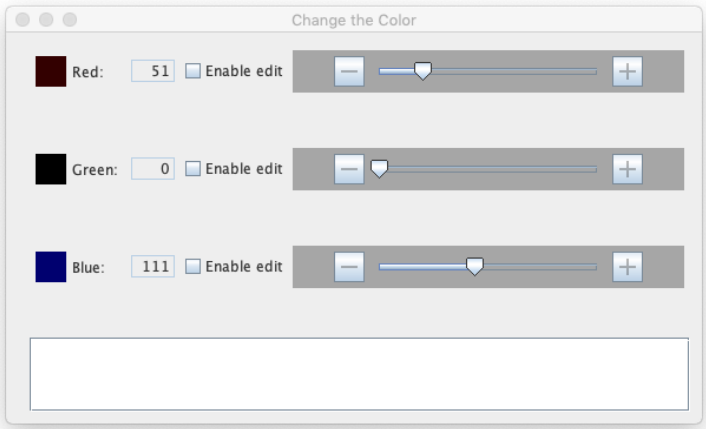
School of Engineering and Technology, University of Washington Tacoma
TCSS 305 Programming Practicum, Winter 2021
Assignment 4 – Color Adjuster
Value: 25% of the course grade
Due: Friday, 12 March 2021, 23:59:59

Program Description:

This assignment is designed to test your understanding of graphical user interface components and Event Driven programming in Java using Swing. You will write and attach event handlers to an existing graphical user interface (GUI) for an application that displays and manipulates a Color Model.

When the program initially loads, *two* windows should load with the following appearance:



What is provided:

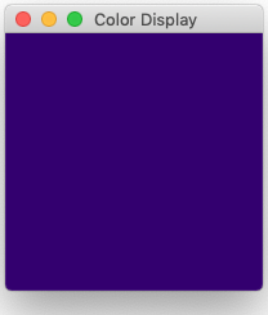
- The Color Model classes (model)
- All GUI classes with Swing components placed
- Plumbing for GUI components (views) to respond to changes to the Color Model

What you need to complete:

- Add event handlers to components that respond to user interactions
- The **`ColorModel.java`** class
- The **`ColorPanel.java`** class

Project Structure

- **`(default package)`**
  - **`Application.java`**
    - No Changes, main method
- **`controller`**
  - **`BlueRowPanel.java`**
    - **requires event handlers**
  - **`ColorController.java`**
    - No changes, is the Frame for the controller
  - **`GreenRowPanel.java`**
    - **requires event handlers**
  - **`RedRowPanel.java`**
    - **requires event handlers**



- **`model`**
  - **`ColorModel.java`**
    - **Requires implementation of Property Change API to act as Observable Model**
  - **`MutableColor.java`**
    - No changes, interface that defines behaviors specific to a Color that can be changed
    - **These are the behaviors that your event handlers will use when they need to change the state of the Color Model**
  - **`PropertyChangeEnabledMutableColor.java`**
    - No changes, interface that defines Property Change behaviors to notify interested parties of changes to the Color Model
- **`view`**
  - **`ColorPanel.java`**
    - **requires Observer implementation**

Implementation Guidelines:

The following three GUI classes require event handlers to react to user input. **The following project description includes statements for BlueRowPanel.java only. Note that you must implement handlers for all three classes.**

`BlueRowPanel.java`

`GreenRowPanel.java`

`RedRowPanel.java`

BlueRowPanel.java includes the following GUI swing components with the following user interaction requirements (displayed left to right)

- **`JPanel`** **`myColorDisplayPanel`**
- **`JLabel`** Local variable
- **`JTextField`** **`myValueField`**
  - Initially un-editable
  - Allows user interaction:
    - When editable, the user may enter any text
      - If the text is an integer value [0-255] and the user presses the enter key, the new integer value is sent to the Color Model
      - If the text is anything OTHER than an integer value [0-255] and the user presses the enter key, disregard the text and return the text field to the correct value for Blue in the Color Model
- **`JCheckBox`** **`myEnableEditButton`**
  - Initially un-checked
  - Allows user interaction:
    - When checked, **`myValueField`** is editable
    - When un-checked, **`myValueField`** is un-editable
- **`JButton`** **`myDecreaseButton`**
  - Initially enabled
  - Allows user interaction:
    - When clicked, decrease the Blue value in the color model by 1
    - When the Color Model's blue value is 0, disable this button, ensure that the button is enabled for any other value
- **`JSlider`** **`myValueSlider`**
  - Allows user interaction:
    - When dragged, change the Blue value in the Color Model to the new value of the slider
- **`JButton`** **`myIncreaseButton`**
  - Initially enabled
  - Allows user interaction:
    - When clicked, increase the Blue value in the color model by 1
    - When the Color Model's blue value is 255, disable this button, ensure that the button is enabled for any other value

Color Model

The following classes and interfaces make up the Color Model. The application instantiates only one **`ColorModel`** object and sends a reference to the one **`ColorModel`** object to interested parties (including **`[Red/Green/Blue]RowPanel`** which each have an instance field called **`myColor`**)

`model:`

- **`ColorModel.java`**
- **`MutableColor.java`**
- **`PropertyChangeEnabledMutableColor.java`**

When one of the **`[Red/Green/Blue]RowPanel`** objects needs to make a change to the Color Model, it will do so through the **`MutableColor`** interface. The following is an abridged API documentation for **`MutableColor`**. See the JavaDoc found in **`MutableColor`** for the full API documentation.

`int`	**`getRed`**	Returns the red component in the range 0-255 in the default RGB space.
`void`	**`setRed(int:theRed)`**	Sets red component with the specified value in the range (0 - 255). Notify all registered Observers of both the change in Red value (as an **`int`**) and then the change to the entire color (as an **`java.awt.Color`** object)
`void`	**`adjustRed(int:theOffset)`**	Adjust the red component by the specified positive or negative offset value. Notify all registered Observers of both the change in Red value (as an **`int`**) and then the change to the entire color (as an **`java.awt.Color`** object)
`int`	**`getGreen`**	Returns the green component in the range 0-255 in the default RGB space.
`void`	**`setGreen(int:theGreen)`**	Sets green component with the specified value in the range (0 - 255). Notify all registered Observers of both the change in Green value (as an **`int`**) and then the change to the entire color (as an **`java.awt.Color`** object)
`void`	**`adjustGreen(int:theOffset)`**	Adjust the green component by the specified positive or negative offset value. Notify all registered Observers of both the change in Green value (as an **`int`**) and then the change to the entire color (as an **`java.awt.Color`** object)
`int`	**`getBlue`**	Returns the blue component in the range 0-255 in the default RGB space.
`void`	**`setBlue(int:theBlue)`**	Sets blue component with the specified value in the range (0 - 255). Notify all registered Observers of both the change in Blue value (as an **`int`**) and then the change to the entire color (as an **`java.awt.Color`** object)
`void`	**`adjustBlue(int:theOffset)`**	Adjust the blue component by the specified positive or negative offset value. Notify all registered Observers of both the change in Blue value (as an **`int`**) and then the change to the entire color (as an **`java.awt.Color`** object)

Color View

`ColorPanel.java` should be an Observer of the Model. Notice that the class extends **`JPanel`**. Implement **`PropertyChangeListner`** and change the background color of the object to match the current color represented by the Model.

Do not store a reference to the Model in an instance field. **`ColorPanel`** objects only receive updates from the Model via the Observer Design Pattern.

Hints:

`BlueRowPanel.java` includes an empty method called **`addListeners()`**. *Attach* all listeners to the corresponding swing components in this method. Note: the listeners may be defined outside of the method.

The Color Model uses the Property Change API. This API allows a model to inform attached listeners of changes to its state. To listen for state changes in the color model, implement **`PropertyChangeListener`** (implements the method **`propertyChange(PropertyCheangeEvent)`**) and add the implementing object to the Color Model through the **`addPorpertyChangeListener()`** method.

The existing code includes many examples of this:

- **`ColorController.java`** is an attached **`PropertyChangeListener`** and displays the **`toString()`** of the color model to a **`JTextArea`** called **`myOutput`** when it is informed of Color Model state changes
- **`BlueRowPanel.java`** is an attached **`PropertyChangeListener`** and it changes the value of **`myValueField`**, **`myValueSlider`**, and **`myColorDisplayPanel`** when it is informed of color model state changes
    - Feel free to add statements to **`propertyChange()`** as needed
        - HINT: Are there any requirements that are based on the state of the color model?
- Take a close look at all of these examples and notice the different Property values used when the model notifies its observers of a change. When implementing the model, ensure that it is sending out these different notifications. For example, a change in the Red value should, using one statement, notify the observers of the **`int`** change in Red, then, using a second statement, notify the observers of the **`java.awt.Color`** change in for the entire Color value.

Extra Credit

Up to 10% extra credit can be earned by completing the following challenge. You should notice that the three **`[Red/Green/Blue]RowPanel`** classes are almost the same. The only differences are the initial values, text on a **`JLabel`** (**`rowLabel`**), and the behavior when the user interacts with the components.

This configuration does not follow good Object-oriented practices. It uses three (very similar) classes to define and instantiate three (very similar) objects. There is code duplication across the three classes. A better solution would be one class to define and instantiate three (very similar) objects.

Create a new class called **`ValueRowPanel`** that is used to instantiate the three **`RowPanel`** objects and incorporate it into ColorController.java

Up to 5% extra credit can be earned by completing the task with no restrictions.

An additional 5% extra credit can be earned if you do not include any branching statements (**`if`**, **`switch`**, **`:?`**) or logic expressions to handle the difference between Red/Green/Blue.

An example of what is NOT allowed for the additional 5%: (NOTE: pseudo-ish code)

```
if (red) {
      setRed(value);
} else if (green) {
      setGreen(value);
} else if (blue) {
      setBlue(value);
}
```

Submission and Grading:

Create your Eclipse project by downloading the `hw4-project.zip` file from the Assignment 4 page on Canvas, importing it into your workspace (as described for `hw0-project.zip` in Assignment 0), and using "Refactor" to change "username" in the project name to your UWNetID.

You must check your Eclipse project into Subversion (following the instructions from Lecture 1 and Assignment 0), including all configuration files that were supplied with it (even if you have not changed them from the ones we distributed). When you have checked in the revision of your code you wish to submit, make a note of its Subversion revision number. To get the revision number, *perform an update* on the top level of your project; the revision number will then be displayed next to the project name.

After checking your project into Subversion, you must submit (on Canvas) an *executive summary*, containing the Subversion revision number of your submission, an "assignment overview" (1 paragraph, up to about 250 words) explaining what you understand to be the purpose and scope of the assignment, and a "technical impression" section (1-2 paragraphs, about 200-500 words) describing your experiences while carrying out the assignment. The assignment overview shows how well you understand the assignment; the technical impression section helps to determine what parts of the assignment need clarification, improvement, *etc.* for the future.

The filename for your executive summary must be "`username-coloradjuster.txt`", where `username` is your UWNetID. An executive summary template is available on the Canvas. Executive summaries will *only* be accepted in plain text format – other file formats (RTF, Microsoft Word, Acrobat PDF, Apple Pages) are *not* acceptable. You will, in general, not lose any points on your executive summary itself unless you fail to turn it in or it is short of the length requirement and / or trivial.

Part of your program's score will come from its "external correctness." For this assignment, external correctness is graded by running the GUI and examining the result. Exceptions should not occur under normal usage. The program should not produce any console output.

For this assignment, the percentage breakdown is 10% executive summary, 65% external correctness, 25% internal correctness.