Alvin Tran

CSE 220-50

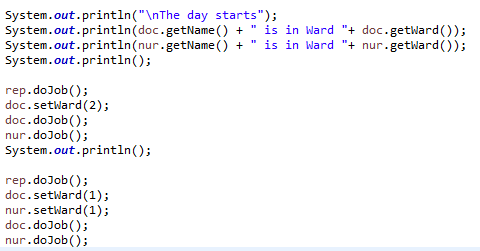
Assignment #5

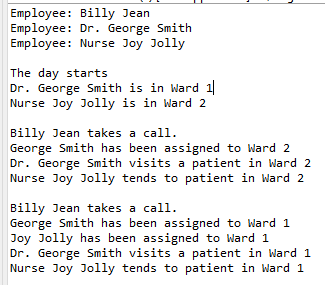
Submission Date: 7/19/2021

**Problem 1**

This problem requires us to implement different employees of a hospital as classes, and then have each contain methods that act out an action that would fit for that role and printing an appropriate message.

We utilize **Inheritance** as many of these classes will be sharing methods and data. More specifically, we will define an **Employee** class to inherit, which contains the classes **getName**, **doJob,** and data for **firstName** and **lastName.** Then, I specify another class to inherit, **medicalEmployee.** This class has methods and data representing what **ward** they are assigned. Finally, we create the actual employee classes. I created **Doctor, Nurse,** and **Receptionist**. The first 2 **extends** the medicalEmployee class, and the last **extends** employee. Each one **overrides** the **doJob()** method created by the employee class, printing an appropriate message. In addition, Doctor and Nurse overrides getName() to add the titles “Dr.” and “Nurse” to the String output.



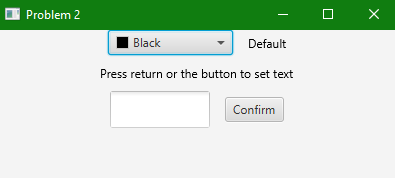


**Set Up and Output; Note the use of doJob() and different ouputs.**

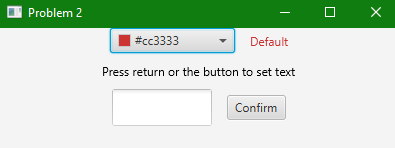
**Problem 2**

This problem wants us to display a JavaFX application contain a text field, Color Picker, button, and some text. The text field can control said text by typing and pressing “return” or the button. The Color Picker also assigns the text color.

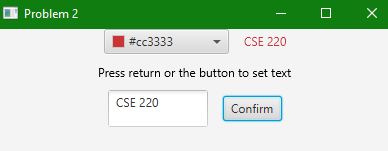
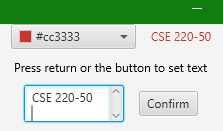
I organize my **ColorPicker** and **Text display** into a **HBox**, some instruction text into an HBox, the **TextArea** and **Button** for user input into an HBox, and then put that all into a **VBox** to put into a scene for formatting. I create an **ActionEvent** to tag onto my ColorPicker and setting the **Text display** to the value given by **ColorPicker.getValue()**. The **TextArea** is assign a **KeyEvent** which uses **event.getCode()** to find which key was entered and a **Switch** statement to check if the return (enter) key was press to apply the text change with **Text.setText(TextArea.getText()).** The button similarly is set with an ActionEvent that does the same, without the need to check if a certain key is pressed.



**Launch**



**Updating Color**

**Updating Text: Note different items highlighted to demonstrate both methods of update**

**Problem 3**

This problem requires the reuse of the program shown in class RubberLines and modifying it to include a Color Picker to determine the Line color.

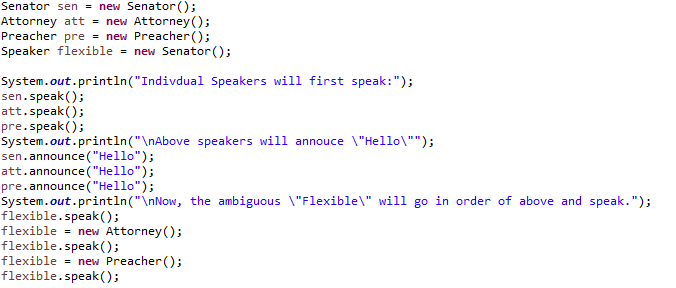
We reuse the source code given since it was posted on Blackboard. After so, we add the Color Picker by making an new object, and use **group.getChildren().add(ColorPicker)** to add our Color Picker. After so, we modify the **processMousePress** **mouseEvent** to include **currentLine.setStroke(colorPicker.getValue())** to set our color. 

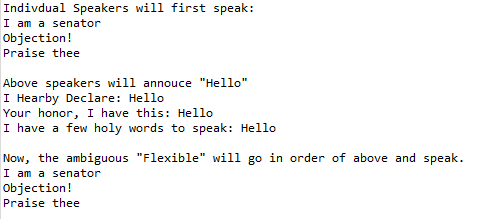
**Demonstration of ColorPicker to change color and create lines of different color**

**Problem 4**

This problem requires the use of the Speaker Interface shown in class and using that to create Classes of Senator, Attorney, and Preacher. Furthermore, we should demonstrate polymorphism from the chapter.

We reuse the **interface** from class and create new **Senator, Attorney,** and **Preacher classes,** which **extends** the **Speaker** interface. Each one has a unique **Speak()** and **Annouce()** method. We have a driver classes creating one of each class, and then an object representing our **Speaker** to demonstrate **Polymorphism**.





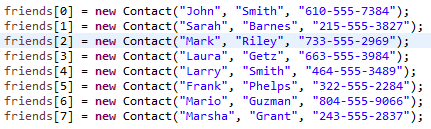
**Set Up and Output**

**Note that Speaker flexible can be any of the 3 classes**

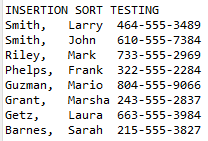
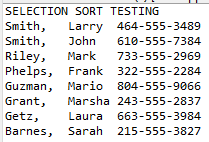
**Problem 5**

This problem requires the reuse the Sorting class shown in class and rewrite it to descending order instead of ascending class.

I reuse the **Contacts** and **Sorting** class since those were on Blackboard. I simply changed any instance of “<” and “>” to the opposite, “>” and “<” , when it applies to comparing values. The logic behind this is that normally you check to see if a value is less than your current value to put it on the left (putting it in ascending order). By reversing the logic, we check if anything is higher, so we would put the highest value on the left, making a descending order sort.



**Initial List for Both Sorts**



**Output as expected in Descending Order**