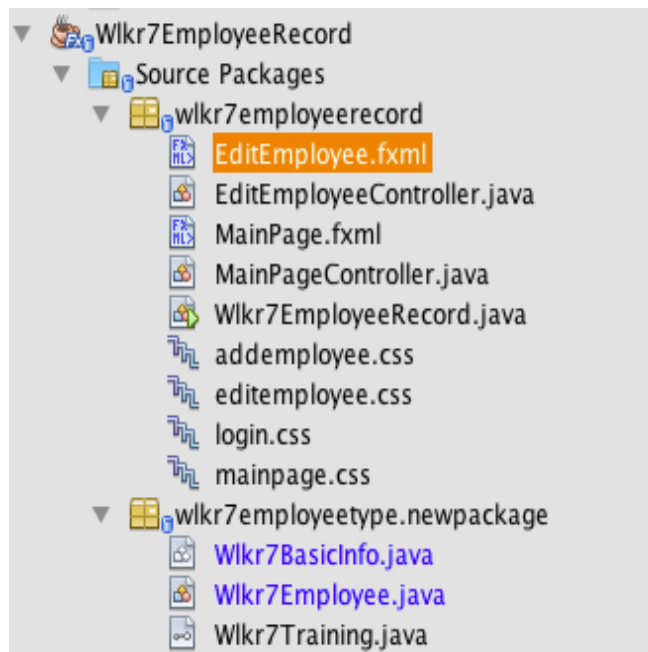


Documentation for Wlkr7EmployeeRecord

This JavaFX Application is made by Wei Xian Low (PawPrint: WLKR7) for the CS3330 Final Project.

This application has utilized multiple classes, which include, an abstract class, a subclass, and an interface.



Based on the snapshot on the left, the abstract class, subclass, and the interface is located inside the package called: "wlkr7employeetype.newpackage"

The java file named "Wlkr7BasicInfo.java" is the abstract class.

The java file named "Wlkr7Employee.java" is the subclass implementing the abstract class "Wlkr7BasicInfo.java"

The java file named "Wlkr7Training.java" is the interface class where it's implemented by the file "Wlkr7Employee.java"

-Object Oriented Elements

```
public abstract class Wlkr7BasicInfo {

    private StringProperty firstName;
    private StringProperty lastName;

    private StringProperty address;
    private StringProperty phoneNumber;

    private StringProperty employeeID;
    private Boolean morts;
    private Boolean doMundos;
    private Boolean infusion;
    private Boolean kateAndEmmas;
    private Boolean pomodoros;
    private Boolean employee;
    private Boolean supervisor;
    private Boolean manager;
}
```

The above picture shows a snapshot of codes contained in the file "Wlkr7BasicInfo.java". This shows that the class is an abstract class.

```

public class Wlkr7Employee extends Wlkr7BasicInfo implements Wlkr7Training{

    private Boolean cashManagementTraining;
    private Boolean healthCode;
    private Boolean foodTraining;

```

The above picture shows a snapshot of codes contained in the file “Wlkr7Employee.java”. This shows that this class is a subclass of “Wlkr7BasicInfo” and it implements the interface file “Wlkr7Training”

-Utilizing Code Elements:

Collection Classes:

```

public class Wlkr7EmployeeRecord extends Application {

    private Stage primaryStage;
    private AnchorPane rootLayout;

    private ObservableList<Wlkr7Employee> employeeData = FXCollections.observableArrayList();

    public Wlkr7EmployeeRecord(){

```

The above picture is a snapshot of codes contained in the file “Wlkr7EmployeeRecord.java”. This shows that there is an ArrayList utilized here to serve as a collection class handling an ArrayList of java object which is Wlkr7Employee.

```

try {
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(Wlkr7EmployeeRecord.class.getResource("MainPage.fxml"));
    rootLayout = (AnchorPane) loader.load();

    Scene scene = new Scene(rootLayout);

    MainPageController controller = loader.getController();
    controller.setMainApplication(this);

    primaryStage.setScene(scene);
    primaryStage.show();
} catch (IOException ex) {
    throw ex;
}

```

The above picture is a snapshot of codes contained in the file “Wlkr7EmployeeRecord.java”. This is an example of one of many Exception handling in this program. A try-catch is used to catch an exception and a different set of code is used to handle the captured exception.

```

@FXML
public void handleOpen() throws IOException, Exception{
    FileChooser fileChooser = new FileChooser();
    File file = fileChooser.showOpenDialog(mainApp.getPrimaryStage());

    String jsonString = new String();
    if(file != null){
        try{
            FileReader fileReader = new FileReader(file.getPath());
            BufferedReader bufferedReader = new BufferedReader(fileReader);

            String inputLine;
            while((inputLine = bufferedReader.readLine()) != null){
                jsonString += inputLine;
            }
            bufferedReader.close();
        }catch(IOException ioex){
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Unable to Open File");
            alert.setHeaderText("File Unable to be opened");
            alert.setContentText("Sorry, the file you have selected may have been corrupted or invalid. Please try with a different file.");

            alert.showAndWait();
        }

        System.out.println(jsonString);

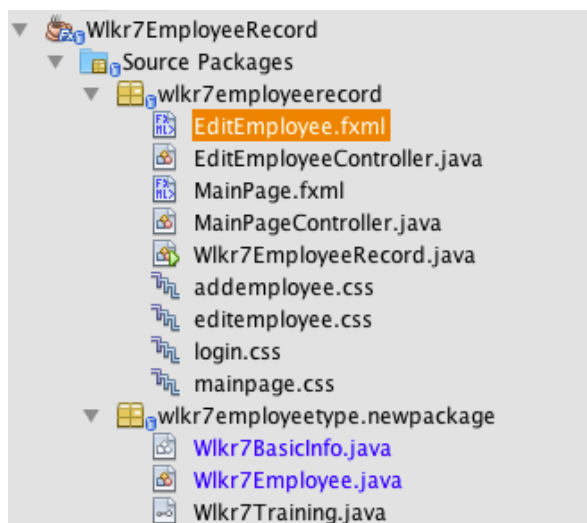
        JSONArray array;
        try{
            array = parseJSONArray(jsonString);
        }catch(Exception ex){
            throw ex;
        }

        for(Object e: array){
            try{
                JSONObject employeeParsed = (JSONObject)e;
                Wlkr7Employee employee = new Wlkr7Employee();
                employee.initFromJsonString(employeeParsed.toString());
                mainApp.getPersonData().add(employee);
            }catch(Exception ex){
                throw ex;
            }
        }
    }
}

```

The above picture is a snapshot of codes contained in the file “MainPageController.java”. This is another example of Exception handling in this program. Multiple try-catch is used to ensure an exception is properly handled based on the codes.

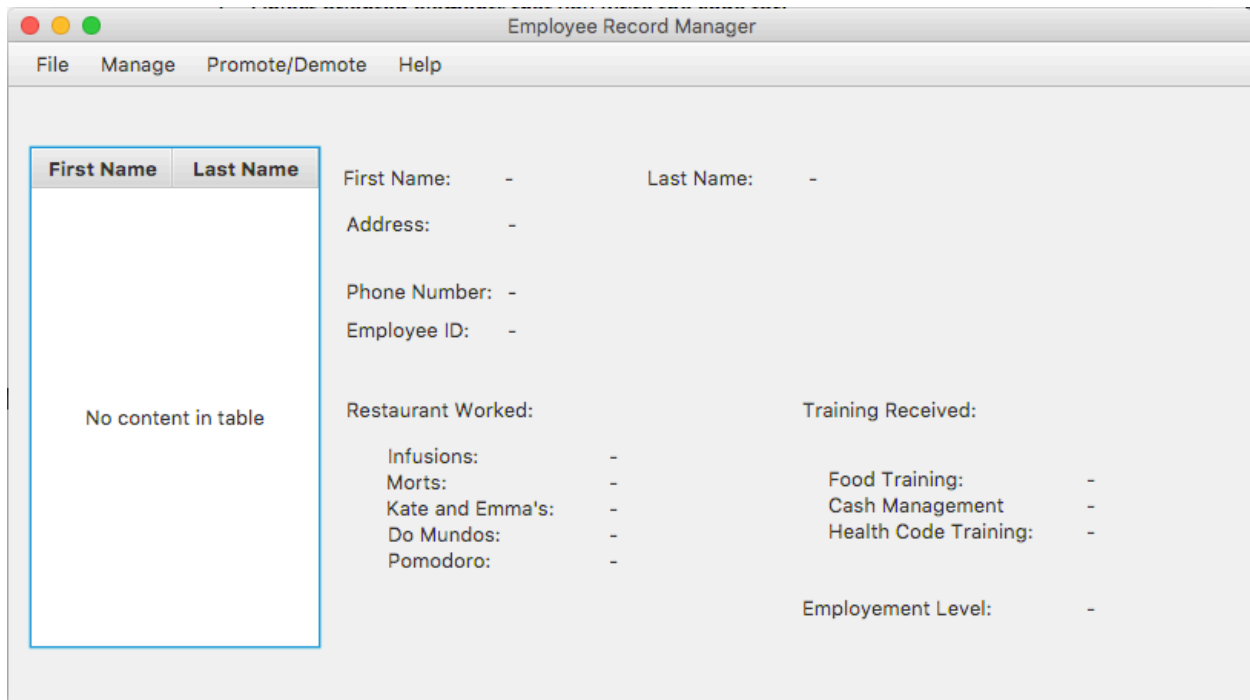
-Model-View-Controller (MVC)



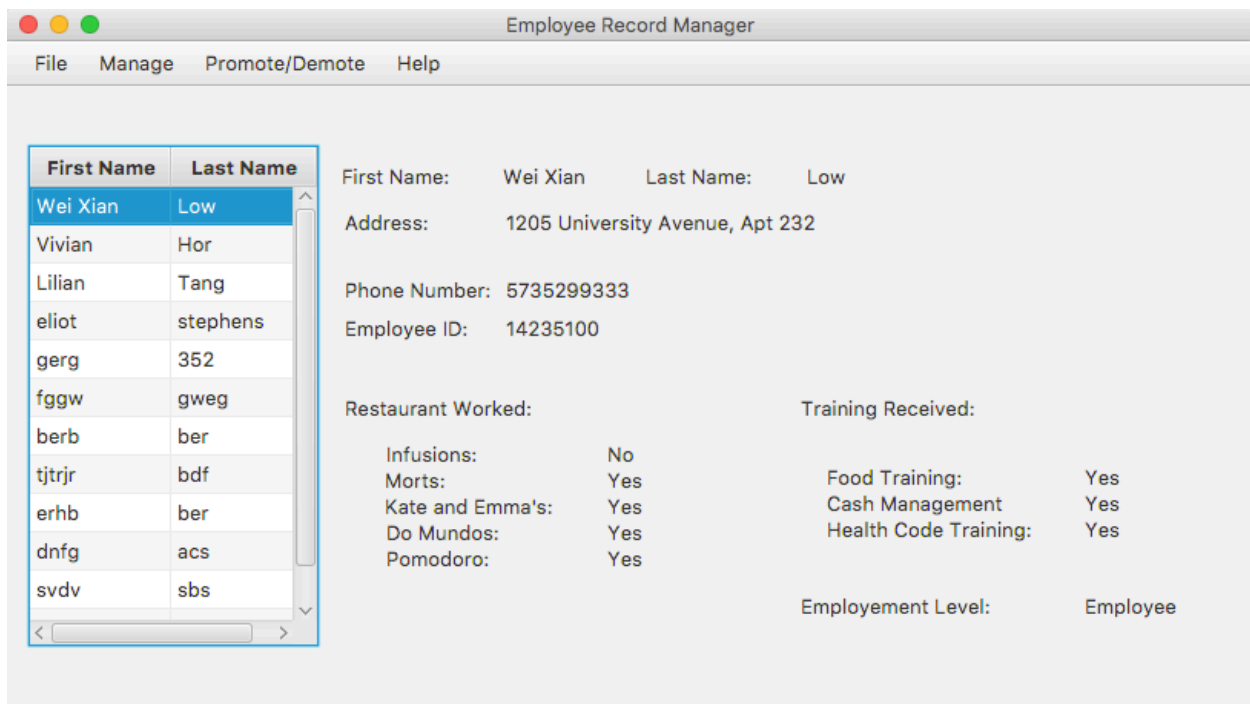
Based on the picture on the left, there's a clear look on how the MVC is implemented here in this program. This program uses JavaFX to develop the MVC.

There are three scenes where user could interact with as shown by the .fxml files.

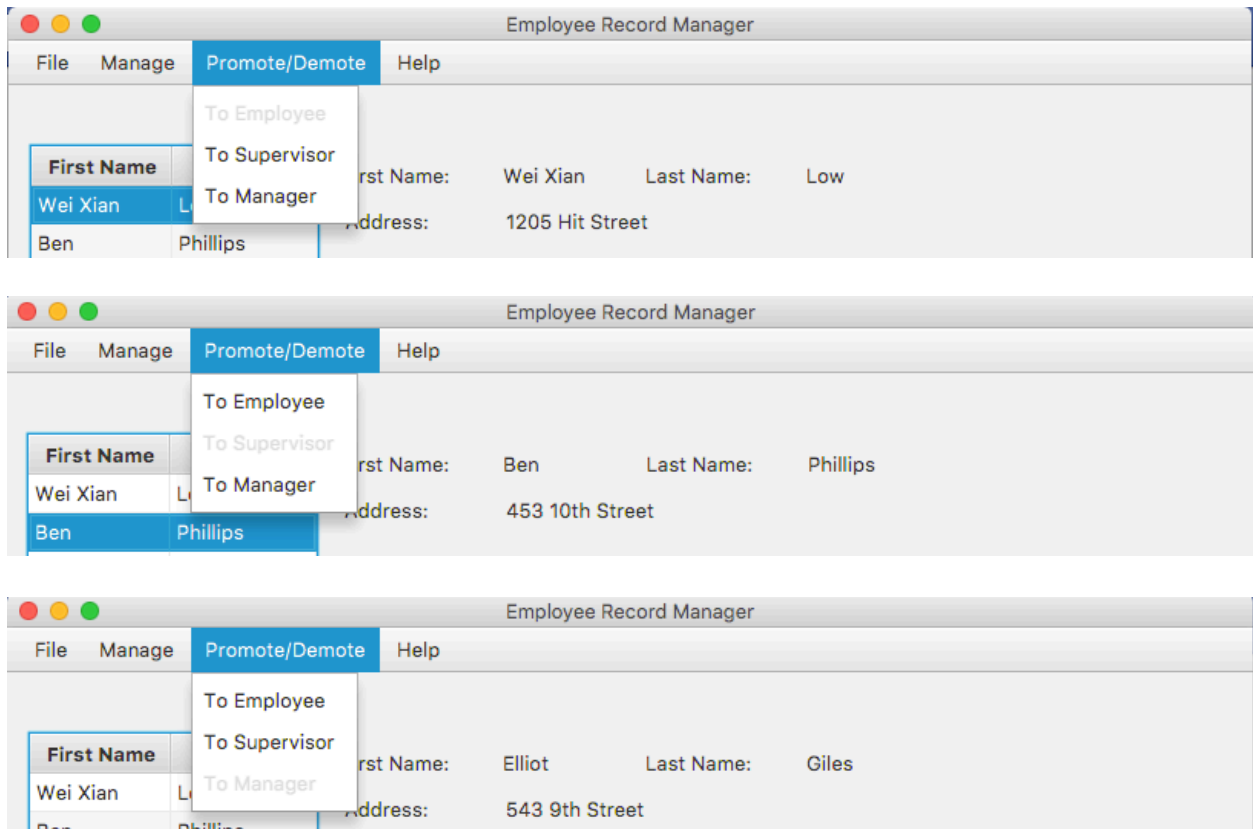
There are also controller files that controls and manipulate data based on the user's interaction with the scene.

-User Interfaces/Scenes

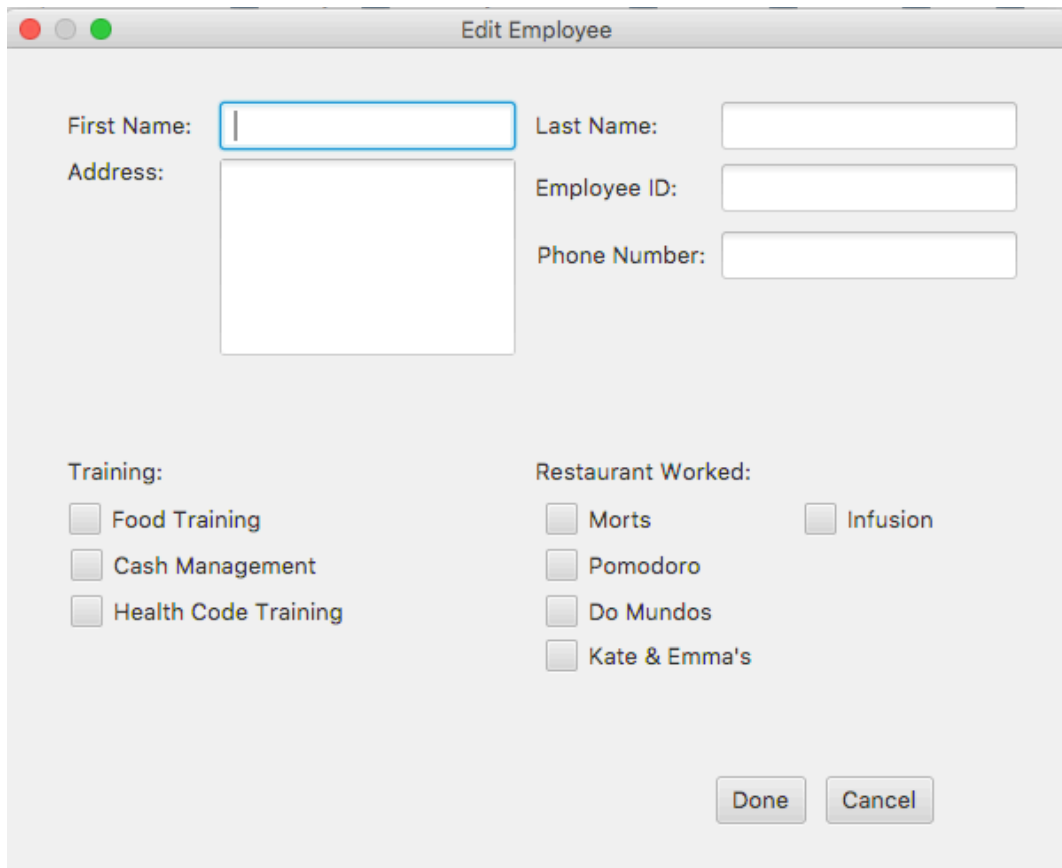
The above picture is a snapshot taken from the program. This scene serves as a main screen where user will first see when the program has complete its initialization.



The above picture is a snapshot taken from the program. This scene shows how the program will look like when it's populated with data and information provided by the user.

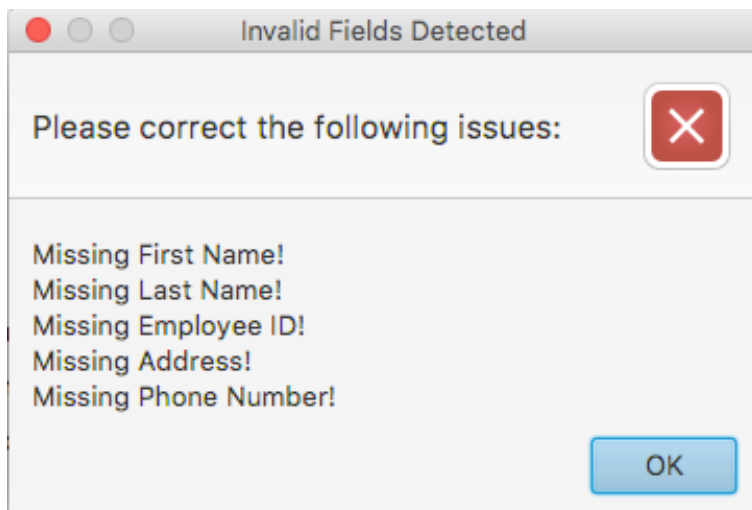


The above snapshots are taken from the application showing the availability of a menu item depending on the state of the data selected from the TableView on the left of the program. If the employee has an employee status, the employee menu item will be disabled to remind the user that the selected employee is an employee. This works too when the employee is a supervisor, and a manager.



The 'Edit Employee' dialog box is a standard macOS-style window with a title bar containing red, yellow, and green window control buttons. The title is 'Edit Employee'. The form is organized into two columns. The left column contains 'First Name:' with a single-line text field, 'Address:' with a larger multi-line text area, and a 'Training:' section with three unchecked checkboxes: 'Food Training', 'Cash Management', and 'Health Code Training'. The right column contains 'Last Name:' with a single-line text field, 'Employee ID:' with a single-line text field, 'Phone Number:' with a single-line text field, and a 'Restaurant Worked:' section with five unchecked checkboxes: 'Morts', 'Infusion', 'Pomodoro', 'Do Mundos', and 'Kate & Emma's'. At the bottom right, there are two buttons: 'Done' and 'Cancel'.

The above snapshot is taken from the application, showing the ability to have a pop-up window when the user clicks on the add employee menu item on the menubar. An empty form will be produced for the user to interact with.



The 'Invalid Fields Detected' alert dialog is a small window with a title bar containing red, yellow, and green window control buttons. The title is 'Invalid Fields Detected'. The main content area has a header 'Please correct the following issues:' followed by a list of five error messages: 'Missing First Name!', 'Missing Last Name!', 'Missing Employee ID!', 'Missing Address!', and 'Missing Phone Number!'. A red square button with a white 'X' icon is located to the right of the header. At the bottom right, there is a blue 'OK' button.

The above snapshot is taken from the application, if the user were to click done on the previous picture without filling out any information required, an alert will be prompted to the user to warn them of their actions.

Edit Employee

First Name: Last Name:

Address: Employee ID:

Phone Number:

Training:

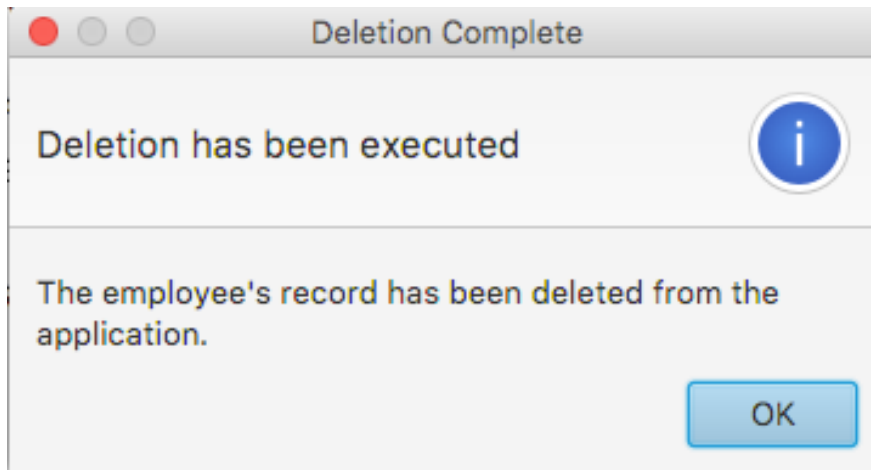
- ☒ Food Training
- ☐ Cash Management
- ☒ Health Code Training

Restaurant Worked:

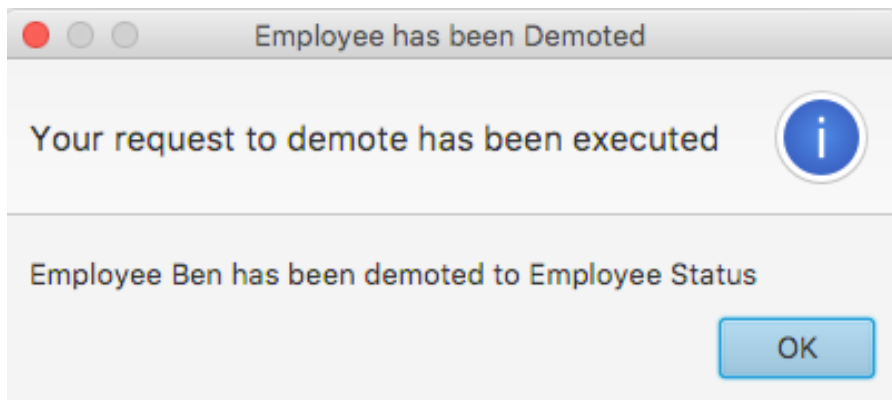
- ☐ Morts
- ☐ Pomodoro
- ☐ Do Mundos
- ☐ Kate & Emma's
- ☒ Infusion

Done Cancel

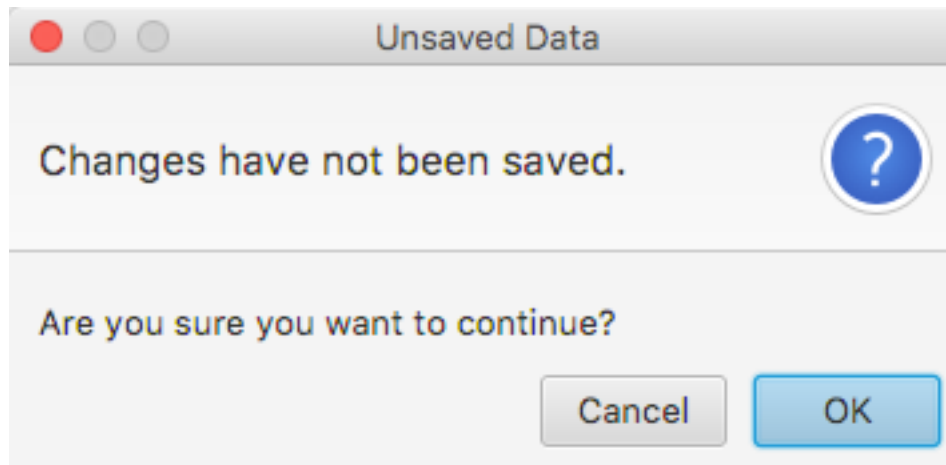
The above snapshot is taken from the application. This snapshot shows that when an employee is selected from the TableView, where the user chooses to edit the employee information, a pop-up window will appear showing the form where it has already been populated with the selected employee's information.



The above snapshot is taken from the application. This snapshot shows an alert prompt that is given to the user if the user chooses to delete the selected employee's record.

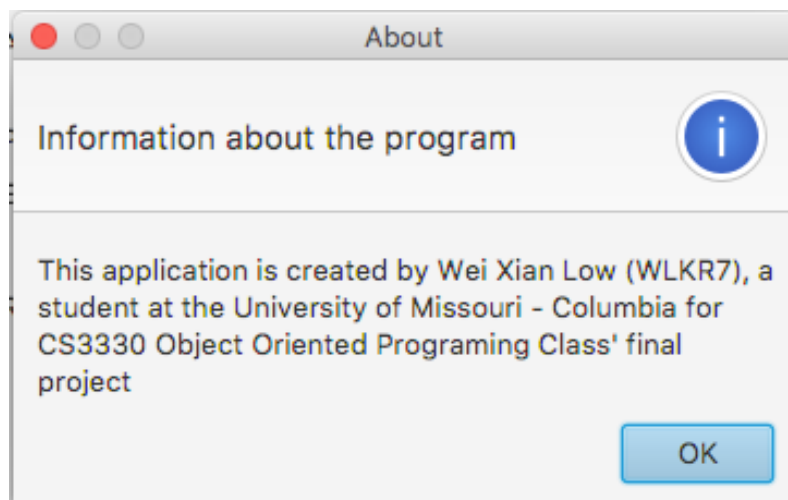


The above snapshot is taken from the application. This snapshot shows an alert prompt that is given to the user if the user chooses to promote or demote an employee's status. A different message appears depending on the state of the promotion or demotion. In the snapshot above, the employee is demoted to a basic employee status.

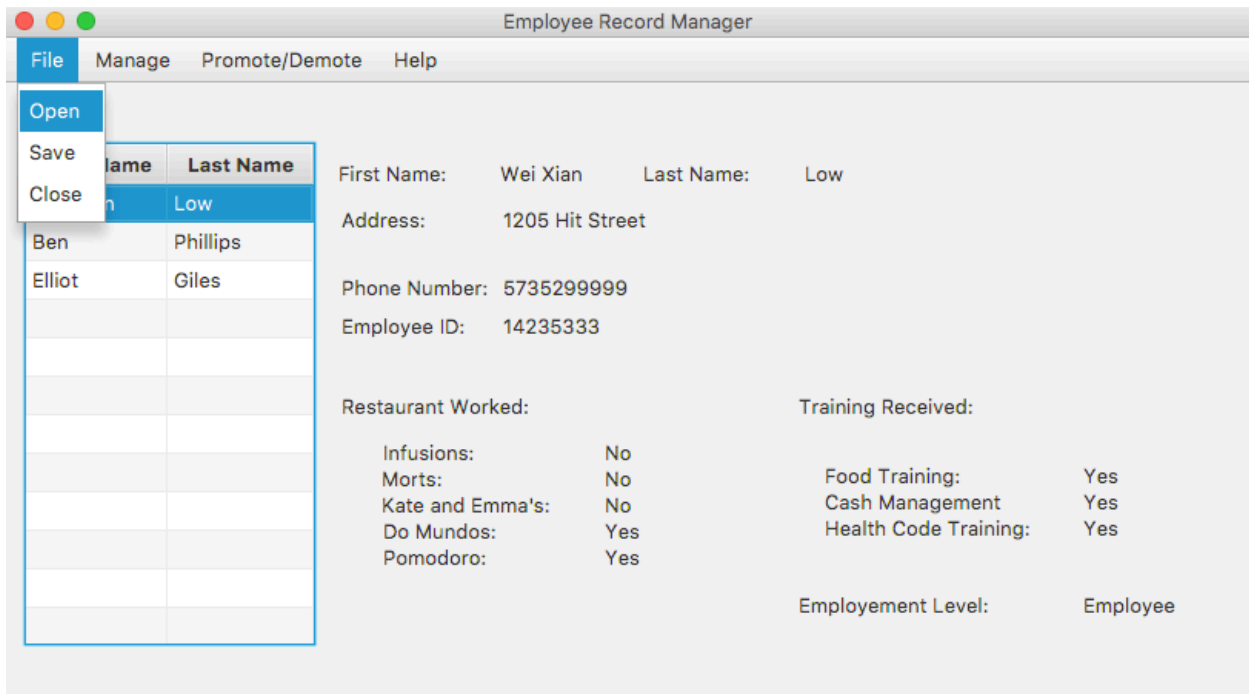


The above snapshot is taken from the application. This snapshot shows an alert prompt to the user if the user clicks the closes menu item from the menu bar or directly closes the window. An alert will be given warning user that the data is not saved after modification of employee's record.

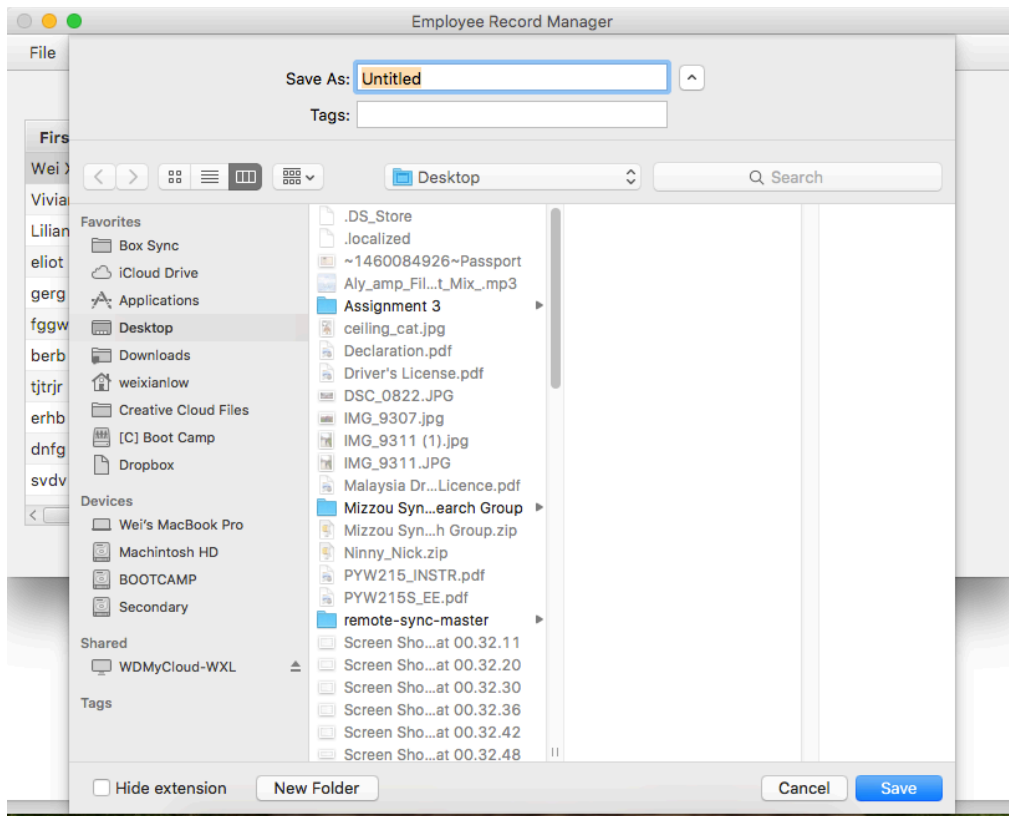
-About information



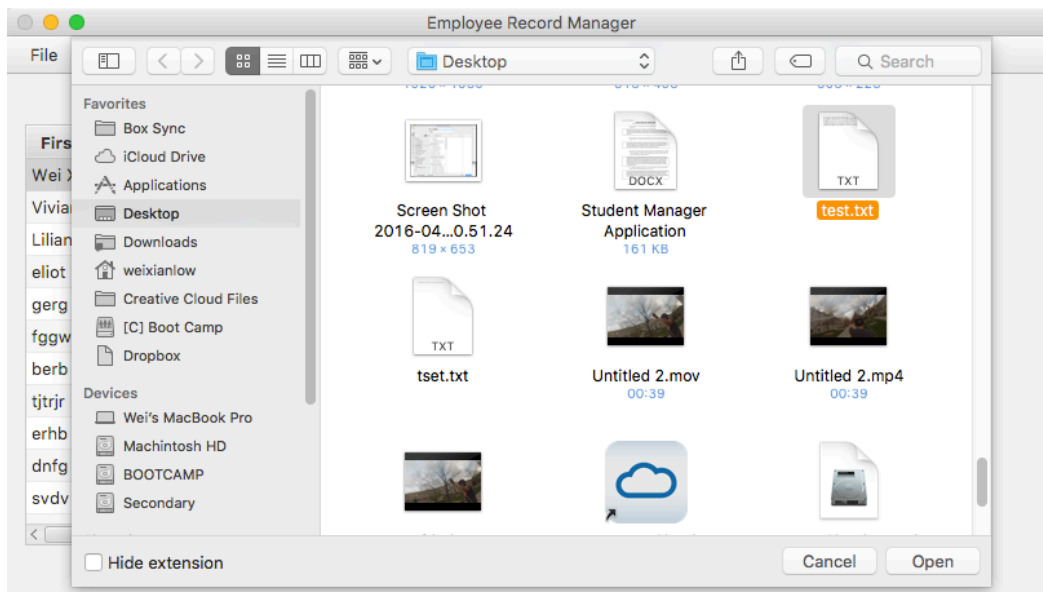
The above snapshot is taken from the application. This snapshot shows that the about information could be reached by selecting the about menu item in the menu bar.

-Save and Open data

The above snapshot is taken from the application. This snapshot shows that users are able to save and open their data that is created in the application. The Open and Save menu item is available at the menubar.



The above snapshot is taken from the application. This snapshot shows that a save dialog is shown to the user in determining where to save the data.



The above snapshot is taken from the application. This snapshot shows that an open dialog is showed to the user in determining which file to open.

```

@FXML
public void handleSave(){

    ObservableList<Wlkr7Employee> listToSave = mainApp.getPersonData();

    if(listToSave == null){
        System.out.println("error at reading listToSave");
        return;
    }

    FileChooser fileChooser = new FileChooser();
    File file = fileChooser.showSaveDialog(mainApp.getPrimaryStage());

    JSONArray array = new JSONArray();
    JSONObject jsonObj;
    for(Wlkr7Employee e: listToSave){
        jsonObj = e.toJsonString();
        array.add(jsonObj);
    }

    if(file != null){
        try{
            String jsonString = array.toJSONString();
            System.out.println(jsonString);
            PrintWriter out = new PrintWriter(file.getPath());
            out.print(jsonString);
            out.close();
            edited = false;
        }catch(IOException ioex){
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Exception Dialog");
            alert.setHeaderText("Exception!");
            alert.setContentText("Error writing to: " + file.getPath());

            // Create expandable Exception.
            StringWriter sw = new StringWriter();
            PrintWriter pw = new PrintWriter(sw);
            ioex.printStackTrace(pw);
            String exceptionText = sw.toString();

            Label label = new Label("The exception stacktrace was:");

            TextArea textArea = new TextArea(exceptionText);
            textArea.setEditable(false);
            textArea.setWrapText(true);

            textArea.setMaxWidth(Double.MAX_VALUE);
            textArea.setMaxHeight(Double.MAX_VALUE);
            GridPane.setVgrow(textArea, Priority.ALWAYS);
            GridPane.setHgrow(textArea, Priority.ALWAYS);

            GridPane expContent = new GridPane();
            expContent.setMaxWidth(Double.MAX_VALUE);
            expContent.add(label, 0, 0);
            expContent.add(textArea, 0, 1);

            // Set expandable Exception into the dialog pane.
            alert.getDialogPane().setExpandableContent(expContent);

            alert.showAndWait();
        }
    }
}

```

The above snapshot of code is taken from the file “MainPageController.java”. This portion of the code shows the code responsible in handling saving user’s data. The code saves the employee’s record in an JSONObject format, in which later is handled and saved in an JSONArray object. The final save file will include the JSON string where it’s readable by user and allows open source of save file to be open in other application.

```

@FXML
public void handleOpen() throws IOException, Exception{
    FileChooser fileChooser = new FileChooser();
    File file = fileChooser.showOpenDialog(mainApp.getPrimaryStage());

    String jsonString = new String();
    if(file != null){
        try{
            FileReader fileReader = new FileReader(file.getPath());
            BufferedReader bufferedReader = new BufferedReader(fileReader);

            String inputLine;
            while((inputLine = bufferedReader.readLine()) != null){
                jsonString += inputLine;
            }
            bufferedReader.close();
        }catch(IOException ioex){
            throw ioex;
        }

        System.out.println(jsonString);

        JSONArray array;
        try{
            array = parseJSONArray(jsonString);
        }catch(Exception ex){
            throw ex;
        }

        for(Object e: array){
            try{
                JSONObject employeeParsed = (JSONObject)e;
                Wlkr7Employee employee = new Wlkr7Employee();
                employee.initFromJsonString(employeeParsed.toJSONString());
                mainApp.getPersonData().add(employee);
            }catch(Exception ex){
                throw ex;
            }
        }
    }
}

```

The snapshot of code above is taken from the file “MainPageController.java”. This portion of the code handles the open operation when asked by user. This code will open a file previously saved by the user. The code will then parse the JSON related code and execute them inside the program.