Alvin Tran

CSE 220-50

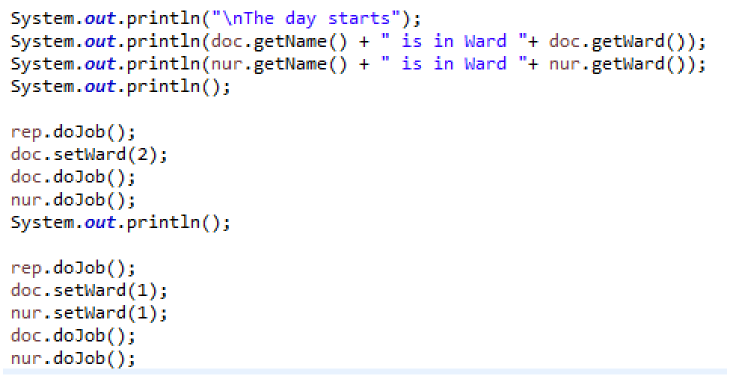
Assignment #5

Submission Date: 7/19/2021

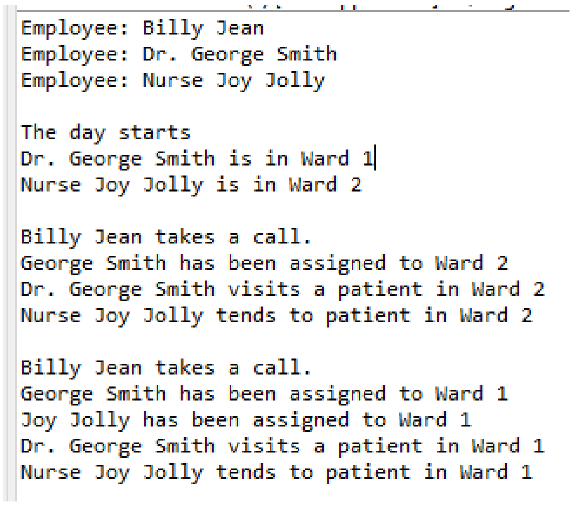
Problem 1

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.



```
System.out.println("\nThe day starts");
System.out.println(doc.getName() + " is in Ward "+ doc.getWard());
System.out.println(nur.getName() + " is in Ward "+ nur.getWard());
System.out.println();

rep.doJob();
doc.setWard(2);
doc.doJob();
nur.doJob();
System.out.println();

rep.doJob();
doc.setWard(1);
nur.setWard(1);
doc.doJob();
nur.doJob();
```



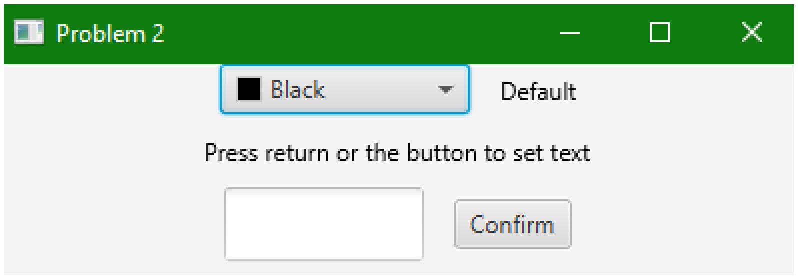
```
Employee: Billy Jean
Employee: Dr. George Smith
Employee: Nurse Joy Jolly

The day starts
Dr. George Smith is in Ward 1
Nurse Joy Jolly is in Ward 2

Billy Jean takes a call.
George Smith has been assigned to Ward 2
Dr. George Smith visits a patient in Ward 2
Nurse Joy Jolly tends to patient in Ward 2

Billy Jean takes a call.
George Smith has been assigned to Ward 1
Joy Jolly has been assigned to Ward 1
Dr. George Smith visits a patient in Ward 1
Nurse Joy Jolly tends to patient in Ward 1
```

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

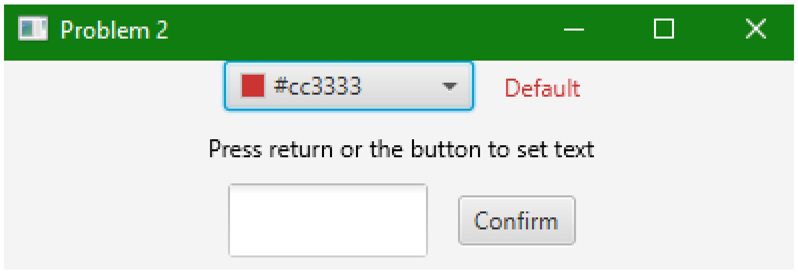
Problem 2

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.

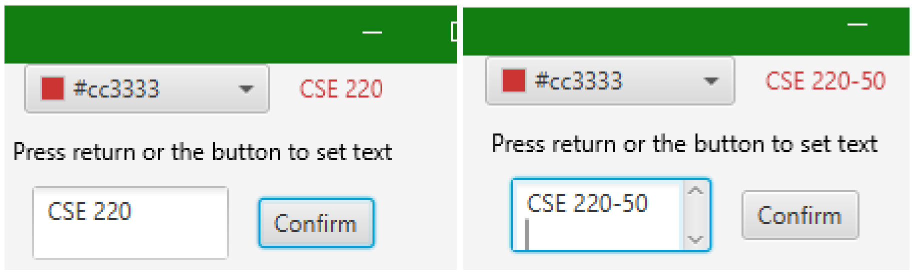
I organize my **ColorPicker** and **Text display** into a **HBox**, some instruction text into an HBox, the **TextArea** and **Button** for user input into an HBox, and then put that all into a **VBox** to put into a scene for formatting. I create an **ActionEvent** to tag onto my ColorPicker and setting 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

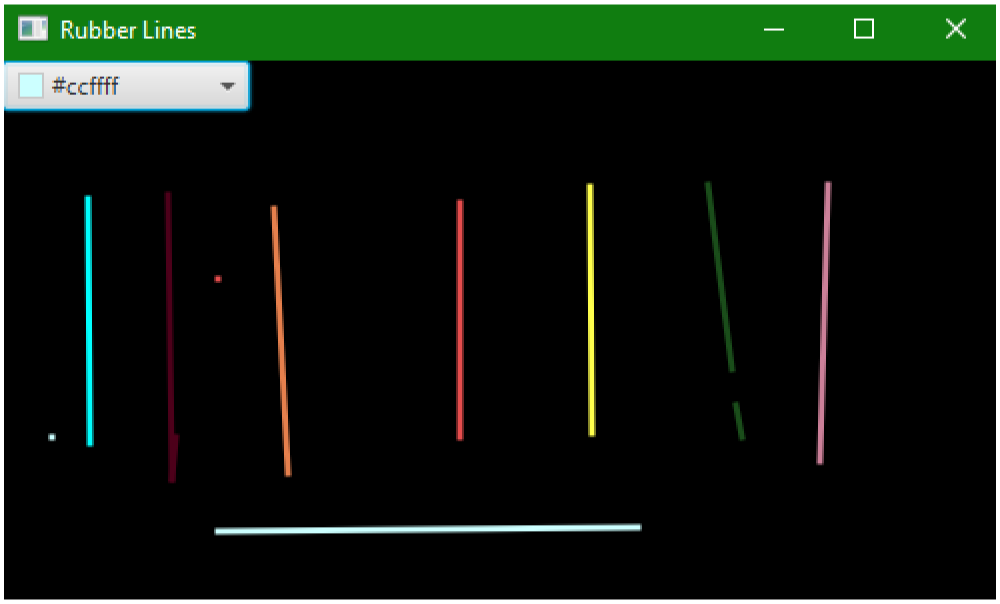

#cc3333
CSE 220
Press return or the button to set text
CSE 220
Confirm
#cc3333
CSE 220-50
Press return or the button to set text
CSE 220-50
Confirm

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.

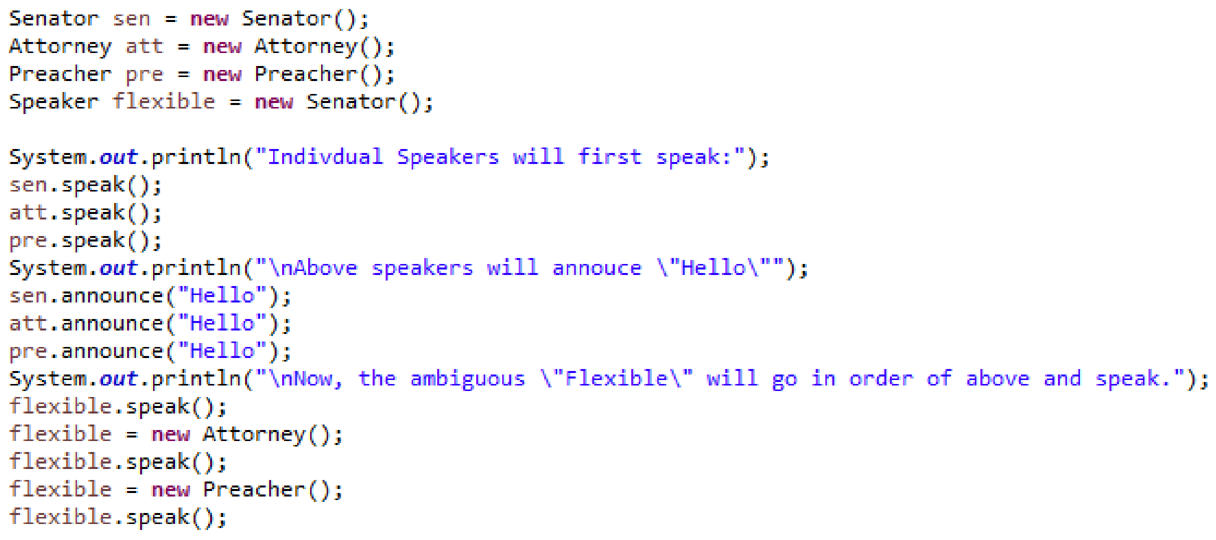


Demonstration of ColorPicker to change color and create lines of different color

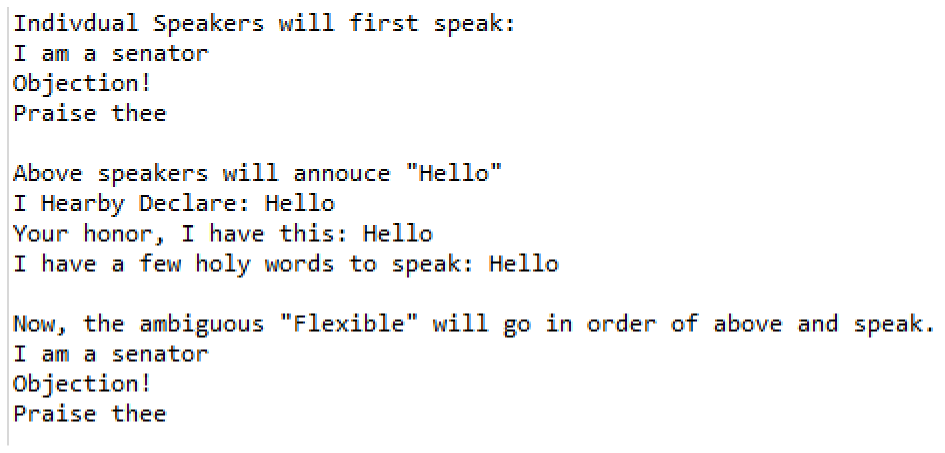
Problem 4

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**.



```java
Senator sen = new Senator();
Attorney att = new Attorney();
Preacher pre = new Preacher();
Speaker flexible = new Senator();

System.out.println("Indivdual Speakers will first speak:");
sen.speak();
att.speak();
pre.speak();
System.out.println("\nAbove speakers will annouce \"Hello\"");
sen.announce("Hello");
att.announce("Hello");
pre.announce("Hello");
System.out.println("\nNow, the ambiguous \"Flexible\" will go in order of above and speak.");
flexible.speak();
flexible = new Attorney();
flexible.speak();
flexible = new Preacher();
flexible.speak();
```



```
Indivdual Speakers will first speak:
I am a senator
Objection!
Praise thee

Above speakers will annouce "Hello"
I Hearby Declare: Hello
Your honor, I have this: Hello
I have a few holy words to speak: Hello

Now, the ambiguous "Flexible" will go in order of above and speak.
I am a senator
Objection!
Praise thee
```

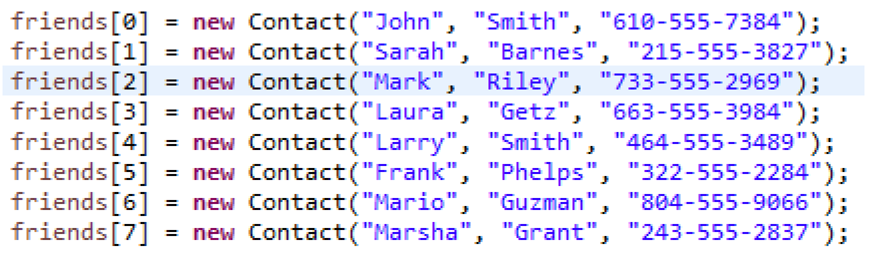
Set Up and Output

Note that Speaker flexible can be any of the 3 classes

Problem 5

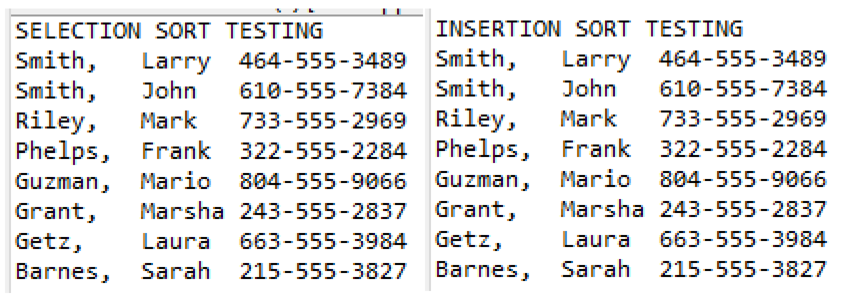
This problem requires the reuse the Sorting class shown in class and rewrite it to descending order instead of ascending class.

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.



```
friends[0] = new Contact("John", "Smith", "610-555-7384");
friends[1] = new Contact("Sarah", "Barnes", "215-555-3827");
friends[2] = new Contact("Mark", "Riley", "733-555-2969");
friends[3] = new Contact("Laura", "Getz", "663-555-3984");
friends[4] = new Contact("Larry", "Smith", "464-555-3489");
friends[5] = new Contact("Frank", "Phelps", "322-555-2284");
friends[6] = new Contact("Mario", "Guzman", "804-555-9066");
friends[7] = new Contact("Marsha", "Grant", "243-555-2837");
```

Initial List for Both Sorts



```
SELECTION SORT TESTING            INSERTION SORT TESTING
Smith,    Larry  464-555-3489     Smith,    Larry  464-555-3489
Smith,    John   610-555-7384     Smith,    John   610-555-7384
Riley,    Mark   733-555-2969     Riley,    Mark   733-555-2969
Phelps,   Frank  322-555-2284     Phelps,   Frank  322-555-2284
Guzman,   Mario  804-555-9066     Guzman,   Mario  804-555-9066
Grant,    Marsha 243-555-2837     Grant,    Marsha 243-555-2837
Getz,     Laura  663-555-3984     Getz,     Laura  663-555-3984
Barnes,   Sarah  215-555-3827     Barnes,   Sarah  215-555-3827
```

Output as expected in Descending Order