer.)
* You will also need data members for most of the UI elements – the JTextField, JSlider, JSpinner, the two JButtons, and the two output JLabels. The labels that do not have their text changed can be created as local variables instead.

*Methods*

* main() - should create an instance of TipGUI and then use it to call method(s) to set up the user interface elements (e.g., createGUI()). Because Swing is not thread-safe, you might want to create your Swing objects the Event Dispatch Thread, or EDT. See Assignment 0, Part 2, or Lecture Notes 12 for examples of the code pattern that is typically used to do this.
* actionPerformed() - Override this method to handle ActionEvents from the two buttons. This method will run on the EDT.
* The “Calculate” button should retrieve the text from the text field and try to convert it to a double number. If that is successful, then you can use the number to set the bill amount for the TipCalculator and then call the getTotalBill() and getTotalShare() methods to compute the two output values. Finally, display the output values in the appropriate labels.
  + - If the conversion to double fails or the user enters a value less than or equal to 0 for the bill amount, you will need to display an error message. In a GUI application, printing error output to the console can be useful to the *developer*, but it won’t help the *user*. Instead, display the error messages using the class method JOptionPane.showMessageDialog(). That can produce a nice message dialog box for the user similar to the following examples:

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |

* The “Clear” button should set the user interface back to its initial state (empty string in the text field, slider value set to 20, spinner value set to 1) and also set the data members of the TipCalculator object back to their default values (0 for the bill amount, 20 for the tip percentage, and 1 for the party size).
* stateChanged() - Override this method of the ChangeListener interface to handle any and all ChangeEvents from the slider and spinner. This method will run on the EDT.
* Moving the JSlider should update the tip percentage data member of the TipCalculator object.
* Changing the JSpinner should update the party size data member of the TipCalculator object.

**Output**

*Screen shot of the results of pressing the “Calculate” button*

**![A screenshot of a cell phone

Description automatically generated]()**

**Notes**

* It is not necessary to match the UI design in the screenshots above precisely. A simple 6 x 2 GridLayout for the application is acceptable but will not look particularly good or look much like the screenshots because GridLayout ignores the preferred sizes of the UI elements that you add to it and just displays them all at the same size.
* Do not use the asterisk in import statements!
* A number of more advanced tricks were used to improve the appearance of the application:

1. An empty border around the main panel so that the UI elements were not jammed up against the edges of the frame.
2. HTML was used to display the header labels in boldface.
3. A more complex scheme of nested layouts (vertical and horizontal BoxLayouts, FlowLayout for the button row, and GridLayout for the output label rows) were used so that the UI elements are displayed at their preferred sizes and there is reasonable spacing between them.
4. Setting the preferred size on the “Clear” button to match the preferred size of the “Calculate” button.

* It is recommended that you first try to get the application to work using a simple GridLayout and then, if time permits, play around with a copy of it and attempt to improve the appearance of the UI.
* See the Java Tutorials sections titled *How to Use Sliders* and *How to Use Spinners* as well as the JSlider and JSpinner API documentation for tips and example code regarding those UI objects.
* To make grading by the TAs easier, please do NOT put a package statement in the .java files to be submitted.
* As before, document appropriately and submit the two or three .java files on Blackboard.