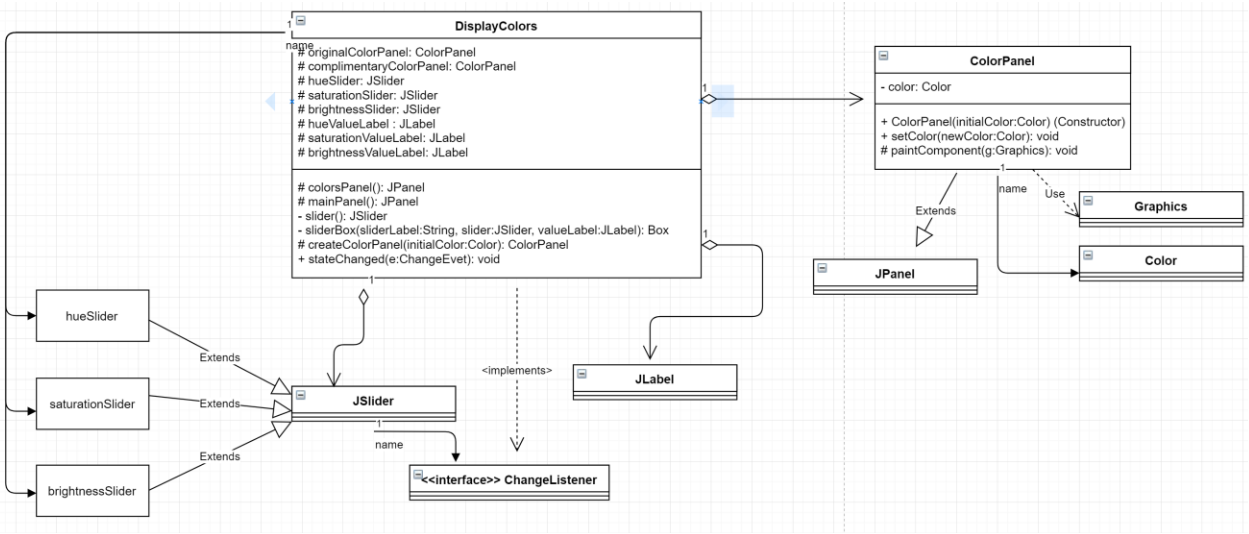
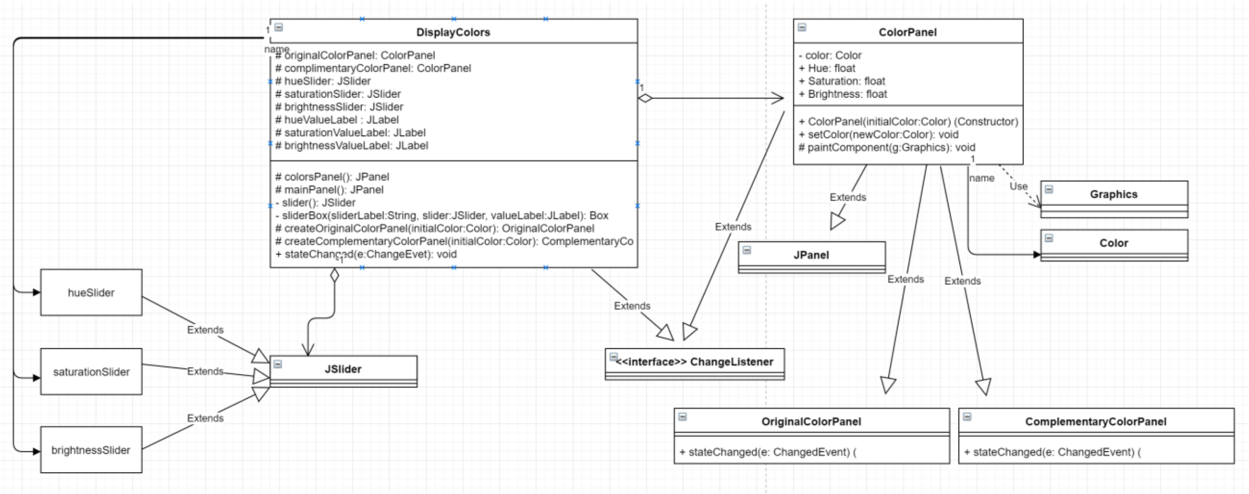
Task 1:


DisplayColors
1
name
# originalColorPanel: ColorPanel
# complimentaryColorPanel: ColorPanel
# hueSlider: JSlider
# saturationSlider: JSlider
# brightnessSlider: JSlider
# hueValueLabel : JLabel
# saturationValueLabel: JLabel
# brightnessValueLabel: JLabel
# colorsPanel(): JPanel
# mainPanel(): JPanel
- slider(): JSlider
- sliderBox(sliderLabel:String, slider:JSlider, valueLabel:JLabel): Box
# createColorPanel(initialColor:Color): ColorPanel
+ stateChanged(e:ChangeEvet): void
ColorPanel
- color: Color
+ ColorPanel(initialColor:Color) (Constructor)
+ setColor(newColor:Color): void
# paintComponent(g:Graphics): void
1
1
1
1
name
Use
Extends
Graphics
Color
JPanel
hueSlider
saturationSlider
brightnessSlider
Extends
Extends
Extends
JSlider
1
name
<implements>
JLabel
<<interface>> ChangeListener

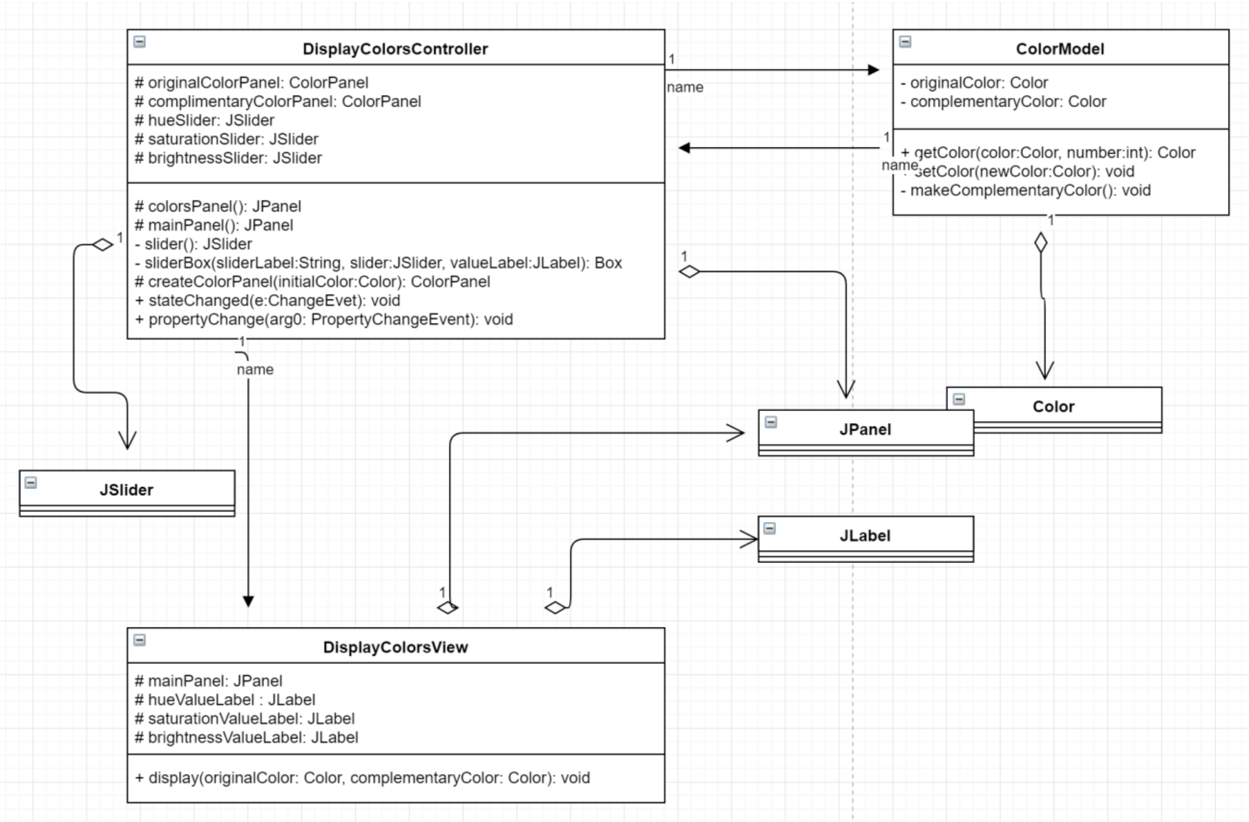
In this first task, the views and controls are held in the DisplayColors class. This class calculates the colors that will be displayed in ColorPanel. DisplayColors uses a ChangeListener and ColorPanel uses Graphics, Color, and JPanel.

Task 2:


DisplayColors
1
name
# originalColorPanel: ColorPanel
# complimentaryColorPanel: ColorPanel
# hueSlider: JSlider
# saturationSlider: JSlider
# brightnessSlider: JSlider
# hueValueLabel : JLabel
# saturationValueLabel: JLabel
# brightnessValueLabel: JLabel
# colorsPanel(): JPanel
# mainPanel(): JPanel
- slider(): JSlider
- sliderBox(sliderLabel:String, slider:JSlider, valueLabel:JLabel): Box
# createOriginalColorPanel(initialColor:Color): OriginalColorPanel
# createComplementaryColorPanel(initialColor:Color): ComplementaryCo
+ stateChanged(e:ChangeEvet): void
1
1
ColorPanel
- color: Color
+ Hue: float
+ Saturation: float
+ Brightness: float
+ ColorPanel(initialColor:Color) (Constructor)
+ setColor(newColor:Color): void
# paintComponent(g:Graphics): void
1
name
Use
Extends
Extends
Extends
Extends
Extends
Graphics
Color
JPanel
hueSlider
saturationSlider
brightnessSlider
Extends
Extends
Extends
JSlider
<<interface>> ChangeListener
OriginalColorPanel
+ stateChanged(e: ChangedEvent) (
ComplementaryColorPanel
+ stateChanged(e: ChangedEvent) (

The second rendition has an OriginialColorPanel and ComplementaryColorPanel that listen for changes in the slider values in DisplayColors. These two classes implement the stateChanged method differently.

Task 4:


DisplayColorsController
# originalColorPanel: ColorPanel
# complimentaryColorPanel: ColorPanel
# hueSlider: JSlider
# saturationSlider: JSlider
# brightnessSlider: JSlider
# colorsPanel(): JPanel
# mainPanel(): JPanel
- slider(): JSlider
- sliderBox(sliderLabel:String, slider:JSlider, valueLabel:JLabel): Box
# createColorPanel(initialColor:Color): ColorPanel
+ stateChanged(e:ChangeEvet): void
+ propertyChange(arg0: PropertyChangeEvent): void
ColorModel
- originalColor: Color
- complementaryColor: Color
+ getColor(color:Color, number:int): Color
- setColor(newColor:Color): void
- makeComplementaryColor(): void
1
name
1
name
1
1
1
1
name
Color
JPanel
JSlider
JLabel
1
1
DisplayColorsView
# mainPanel: JPanel
# hueValueLabel : JLabel
# saturationValueLabel: JLabel
# brightnessValueLabel: JLabel
+ display(originalColor: Color, complementaryColor: Color): void

A Model-View-Controller patter (MVC) is a architectural patterns that separates an app into three components: the model, view, and controller. I split the DisplayColors class into two separate classes, Display ColorsView and DisplayColorsController. The view class utilizes JLabyels and JPanels that reflect non-slider information of DisplayColors which allows for information to be hid from the user. The controller class helps update the view; it reads and deals with input. It updates the view and stores color information into the model. The controller implements a ChangeListener that allows it to listen to JSliders value changing. The controller keeps ColorPanels in order to update ColorModel. The model stores data for the complementary color, in essence it stores the formula for the complementary color.