Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 2

Title of the Laboratory Exercise: Simple application design and development

1. Introduction and Purpose of Experiment

   Students apply object oriented programming concepts for creating simple applications.

2. Aim and Objectives

   Aim

   To apply object oriented programming concepts for creating simple applications

   Objectives

   At the end of this lab, the student will be able to

   - Apply Object Oriented Programming to create simple applications
   - Create stand-alone applications in Java

3. Experimental Procedure

For the problems listed below, design the data structures, algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Java with C and Haskell programming languages. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Create a simple attendance management application using FXML in Java. Include exception handling logic that reports exceptions in human readable pop-up messages

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	
b. Conclusions :	

Procedure:

Netbeans was opened and new JavaFX FXML Application was created, the hence generated project contains some sample code that was removed and the default FXMLDocument was edited in SceneBuilder, the necessary UI Elements are added here. Specific fx:id are assigned to the components that are to be controlled by the Controller Class. The Styling is added to the components needed using CSS or inline CSS and then the Controller Class of the fxml is generated using NetBeans Generate Controller Utility. Then the logic to what to do when those buttons are clicked are typed. Debugging is done and the final code is Build and Tested for a few types of input.

Algorithms:

```
Note: tp refers to Integer total present, ts refers to Integer total
students

Initialize total students with the number of children in VBox
studentList.

goToAttendance:

1. Set the studentListScreen to front of the StackPane

goToStudentsList:
1. Set the totalPresent Text to String of tp
2. Set the totalAbsent Text to String of (ts-tp)
3. Set the totalStudents to ts
4. Set the attendanceScreen to front of the StackPane

updateAttendance:

1.  Get the event source and type cast it to CheckBox, then store it in
    checkbox.
2.  If checkbox is selected then increment tp
    Else decrement tp
```

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Conclusions:

A simple Attendance Management Application was created that on the first screen displays the list of students in that class along with a checkbox with each of these students name, when the checkbox is clicked and we know that the checkbox is checked the current number of present members is incremented by 1, else if it is not checked then the number of present members is decremented by 1.

There are some limitations to using the kind of approach used in the implemented program, such as the list of student is hardcoded in the UI and fixed, though the code part of it is little generalized in the sense that the total number of students is the number of children that the VBox contains, i.e. the number of students in the list, even if there are more students added to this VBox dynamically then the total students also change accordingly.

One recommendation that could be given is that the Students list can be stored in an ArrayList, if there are any new students to be added or any student to be removed, then it should be removed from this list. Then on initialization of the UI, the student from this ArrayList can be added to the list of the students in the UI. This is more flexible now, something even more flexible would be to store the Attendance System into another class itself that manages the students and also the attendance on each day. One more thing that would be a problem is that there can be duplicate entries added by mistake, so in order to make it standardized Students can have their own Classes and objects of these classes would be added to the List, so since each of the student has a Unique ID, i.e. the Reg No. we can now check if the student already exists in the list, and do the necessary. This is even more flexible and much more maintainable code, and OOP is used to the full advantage in such a case.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

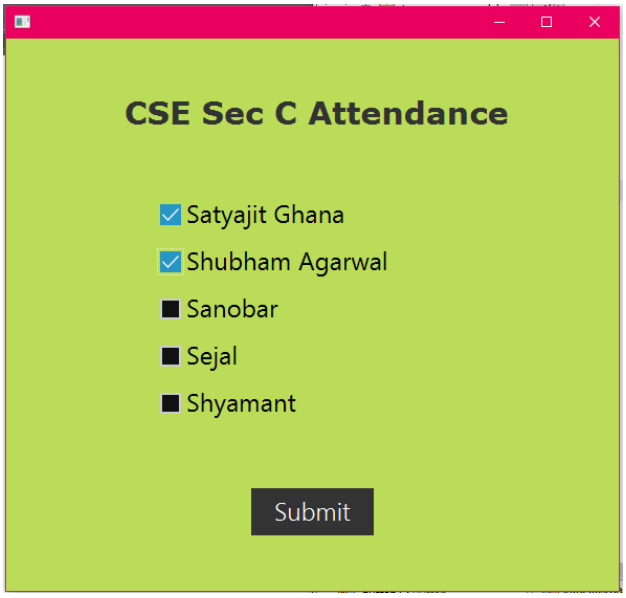
Results and Discussions:
Screenshot:
Discussion:

Core Logic Source Code:

```
    private Integer ts;
    private Integer tp;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        ts = studentsList.getChildren().size();
        tp = 0;
    }

    @FXML
    private void goToAttendance(ActionEvent event) {
        studentListScreen.toFront();
    }

    @FXML
    private void goToStudentList(ActionEvent event) {
        totalPresent.setText(tp.toString());
        totalAbsent.setText((ts-tp)+"");
        totalStudents.setText(ts.toString());
        attendanceScreen.toFront();
    }

    @FXML
    private void updateAttendance(ActionEvent event) {
        CheckBox checkbox = (CheckBox) event.getSource();
        if (checkbox.isSelected()) {
            tp++;
        } else {
            tp--;
        }
    }
```

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Output and Discussion:



The Application was run and the following items were checked in sequence: Satyajit Ghana, Shubham Agarwal, we can observe that the boxes are now checked and also in the backend the code for clicked on checkbox is run, the attendance is already calculated at this point for the selected items since the logic of setting the text on the attendanceScreen is executed on Submit Button click but that screen is not visible since it is at the back of the stack.

Then the Submit button was clicked, now the attendance AnchorPane is set to the front of the screen that displays the attendance of the Students. When back is clicked the studentListScreen is set to Front of the screen.