Final Assignment for CS-0401

1 Introduction

In this assignment you will create a program with an Graphical User Interface that will help customers to decide what car and car features that they want. In doing such a program, you will use many of the techniques that you have been learning in our course, to name a few of them:

- Work with Java FXML:
  - GUI components:
    - Choice boxes
    - Labels
    - Image views
    - Text areas
    - Check boxes
    - Buttons
    - Radio Buttons
    - Text Area
  - Layout managers:
    - Border layout
    - Horizontal Boxes
    - Vertical Boxes
    - Radio Button grouping
  - Initializing and modifying GUI components programmatically
  - Getting status of GUI components programmatically
  - Handling mouse click events
- Working with classes:
  - Inheritance:
    - Creating a super class Car
    - Creating subclasses for each Car being sold
  - Polymorphism:
    - Creating a list of Cars
  - Class fields and Class methods
  - Interface
- Working with General Java concepts:
  - Variables
  - Objects
  - Lists of objects
  - Methods
  - FOR loops
  - Mathematical operations
  - Number formatting
  - Etc.
- Anything else that I have forgot to mention here…

2 The Overall App Feature

Please watch the video found at

```
http://www.paulobrasko.com/pitt-cs0401/assignments/assignment3/CarDealerJavaFXMLGUI.mp4
```

to review what features your app must have.

Notes:

- This is an old video, so few statements do not apply to our current CS-0401 course, such as “this is the 3rd assignment…”.
- The video does not show, besides updating the final price based when the insurance is changed, your App must also include the type of insurance in the text area, such as “Insurance Type: 3 year warranty”.

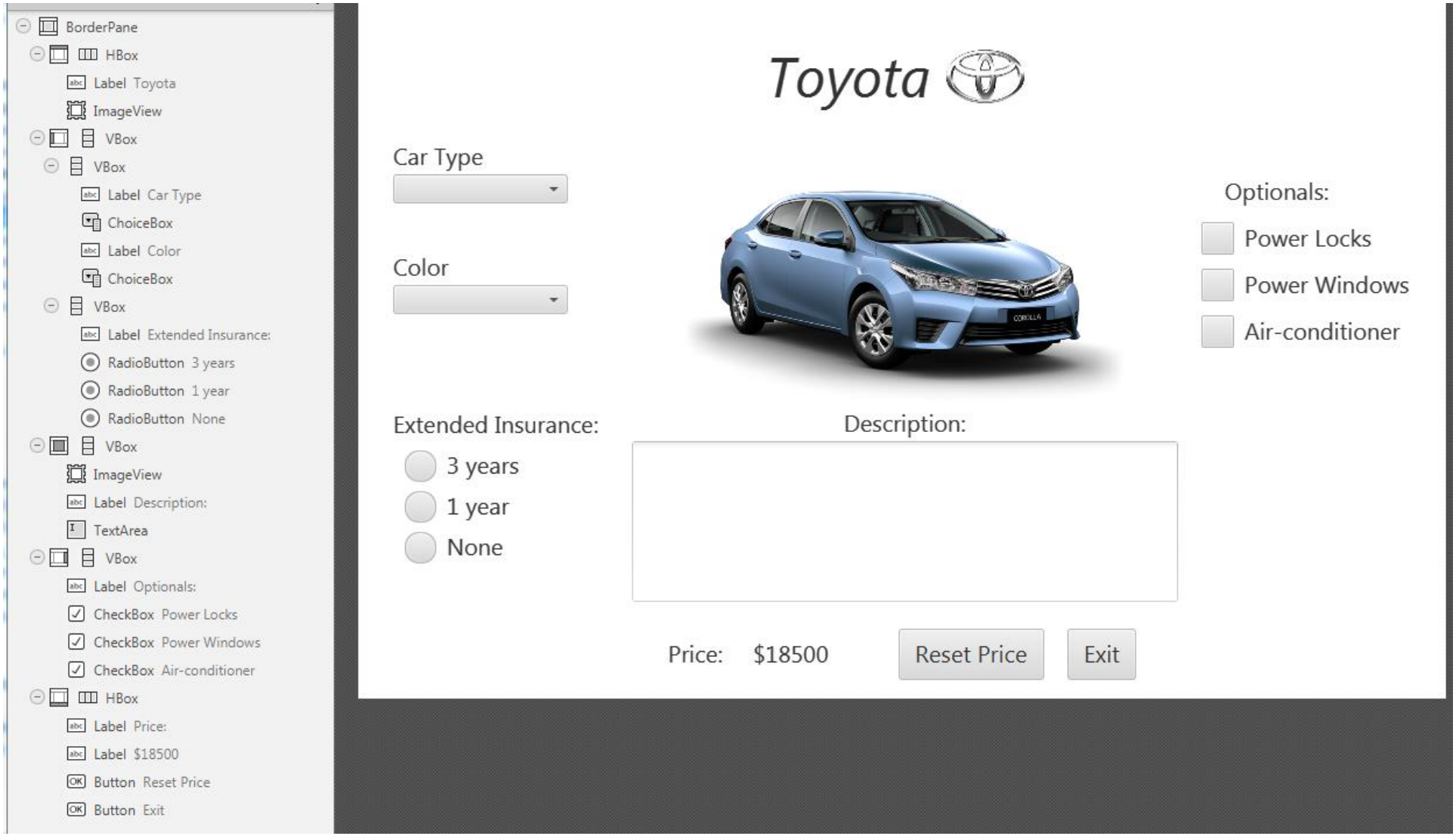
3 General GUI Layout

Since we will have no much time to go over everything you should know about Java FXML in our time-limited class, I would like to instruct you on how I would like to see you creating this GUI. In this week lab, try to create the whole GUI as presented in the next page.

Please watch video from the link below which talks about each of the layout component in more detail.

```
http://www.paulobrasko.com/pitt-cs0401/tutorial-videos/JavaFXML/JavaFXML_LayoutManagers.mp4
```

After watching the video above, during the lab you should create the GUI as shown in the next page, using Border, VBOX, and HBOX layouts (containers).


BorderPane
HBox
Label Toyota
ImageView
VBox
VBox
Label Car Type
ChoiceBox
Label Color
ChoiceBox
VBox
Label Extended Insurance:
RadioButton 3 years
RadioButton 1 year
RadioButton None
VBox
ImageView
Label Description:
TextArea
VBox
Label Optionals:
CheckBox Power Locks
CheckBox Power Windows
CheckBox Air-conditioner
HBox
Label Price:
Label $18500
Button Reset Price
Button Exit
Toyota
Car Type
Color
Optionals:
Power Locks
Power Windows
Air-conditioner
Extended Insurance:
3 years
1 year
None
Description:
Price: $18500
Reset Price
Exit

4 General Hints of Some GUI components

4.1 Choice Boxes

4.1.1 Defining the choice box in your Java code

Choice boxes are drop down lists. The list can be of anything: simple Strings, images, checkboxes, links, etc. The choice boxes can handle almost anything you put in them.

Scene Builder does not know what kind of items your program will place in them and because of that Scene Builder suggests these lines to be placed in your Java code:

```
@FXML
private ChoiceBox<?> carTypeCB;
```

You must replace the ? with the type of item you are going to place in this choice box. In our case, it will be a String, i.e.,

```
@FXML
private ChoiceBox<String> carTypeCB;
```

Note: carTypeCB is just a variable name, it could be anything here.

4.1.2 Populating the choice box

Since the contents of choice boxes can change during run-time (such as the available list of colors depends on the chosen car), you must add the items to the choice box from your Java code, not from the Scene Builder.

To do so, you should do something like this:

```
String[] items = {"Corolla", "Yaris", "Rav4"};
carTypeCB.setItems(FXCollections.observableArrayList(items));
```

4.1.3 Pre-selecting the first item in the choice box

```
carColorCB.getSelectionModel().selectFirst();
```

4.1.4 Listening to changes in the choice box selection

Two common situations can happen in a program that uses a GUI as the main interface between the user and the code logic:

- A program can respond immediately after a change occurs in the GUI, such as the user just picked a different font type and your text change right away.
- The program can wait all the data is entered in the GUI and only when the SUBMIT button is pressed is when your program looks at the selections that the user has chosen.

To have your program to immediately respond to a chance in a choice box (or any other GUI component, such as check box or radio button), you must create an action listener to the component. The general idea to add a listener to a choice box is given below (for other GUI components it may change a little bit from this structure):

```
carTypeCB.getSelectionModel().selectedIndexProperty().addListener(listenerObject);
```

Where listenerObject is an object of a Class MyListener which you define in your project and it must implement the Java Interface named ChangeListener.

There are three ways of creating this class and object. Let's look at them:

4.1.4.1 Creating a Class

In NetBeans you can create a Java Class using the File >> New >> Java Class, and naming the file as anything you want, such as MyChoiceBox1Listener.

This class must implements the interface ChangeListener (it is like an abstract method, that you must add code into it in your new class).

Example of that is given below:

```
public class MyChoiceBox1Listener implements ChangeListener {

    @Override
    public void changed(ObservableValue observable, Object oldValue, Object newValue)
{
        System.out.println("Hello There");
        // perform some activity here…
    }
}
```

And with that Class created you can create an object of such class in your code and use it as shown below:

```
MyChoiceBox1Listener myListener;
carTypeCB.getSelectionModel().selectedIndexProperty().addListener(myListener);
```

or

```
carTypeCB.getSelectionModel().selectedIndexProperty().addListener(new
MyChoiceBox1Listener);
```

both ways work exactly the same.

You can also create what we call inner class in your code, instead of creating a separate file `MyChoiceBox1Listener` in your project. To do so, just before the last } in your Java file that has the choice box defined, place the same contents that you would place in the separate file but now inside the existing Java.

4.1.4.2 Anonymous Class

Instead of creating a separate Class, or an inner class so that we can create an object of that to be given to the addListener method as an input, we can do what is called anonymous class, where we defined the contents of the listener class inside the ( ) where the object should be placed. An example is shown below:

```
carTypeCB.getSelectionModel().selectedIndexProperty().addListener(
        new ChangeListener<Number>() {
    @Override
    public void changed(ObservableValue ov, Number value, Number newValue) {
        // perform some activity here…
    }
});
```

4.1.5 Getting the selection of a choice box

You can get the selected value of a choice box by placing the following command inside the *changed* method shown above.

```
String selectedCar = (String) carTypeCB.getItems().get(newValue.intValue());
```

Or outside the `changed` method:

```
String selectedItem = carColorCB.getSelectionModel().getSelectedItem();
```

4.2 Radio Buttons

To check if the button is selected:

```
boolean value = radioButtonName.isSelected();
```

To set a radio button to selected:

```
radioButtonName.setSelected(true);
```

4.3 Check Boxes

Checking if the check box is selected:

```
checkBoxName.isSelected()
```

To enable or disable a check box:

```
checkBoxName.setDisable(true/false);
```

4.4 ImageView

To display an image in a GUI that is stored in a file, the following code will open that file, load the image and place the image in the ImageView component in your GUI:

```
File file = new File(filename);
Image image = new Image(file.toURI().toString());
imageViewFxid.setImage(image);
imageViewFxid.autosize();
```

4.5 TextArea

To add a text to the text area, I would advise you to create a StringBuilder first, add all the strings you want in it and then place it in the TextArea component as shown below:

```
textAreaFxid.setText(sb.toString());
```

5 The Car-Related Classes

Your GUI application handles the following types of cars: Corolla, Rav4, and Yaris. All of them have the following features:

```
carType;  // Corolla, Rav4,or Yaris
availableColors;  // it depends on the car type (*)
base price;  // it depends on the car type (*)
standard Accessories;  // it depends on the car type (*)
optional Accessories; // it depends on the car type (*)
description;  // it depends on the car type (*)
car Picture Path;
```

(*) please watch the video to figure out what values are allowed for each car type.

5.1 The Super Class Car

Create a super class called Car and place the definition of all the features above as class fields.

5.2 The Corolla, Rav4, and Yaris Classes

These classes shall inherit from the Car Class. For this project, these classes will have the minimum of code as possible. Just set the values of the class fields defined in the super class Car, for each type of car.

Hint: Use an object of the super class Car instead of specific instances of the Corolla, Rav4, or Yaris when interacting with the GUI (Polymorphism).

Example:

```
Car selectedCar;
if (carType.equalsIgnoreCase("Corolla")) {
    selectedCar = new Corolla();
} else if (carType.equalsIgnoreCase("Yaris")) {
    selectedCar = new Yaris();
} else if (carType.equalsIgnoreCase("Rav4")) {
    selectedCar = new Rav4();
}
```

And then use selectedCar object to get the optionals, price, standard features, picture, etc to adjust the GUI components.

Good luck.