MIS 768: Advanced Software Concepts

Spring 2024

GUI Application (3)

Purpose

- Learn to use various controls in JavaFX application
- Set up for a user-friendly GUI application
- Create multiple-scene applications

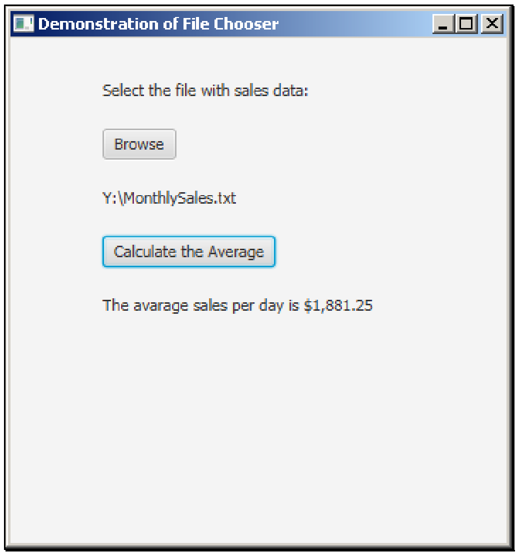
1. Preparation

(1) Launch Eclipse. Create a new package to hold our source file. Name the package as **edu.unlv.mis768.labwork16.**

(2) Download **16_lab_files.zip** from WebCampus. Extract the zip file and then import the .java files into the package.

2. File Chooser

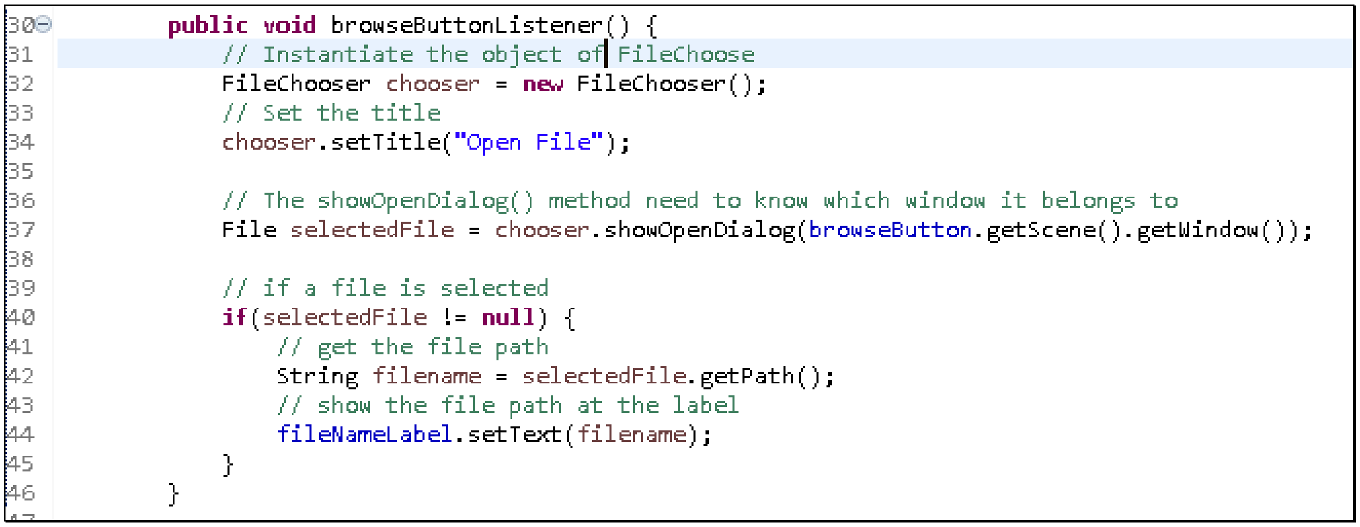
(3) In this application, the user will select a file with sales data and the program calculates the average sales.


Demonstration of File Chooser
Select the file with sales data:
Browse
Y:\MonthlySales.txt
Calculate the Average
The avarage sales per day is $1,881.25

(4) Please open **ReadFile.fxml** in Scene Builder. A few components have been added with fx:id set.

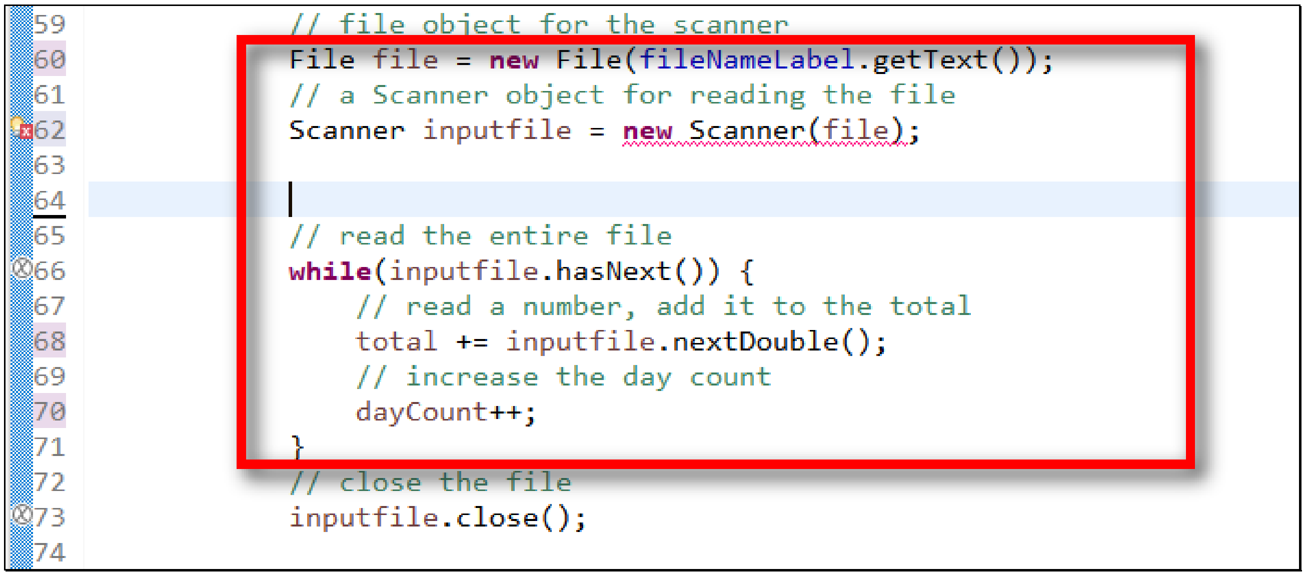
(5) Open **ReadFileController.java** in Eclipse.

There are two Listener methods for the two buttons. **browseButtonListener**() will use the **FileChoose** object. Complete the code as following



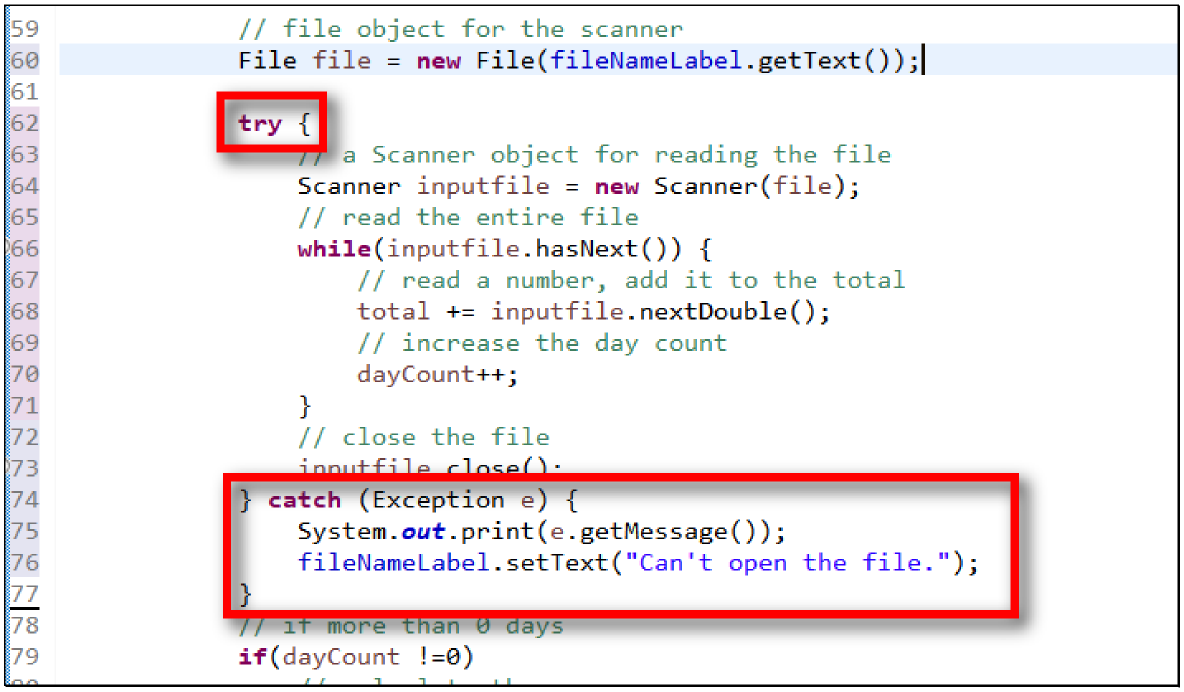
```
        public void browseButtonListener() {
            // Instantiate the object of FileChoose
            FileChooser chooser = new FileChooser();
            // Set the title
            chooser.setTitle("Open File");

            // The showOpenDialog() method need to know which window it belongs to
            File selectedFile = chooser.showOpenDialog(browseButton.getScene().getWindow());

            // if a file is selected
            if(selectedFile != null) {
                // get the file path
                String filename = selectedFile.getPath();
                // show the file path at the label
                fileNameLabel.setText(filename);
            }
        }
```

(6) Please also complete **calcButtonListener**().



```
        // file object for the scanner
        File file = new File(fileNameLabel.getText());
        // a Scanner object for reading the file
        Scanner inputfile = new Scanner(file);


        // read the entire file
        while(inputfile.hasNext()) {
            // read a number, add it to the total
            total += inputfile.nextDouble();
            // increase the day count
            dayCount++;
        }
        // close the file
        inputfile.close();
```

(7) At opening the file, you can either add **throws IOException** at the header of the method, or use try/catch clause.



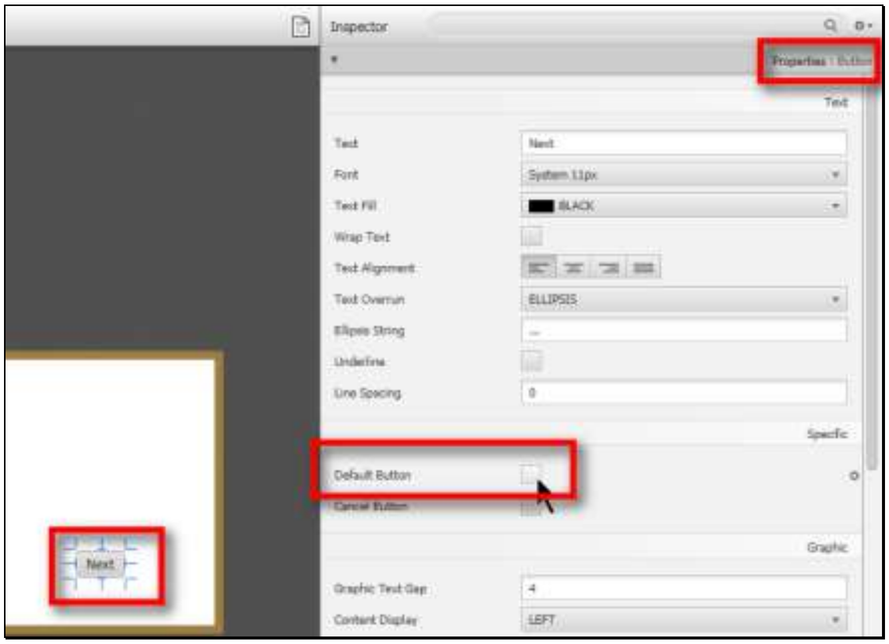
```
        // file object for the scanner
        File file = new File(fileNameLabel.getText());

        try {
            // a Scanner object for reading the file
            Scanner inputfile = new Scanner(file);
            // read the entire file
            while(inputfile.hasNext()) {
                // read a number, add it to the total
                total += inputfile.nextDouble();
                // increase the day count
                dayCount++;
            }
            // close the file
            inputfile.close();
        } catch (Exception e) {
            System.out.print(e.getMessage());
            fileNameLabel.setText("Can't open the file.");
        }
        // if more than 0 days
        if(dayCount !=0)
```

(8) Run **ReadFile.java** to see the result. (Note: to test this program, you can use the salesNumbers.txt or create you own file with some sales to test.)

3. Default Button and Mnemonic

(9) Please open **SandwichMenu.fxml** in **SceneBuilder**.

(10) Click on **nextButton**. Select **Properties** panel and then check the **Default Button** option. By doing so, we specify the button as the default for this scene.



Inspector

Properties : Button

Text

Text: Next
Font: System 11px
Text Fill: BLACK
Wrap Text
Text Alignment
Text Overrun: ELLIPSIS
Ellipsis String: ...
Underline
Line Spacing: 0

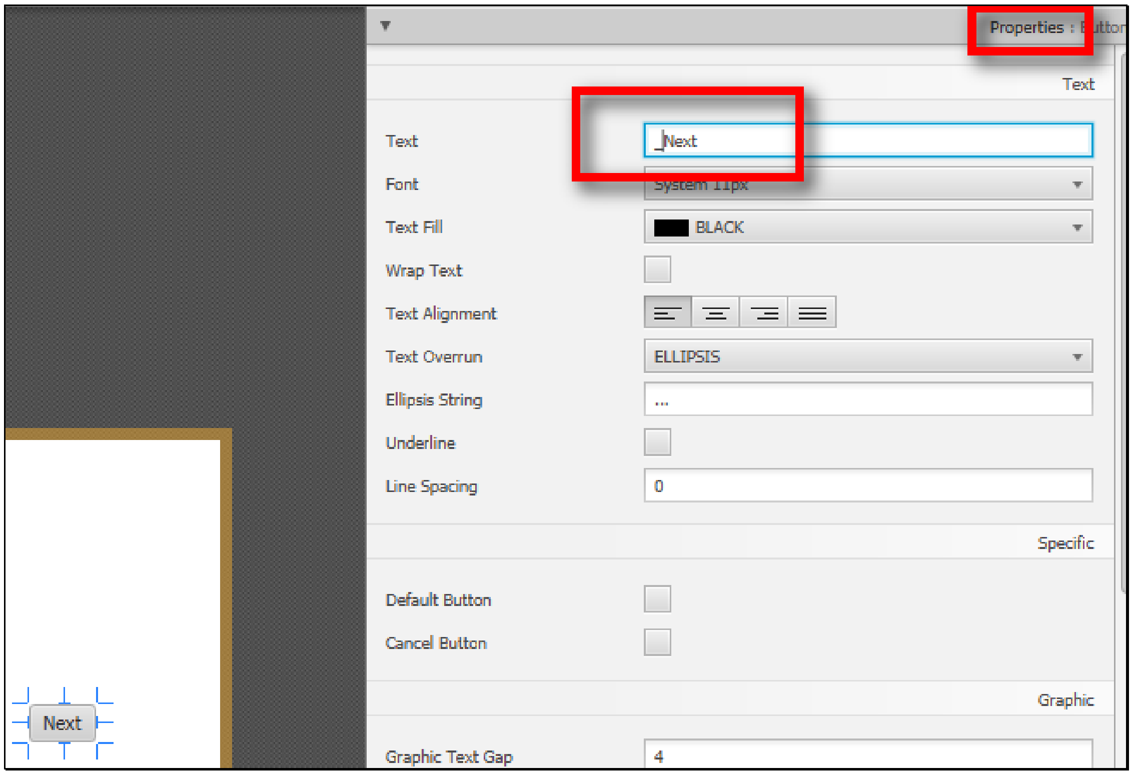
Specific

Default Button
Cancel Button

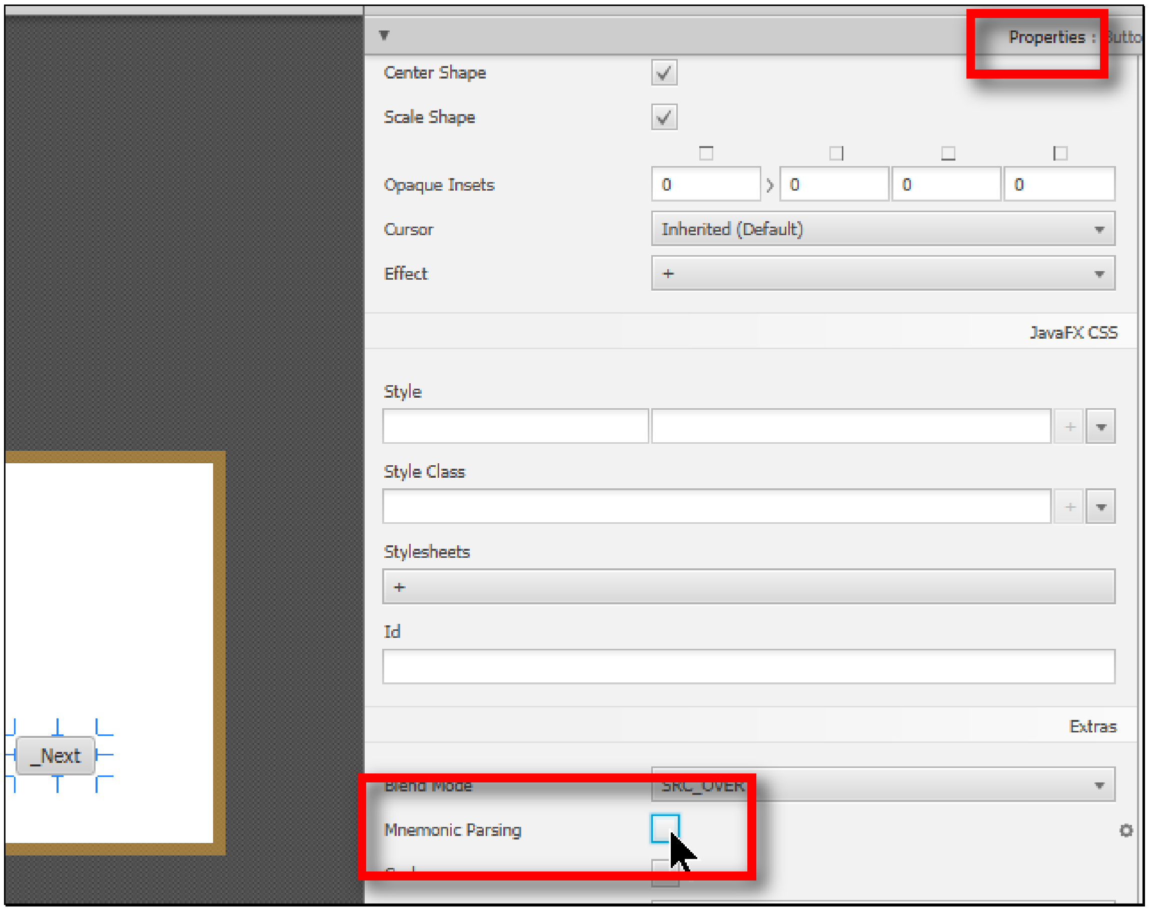
Graphic

Graphic Text Gap: 4
Content Display: LEFT

(11) Edit the Text of the **nextButton** to add a _ (underscore). By doing so, we specify “N” as the keyboard shortcut for this button as Alt+N.



(12) At the same panel, scroll down check the **Mnemonic Parsing** option. By doing so, the underscore will be parsed as a mnemonic specification.



(13) Please check the three radio buttons on the same scene. They have been set up with mnemonics.

(14) Save and close **SandwichMenu.fxml**.

(15) Please also open **SideMenu.fxml** in **SceneBuilder**.

Please specify the mnemonics for the two buttons and choose one to be the default button.

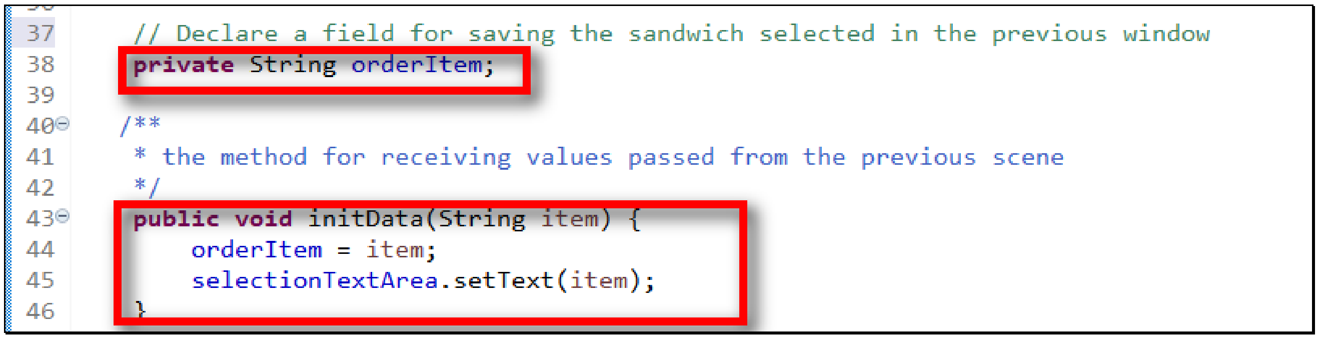
4. Multiple Scene Application

(16) In the Burger Joint Application, the application will start with the selection of the sandwich. It will be pass to the second scene.

(17) Please open **SideMenuController.java**.

Declare a field for this class, representing the item ordered on the previous page.

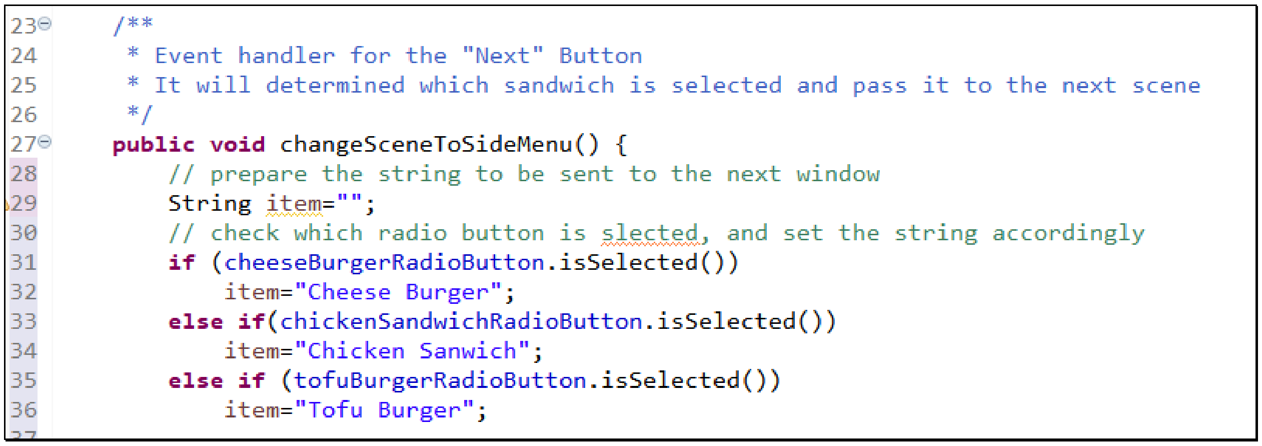
Then add a method, accepting a String variable as the parameter.



```java
     // Declare a field for saving the sandwich selected in the previous window
     private String orderItem;

    /**
     * the method for receiving values passed from the previous scene
     */
     public void initData(String item) {
         orderItem = item;
         selectionTextArea.setText(item);
     }
```

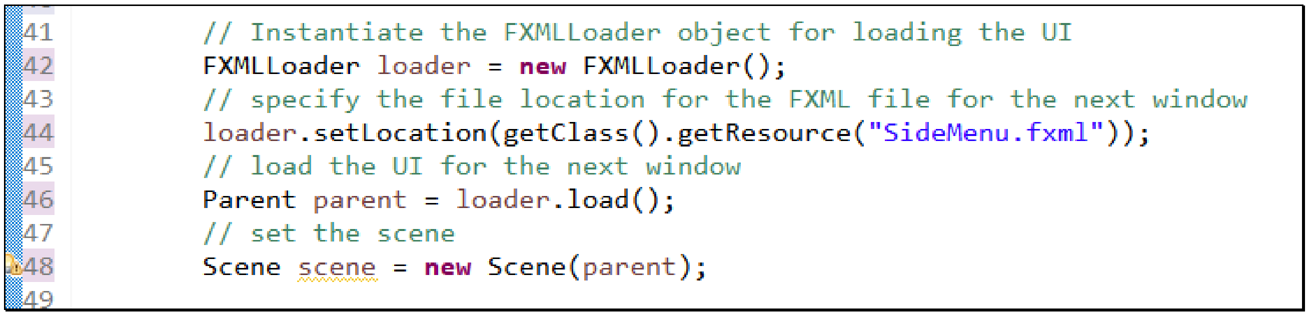
(18) Open **SandwichMenuController.java**

Implement the listener for the button. First prepare the string to be passed.



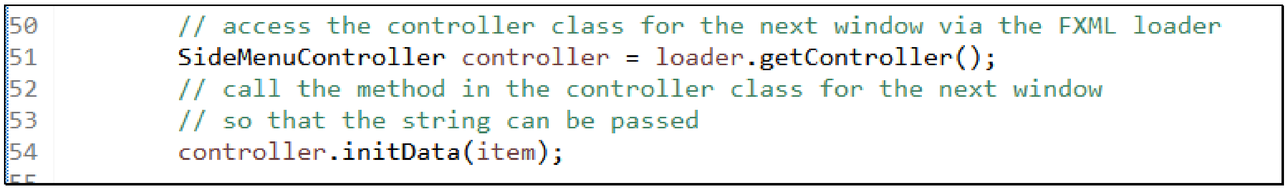
```java
    /**
     * Event handler for the "Next" Button
     * It will determined which sandwich is selected and pass it to the next scene
     */
    public void changeSceneToSideMenu() {
        // prepare the string to be sent to the next window
        String item="";
        // check which radio button is slected, and set the string accordingly
        if (cheeseBurgerRadioButton.isSelected())
            item="Cheese Burger";
        else if(chickenSandwichRadioButton.isSelected())
            item="Chicken Sanwich";
        else if (tofuBurgerRadioButton.isSelected())
            item="Tofu Burger";
```

(19) Prepare the scene for the next window; that is, load the UI for the next window.



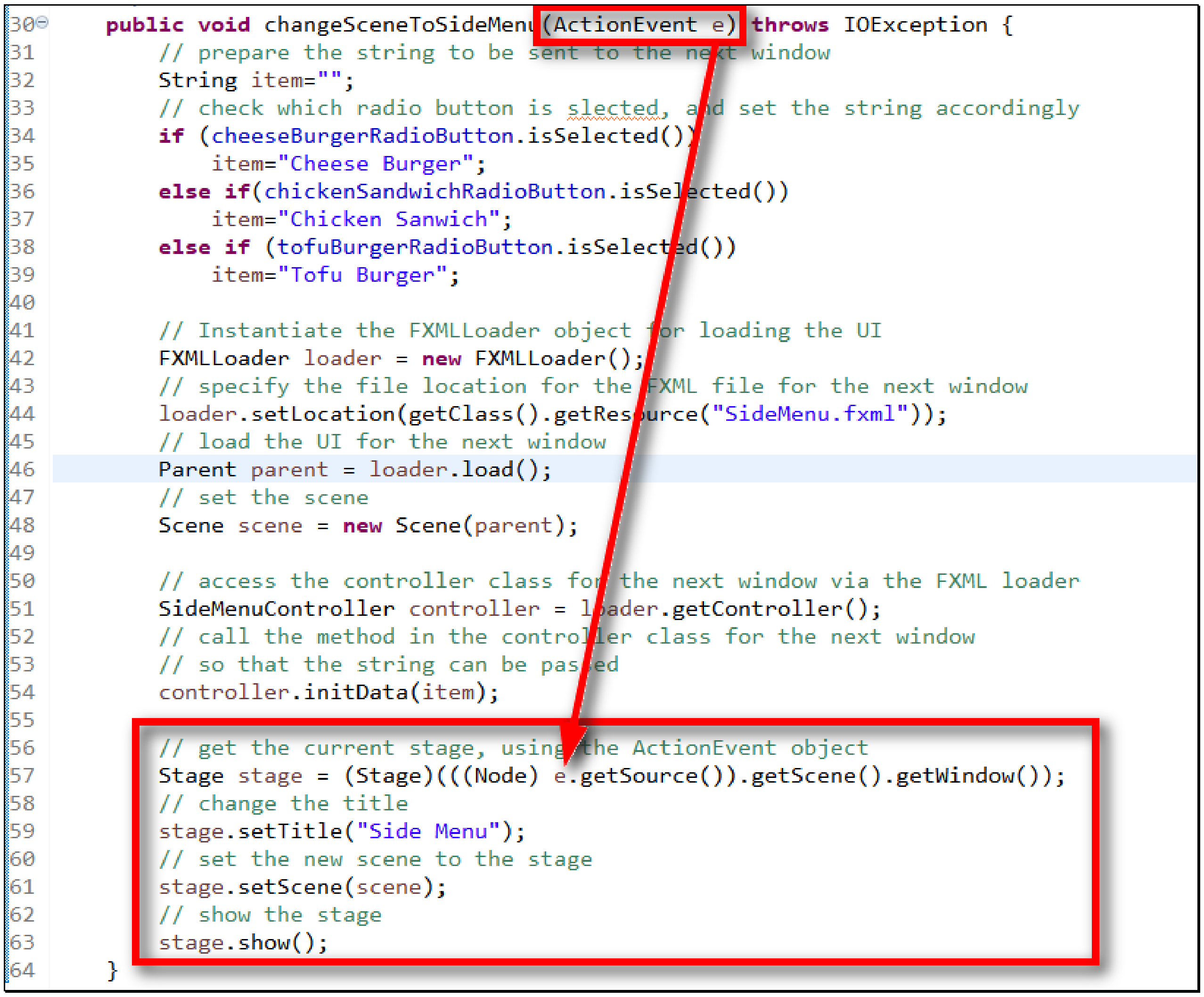
```java
        // Instantiate the FXMLLoader object for loading the UI
        FXMLLoader loader = new FXMLLoader();
        // specify the file location for the FXML file for the next window
        loader.setLocation(getClass().getResource("SideMenu.fxml"));
        // load the UI for the next window
        Parent parent = loader.load();
        // set the scene
        Scene scene = new Scene(parent);
```

(20) Call the method of the controller class for the next window. Pass the item String to the method.



```
        // access the controller class for the next window via the FXML loader
        SideMenuController controller = loader.getController();
        // call the method in the controller class for the next window
        // so that the string can be passed
        controller.initData(item);
```

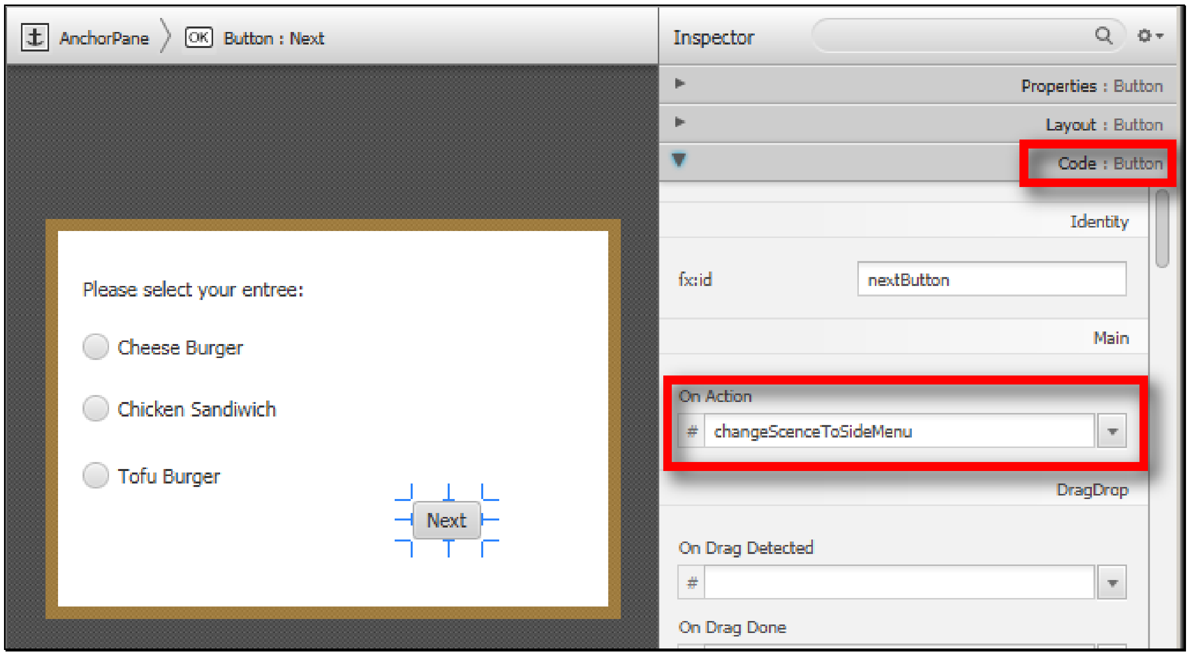
(21) After setting up the scene, make it appear to the stage (i.e., window).



```
    public void changeSceneToSideMenu(ActionEvent e) throws IOException {
        // prepare the string to be sent to the next window
        String item="";
        // check which radio button is slected, and set the string accordingly
        if (cheeseBurgerRadioButton.isSelected())
            item="Cheese Burger";
        else if(chickenSandwichRadioButton.isSelected())
            item="Chicken Sanwich";
        else if (tofuBurgerRadioButton.isSelected())
            item="Tofu Burger";

        // Instantiate the FXMLLoader object for loading the UI
        FXMLLoader loader = new FXMLLoader();
        // specify the file location for the FXML file for the next window
        loader.setLocation(getClass().getResource("SideMenu.fxml"));
        // load the UI for the next window
        Parent parent = loader.load();
        // set the scene
        Scene scene = new Scene(parent);

        // access the controller class for the next window via the FXML loader
        SideMenuController controller = loader.getController();
        // call the method in the controller class for the next window
        // so that the string can be passed
        controller.initData(item);

        // get the current stage, using the ActionEvent object
        Stage stage = (Stage)(((Node) e.getSource()).getScene().getWindow());
        // change the title
        stage.setTitle("Side Menu");
        // set the new scene to the stage
        stage.setScene(scene);
        // show the stage
        stage.show();
    }
```

(22) Open **SandwichMenu.fxml** in **SceneBuilder**.

Specify the listener of **nextButton** as **changeScenceToSideMenu**().



fx:id: nextButton

On Action: # changeScenceToSideMenu

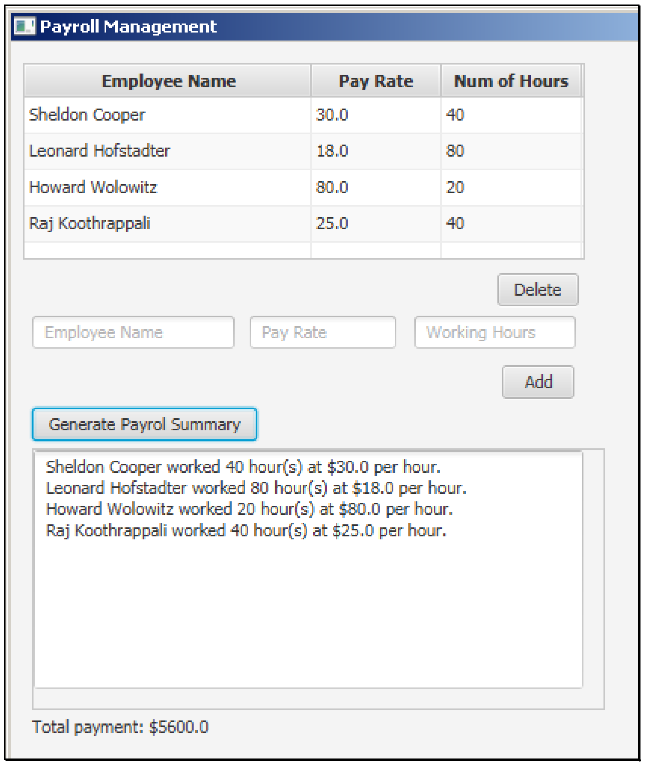
(23) Save and close the file.

(24) You can run **BurgerJoint.java** to test the program.

(25) Please also check the **startOverButtonListener**() method in **SideMenuController.** It switches the scene back to the first window.

5. TableView and TableColumn

(26) In this example, we will create an application that manages the payroll for an organization.



Employee Name	Pay Rate	Num of Hours
Sheldon Cooper	30.0	40
Leonard Hofstadter	18.0	80
Howard Wolowitz	80.0	20
Raj Koothrappali	25.0	40

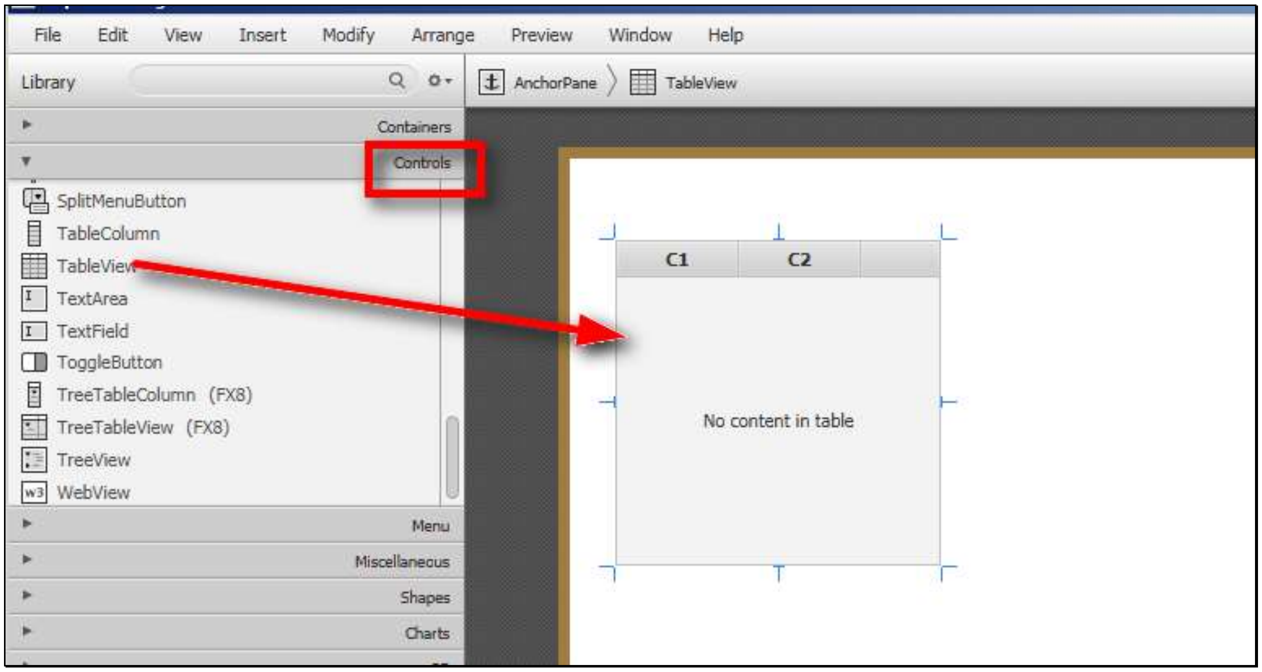
```
Sheldon Cooper worked 40 hour(s) at $30.0 per hour.
Leonard Hofstadter worked 80 hour(s) at $18.0 per hour.
Howard Wolowitz worked 20 hour(s) at $80.0 per hour.
Raj Koothrappali worked 40 hour(s) at $25.0 per hour.
```

Total payment: $5600.0

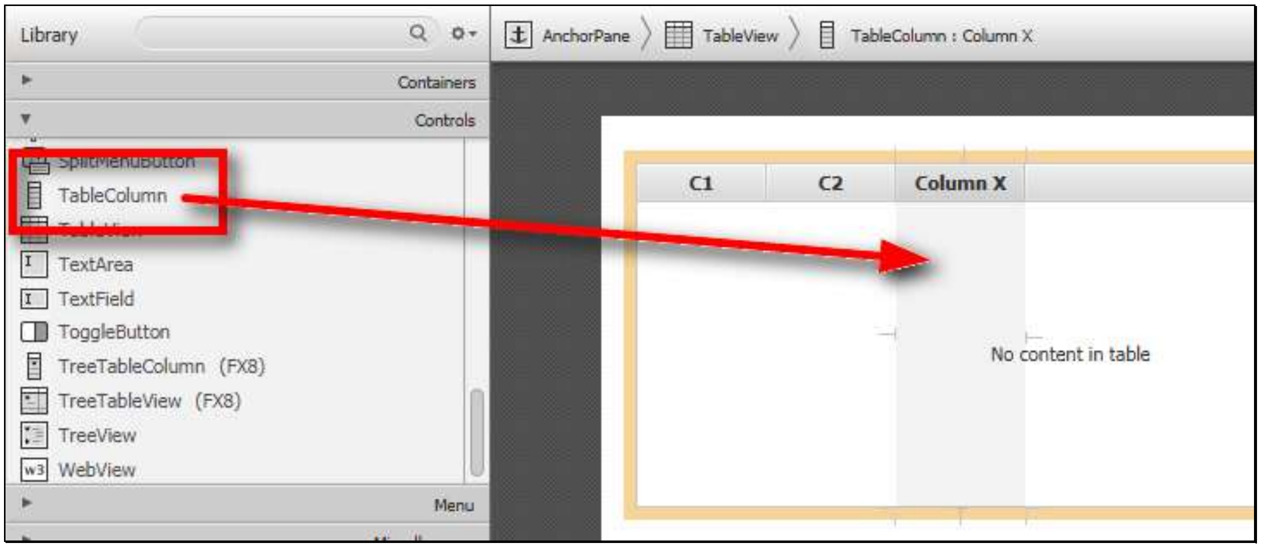
(27) Please open **Payroll.java**. It is the class representing the data model. It has three columns: empName, payRate, numOfHours, and the associated constructors and get and set methods.

(28) Please open **PayrollManagement.fxml** in **SceneBuilder**.

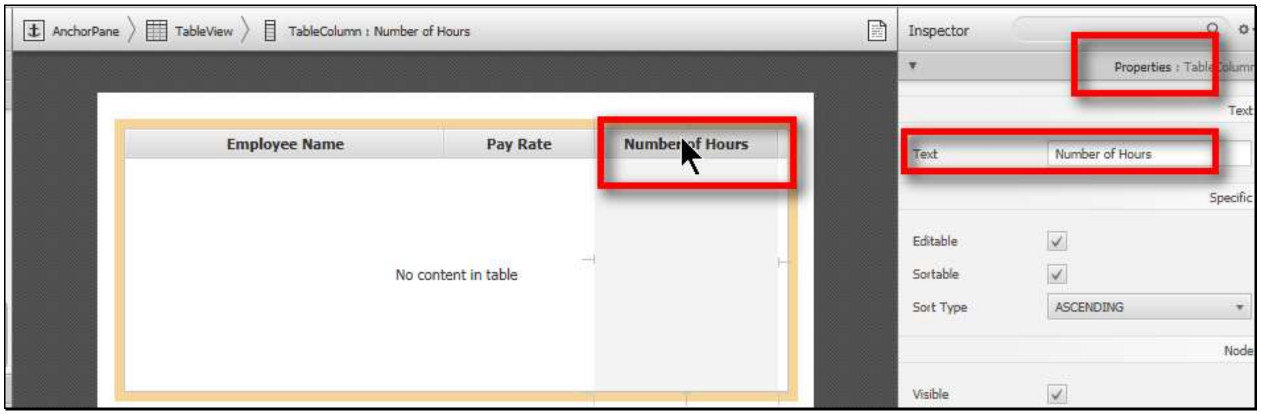
(29) Drag a **TableView** from the **Controls** panel to the pane.



(30) By default, there only two **TablColumn** objects added. Please add one more column by draggin a **TableColumn** control into the **TableView**.



(31) You can resize the column and change the text of each column to show Employee Name, Pay Rate, and Number of Hours respectively.



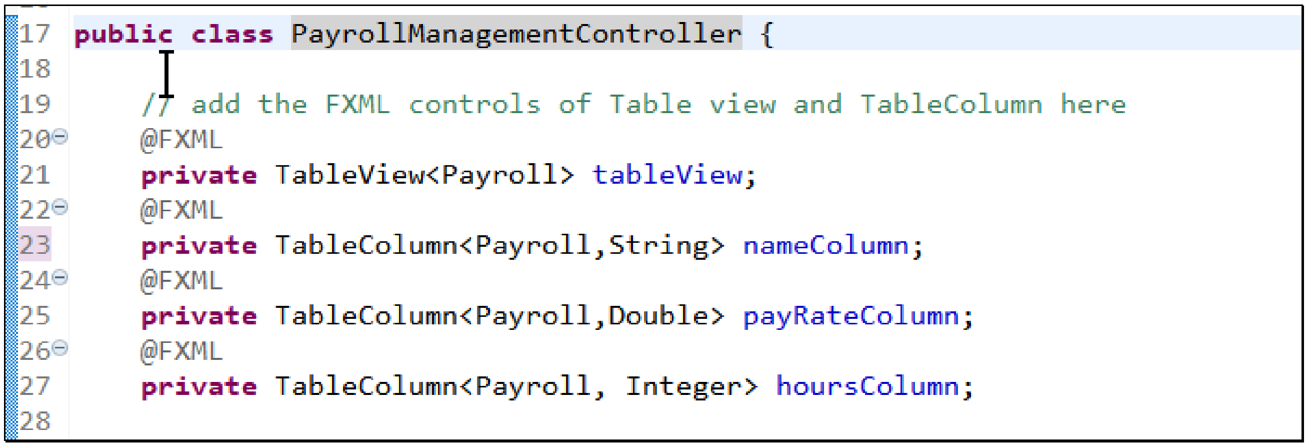
Text: Number of Hours
Sort Type: ASCENDING

(32) Please assign the fx:id to the TableView as **tableView**.

Assign the fx:id to each of the columns as **nameColumn**, **rateColumn**, and **hoursColumn**.

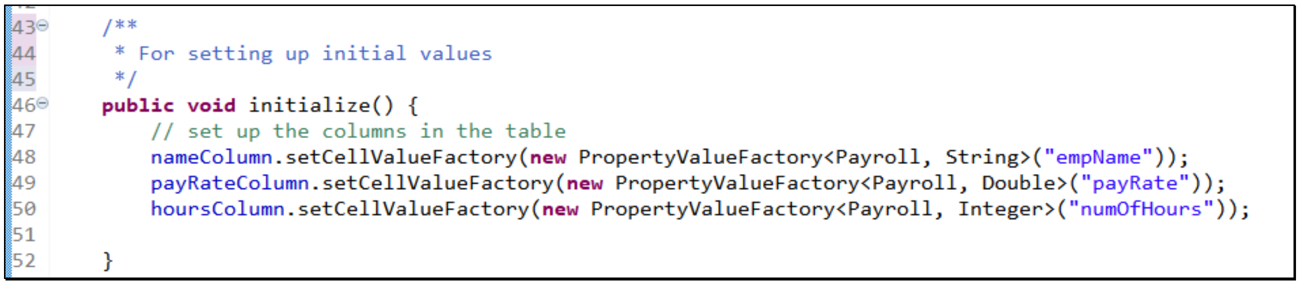
(33) Please open **PayrollManagementController.java** and add the following definition of the **TableView** and **TableColumns**.

By doing so, we defind the columns as represengint the fields of the class, with respective data type.



```
public class PayrollManagementController {

    // add the FXML controls of Table view and TableColumn here
    @FXML
    private TableView<Payroll> tableView;
    @FXML
    private TableColumn<Payroll,String> nameColumn;
    @FXML
    private TableColumn<Payroll,Double> payRateColumn;
    @FXML
    private TableColumn<Payroll, Integer> hoursColumn;

```

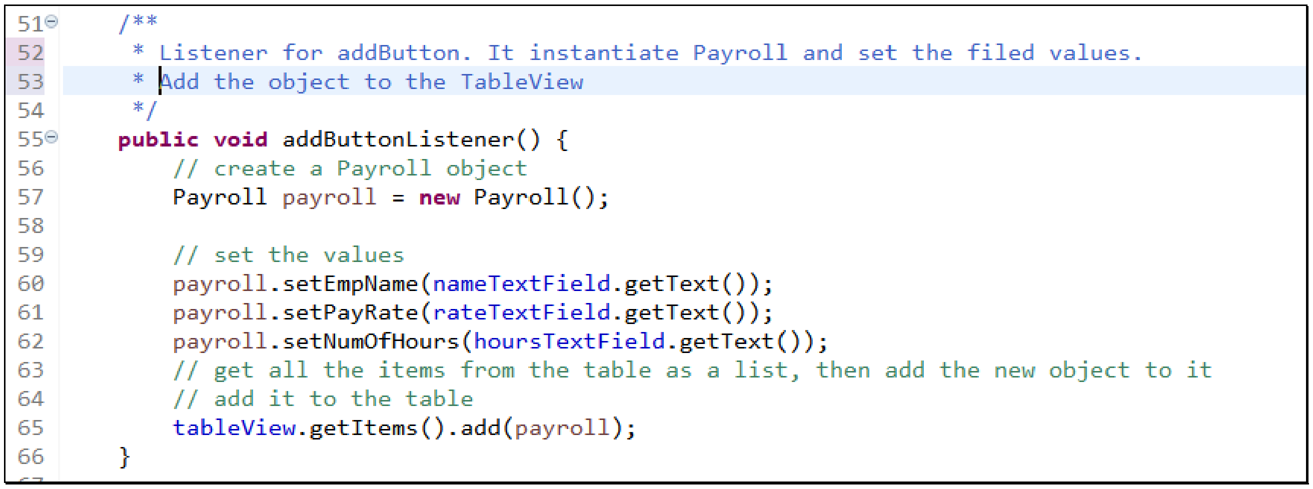
(34) Edit the **initialize()** method to add the following lines. By doing so, the data is associated with the table columns.



```
    /**
     * For setting up initial values
     */
    public void initialize() {
        // set up the columns in the table
        nameColumn.setCellValueFactory(new PropertyValueFactory<Payroll, String>("empName"));
        payRateColumn.setCellValueFactory(new PropertyValueFactory<Payroll, Double>("payRate"));
        hoursColumn.setCellValueFactory(new PropertyValueFactory<Payroll, Integer>("numOfHours"));

    }
```

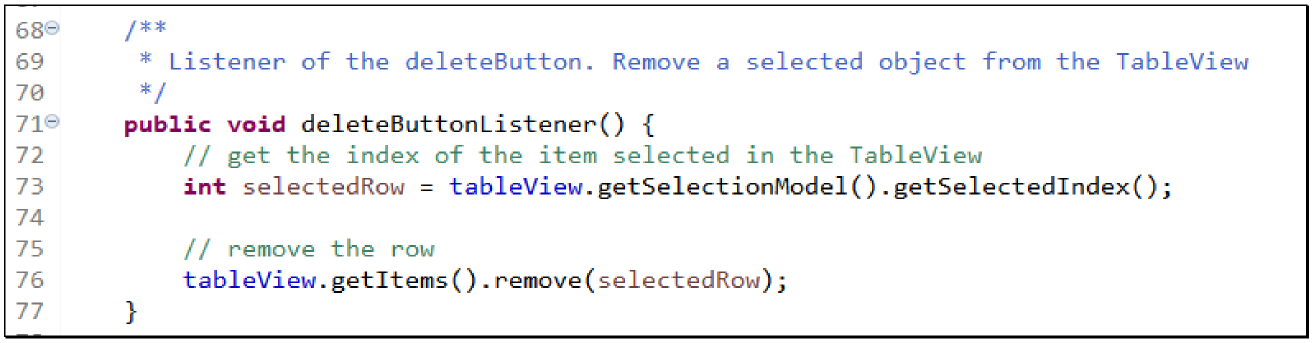
(35) You can now run the **PayrollManagement.java** application but we do not have any data in the table yet.

(36) Please implment the **addButtonListener()** method. It instantiates a **Payroll** object, use the value in the TextFields to set the values. Then the object is added to the **ObservableList** array.



```
    /**
     * Listener for addButton. It instantiate Payroll and set the filed values.
     * Add the object to the TableView
     */
    public void addButtonListener() {
        // create a Payroll object
        Payroll payroll = new Payroll();

        // set the values
        payroll.setEmpName(nameTextField.getText());
        payroll.setPayRate(rateTextField.getText());
        payroll.setNumOfHours(hoursTextField.getText());
        // get all the items from the table as a list, then add the new object to it
        // add it to the table
        tableView.getItems().add(payroll);
    }
```

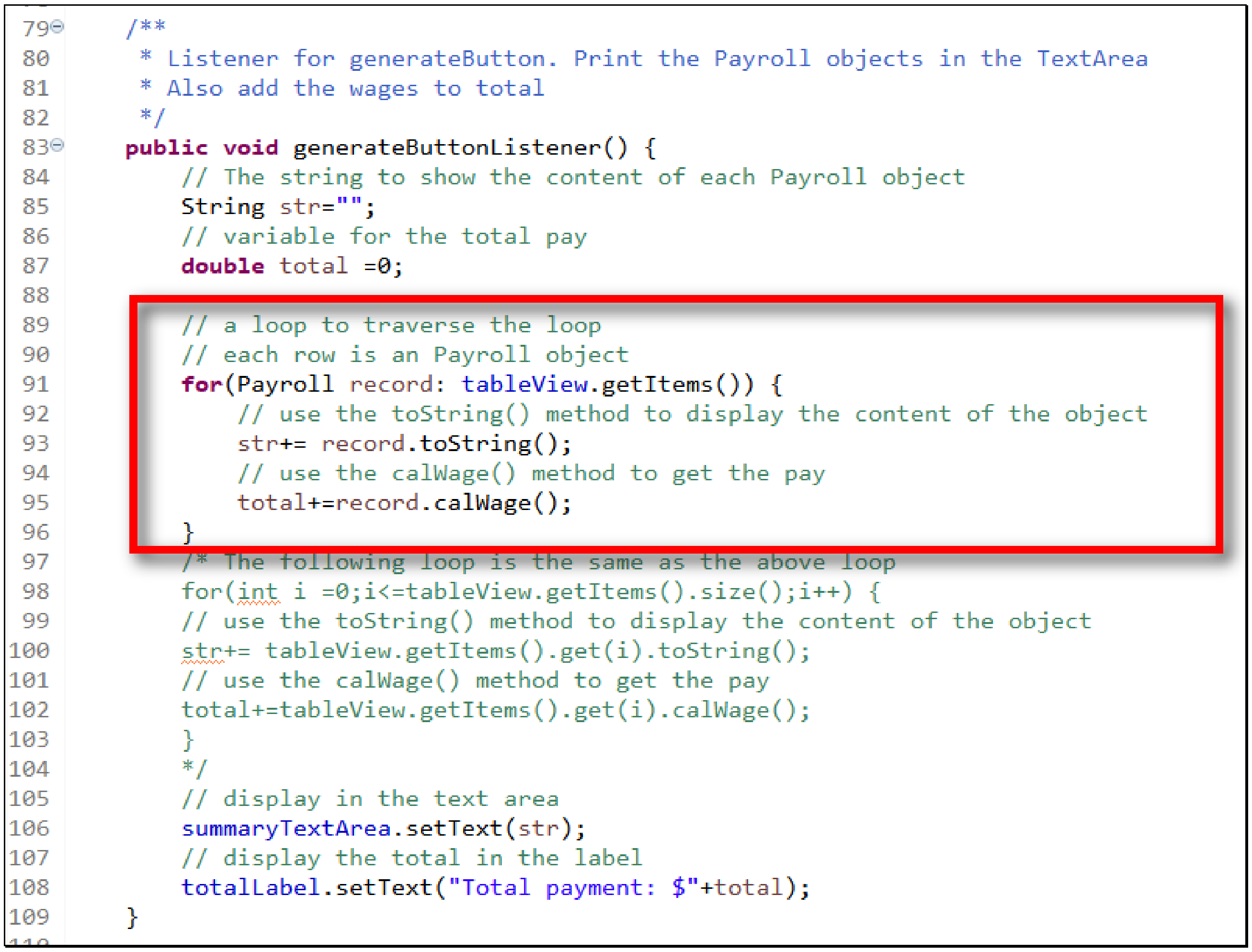
(37) Please implement the **deleteButtonListener()** method to remove a selected record from the TableView.



```
    /**
     * Listener of the deleteButton. Remove a selected object from the TableView
     */
    public void deleteButtonListener() {
        // get the index of the item selected in the TableView
        int selectedRow = tableView.getSelectionModel().getSelectedIndex();

        // remove the row
        tableView.getItems().remove(selectedRow);
    }
```

(38) The **generateButtonListener()** use a loop to traverse all the rows. For each row/object, the program calls the **calWage()** and **toString()** method of **Payroll** class to calculate the pay.

You can also use a regular for loop in this method.



```
    /**
     * Listener for generateButton. Print the Payroll objects in the TextArea
     * Also add the wages to total
     */
    public void generateButtonListener() {
        // The string to show the content of each Payroll object
        String str="";
        // variable for the total pay
        double total =0;

        // a loop to traverse the loop
        // each row is an Payroll object
        for(Payroll record: tableView.getItems()) {
            // use the toString() method to display the content of the object
            str+= record.toString();
            // use the calWage() method to get the pay
            total+=record.calWage();
        }
        /* The following loop is the same as the above loop
        for(int i =0;i<=tableView.getItems().size();i++) {
        // use the toString() method to display the content of the object
        str+= tableView.getItems().get(i).toString();
        // use the calWage() method to get the pay
        total+=tableView.getItems().get(i).calWage();
        }
        */
        // display in the text area
        summaryTextArea.setText(str);
        // display the total in the label
        totalLabel.setText("Total payment: $"+total);
    }
```

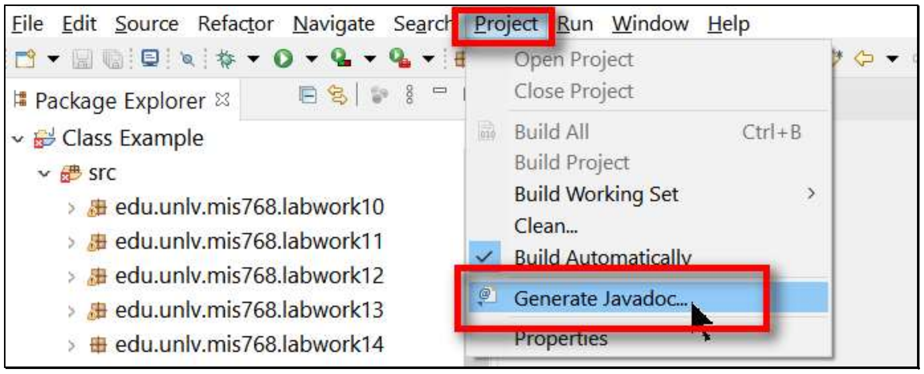
(39) The listeners are linked to the buttons already. You can run **PayrollManagement.java** application to test the program.

6. Generate Javadoc

(40) Please open **RetailItem.java** and **SavingsAccount.java** to see the comments in JavaDoc format.

(41) Select your project that contains the packages.

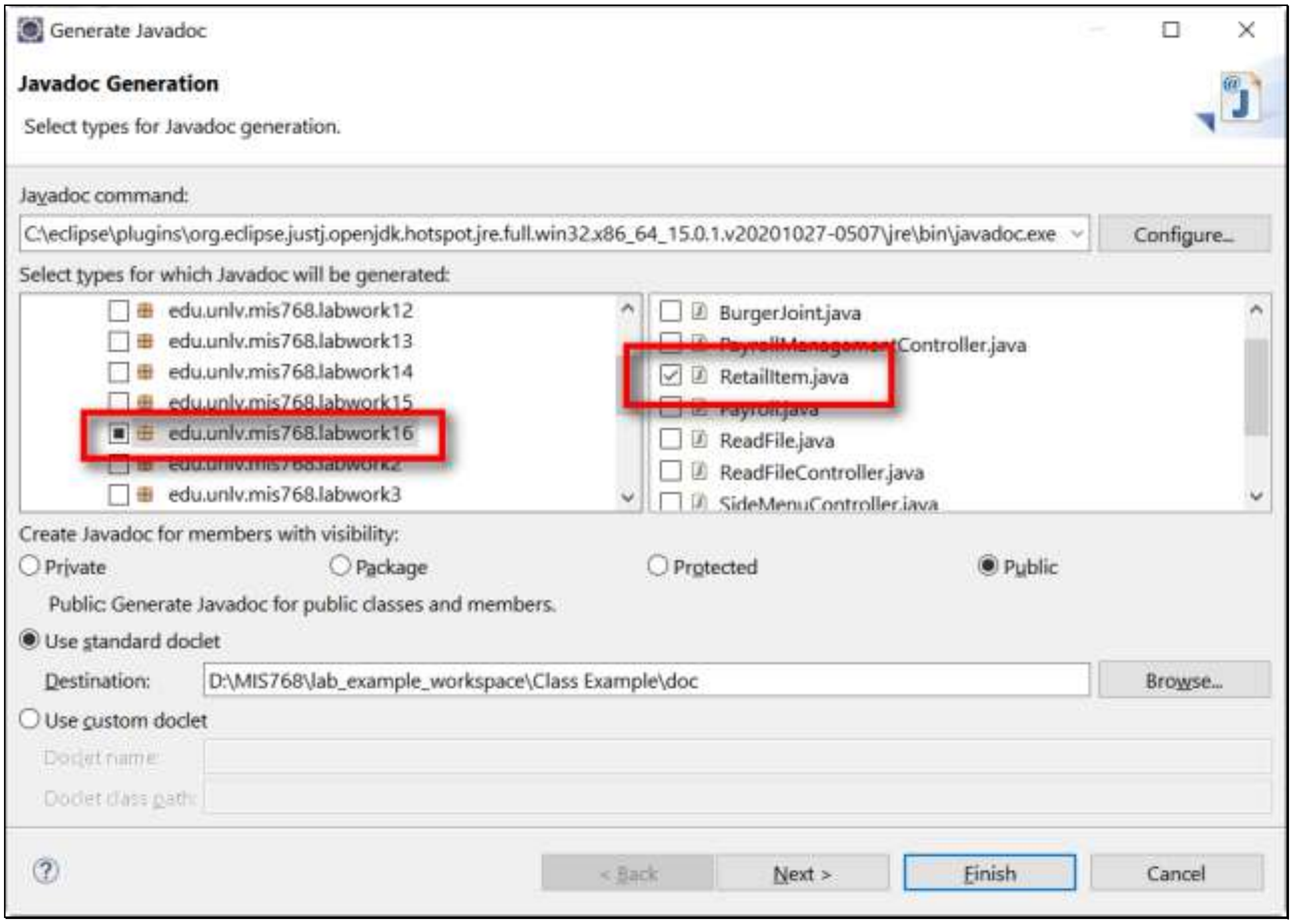
Select **Project \ Generate Javadoc ..** from the menu.


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Class Example
src
edu.unlv.mis768.labwork10
edu.unlv.mis768.labwork11
edu.unlv.mis768.labwork12
edu.unlv.mis768.labwork13
edu.unlv.mis768.labwork14
Open Project
Close Project
Build All Ctrl+B
Build Project
Build Working Set
Clean...
Build Automatically
Generate Javadoc...
Properties

(42) At the **Javadoc commend**: You can use the default path or click the **Configure…** button to select the **javadoc.exe** file (Under the directory the JDK is installed to.)

Make sure you check the package you need for generating the Javadoc.

Select the destination of the generated documents. Click **Finish** button.


Generate Javadoc

Javadoc Generation

Select types for Javadoc generation.

Javadoc command:

C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-0507\jre\bin\javadoc.exe

Select types for which Javadoc will be generated:

- [ ] edu.unlv.mis768.labwork12
- [ ] edu.unlv.mis768.labwork13
- [ ] edu.unlv.mis768.labwork14
- [ ] edu.unlv.mis768.labwork15
- [■] edu.unlv.mis768.labwork16
- [ ] edu.unlv.mis768.labwork2
- [ ] edu.unlv.mis768.labwork3

- [ ] BurgerJoint.java
- [ ] PayrollManagementController.java
- [✓] RetailItem.java
- [ ] Payroll.java
- [ ] ReadFile.java
- [ ] ReadFileController.java
- [ ] SideMenuController.java

Create Javadoc for members with visibility:

○ Private ○ Package ○ Protected ● Public

Public: Generate Javadoc for public classes and members.

● Use standard doclet

Destination: D:\MIS768\lab_example_workspace\Class Example\doc

○ Use custom doclet

Doclet name:

Doclet class path:

(43) After the documents are generated, you can go to your project folder to find the documents. Double click **index.html.**

(44) Please find **SavingsAccount** class on the HTML page, and click it.

(45) Now you can see the detailed description of the class, including the version, author, and description for each method.