Odd/Even/Compact* Semester (year)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	***Surname***	***Given Names***	***Student ID Number***
	Wijaya	*Claudia Rachel*	*2301954892*

Course Code	**: COMP6571**	**Course Name**	**: Programming Language**
Class	**: L2BC**	**Name of Lecturer**	**:** Mr. Jude Martinez
Major	**: Computer Science**		
Title of Assignment	: Check (Expense Tracker)		
Type of Assignment	**: Final project**		
Submission Pattern			
Due Date	**: 20 June 2020**	**Submission Date**	**: 20 June 2020**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

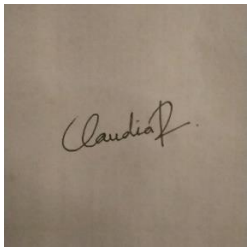
Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: *(Name of Student)*



Claudia Rachel Wijaya

**) Delete the inappropriate option*

Final Project Report

A JavaFx Expense Tracker

Student: Claudia Rachel Wijaya

NIM: 2301954892

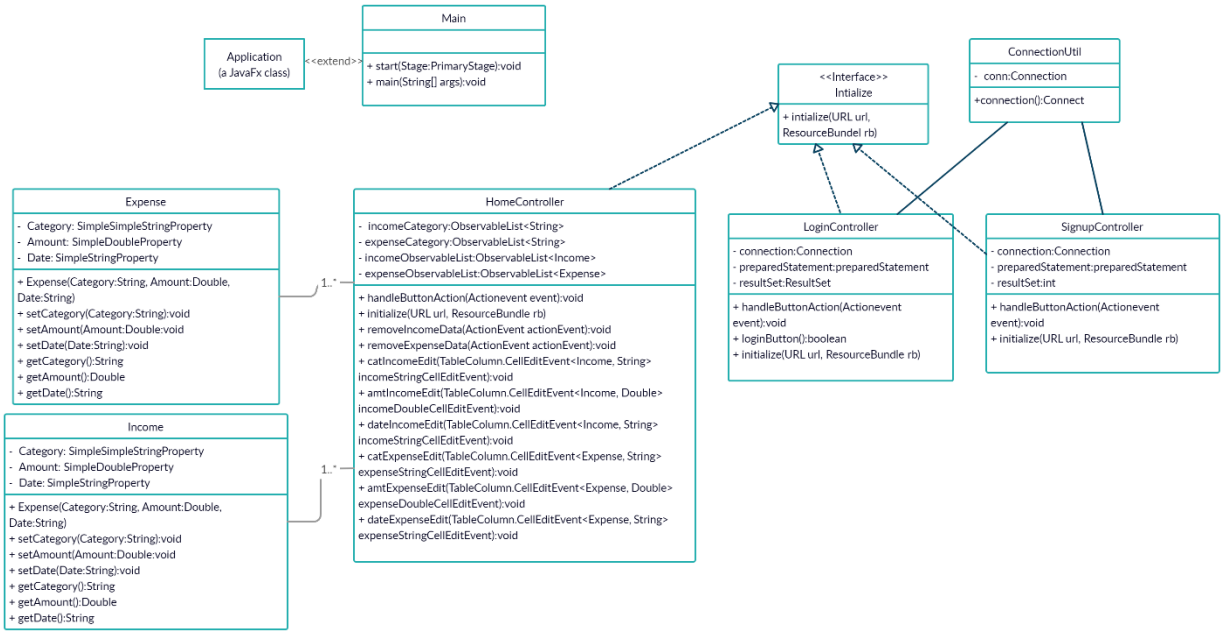
Project Specification

Check is a program made using Java, JavaFx and MySQL. It is an expense tracker that will track how much the user has spent or earned. The user prior that needs to login or signup to a localhost database to be able to have access to the program. The goal of this project has been specified by lecturers in that it should be a challenge and apply the knowledge learned outside of the class. JavaFx, which is a software platform that is used to create applications, was not taught in class hence it was best to use JavaFx. MySQL was only discussed during lab practices. Thus, MySQL and JavaFx was incorporated into this project.

Solution Design

There are a quite a number of expense trackers that are available on mobile. I personally use an app called Money Manager. It was quite simple in that it has a table of expenses and incomes with a pie chart to show the distribution of expenses according to its category. Hence, Check has the following pages:

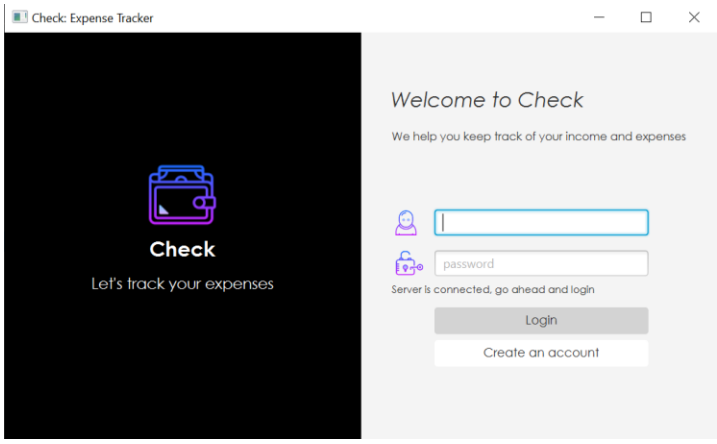
- Login page
- Signup Page
- Home:
    - Income Tab
    - Expense Tab
    - Chart Tab



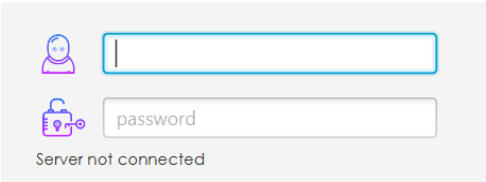
Discussion

I. Login Page

To start off the program, XAMPP must be used to turn on Apache and MySQL, otherwise the program would not allow you to login. This is the first scene that appears when the program is run.



The text underneath the password field indicates our connection to MySQL. According to the label, the app is connected. Otherwise, this will show:



The loginButton() method is used to check if the fields are both filled when entering and also check if the user's details exist in the database.

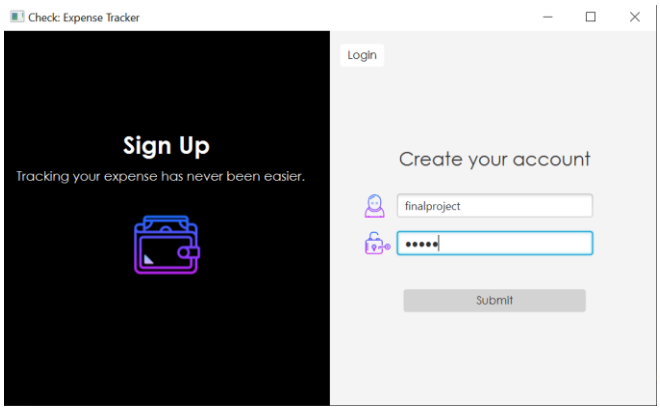
```java
private boolean loginButton() {

    // we first get the strings from both the username and password fields
    String username = usernameText.getText();
    String pass = passText.getText();
    // in the case of one or both fields are empty
    if (username.isEmpty() || pass.isEmpty()) {
        errorlogs.setTextFill(Color.RED);
        errorlogs.setText("Fill both fields please");

    } else {
        // one we get the strings, we search the database from mysql
        String sql = "SELECT * FROM usernames WHERE username = ? and password = ?";
        try {
            preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setString( parameterIndex: 1, username);
            preparedStatement.setString( parameterIndex: 2, pass);
            resultSet = preparedStatement.executeQuery();
            if (!resultSet.next()) { // if user has put incorrect details
                errorlogs.setTextFill(Color.RED);
                errorlogs.setText("Incorrect username/password");
                return false;
            } else {
                // user puts correct detail
                errorlogs.setText("Logging in...");
                return true;
            }
        } catch (SQLException e) {
            System.err.println(e.getMessage());
            return false;
        }
    }
    return false;
}
```

The function returns a boolean value by the user's details exist in the database, if it returns true, the user will be directed immediately to the Home page.

```
public void handleButtonAction (ActionEvent event){
    // when user clicks on a button, this function figures out which button has been clicked
    if (event.getSource() == loginButton){
        if (loginButton()){
            try {
                Parent pane = FXMLLoader.load(getClass().getResource( name: "/fxml/Home.fxml"));
                Scene scene = new Scene(pane);
                Stage window = (Stage)((Node)event.getSource()).getScene().getWindow();
                window.setScene(scene);
                window.show();
            } catch (IOException e) {
                System.err.println(e.getMessage());
            }
        }
    }
```

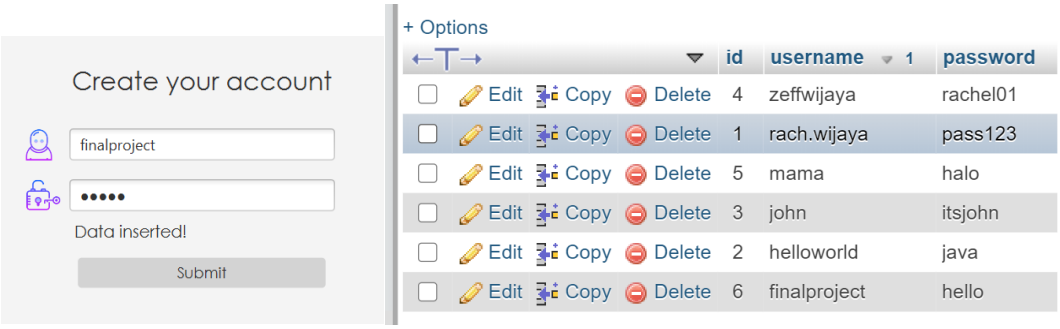
II. Signup Page

If not, the user can click the "Create an Account" button that will direct them towards the signup page.



Once the data has been submitted, a label appears that says the data has been inserted. The update can be seen by opening localhost/phpMyAdmin.

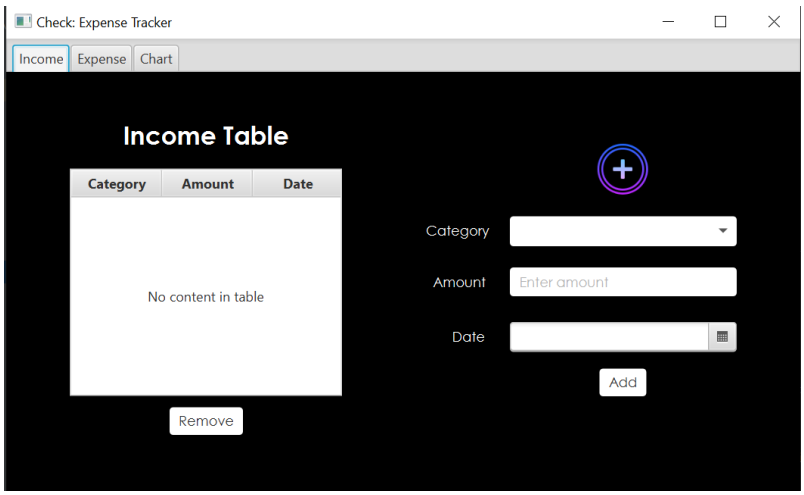
```
public void handleButtonAction(ActionEvent event) {
    String username = userText.getText();
    String password = passText.getText();
    if (event.getSource() == submitButton) {
        if (username.isEmpty() || password.isEmpty()) {
            submitInsert.setTextFill(Color.RED);
            submitInsert.setText("Fill both fields please");

        }
        else {
            String insert = "INSERT INTO usernames (username, password) VALUES (?, ?)";
            try {
                preparedStatement = connection.prepareStatement(insert);
                submitInsert.setText("Data inserted!");
                preparedStatement.setString( parameterIndex: 1, userText.getText());
                preparedStatement.setString( parameterIndex: 2, passText.getText());
                resultSet = preparedStatement.executeUpdate();
            } catch (SQLException e) {
                System.err.println(e.getMessage());
            }
        }
    }
```



The user then can login, leading to the home page.

III. Home page

Income Tab:



Expense tab:

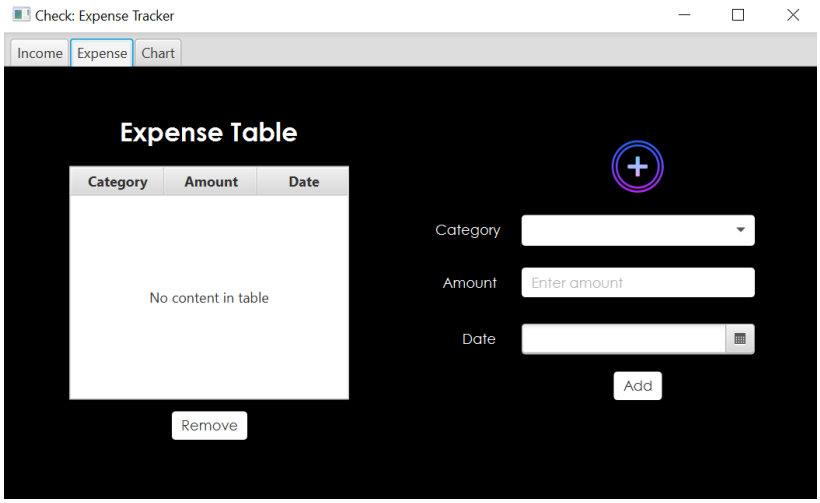
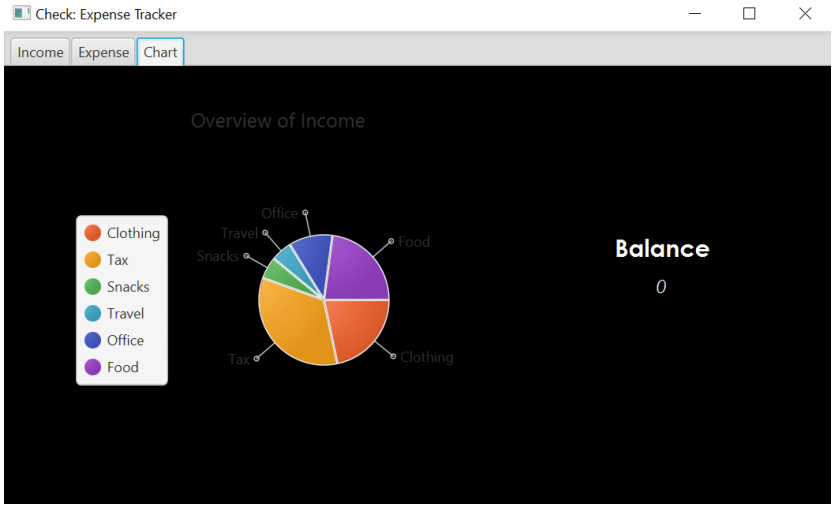
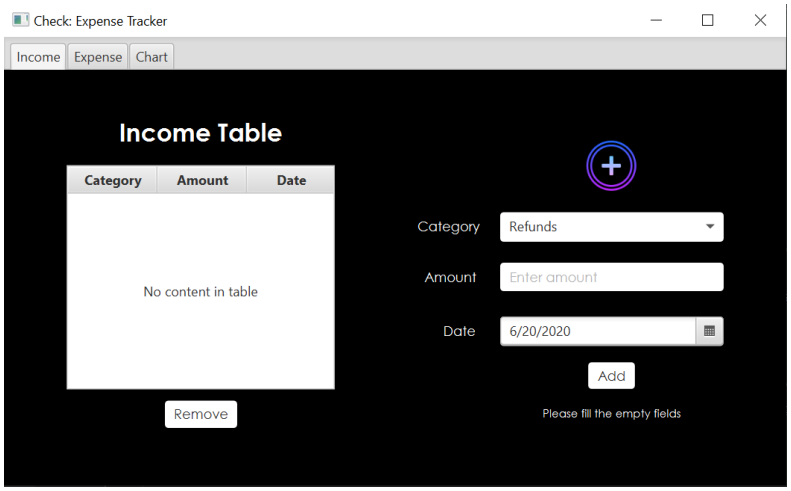


Chart Tab

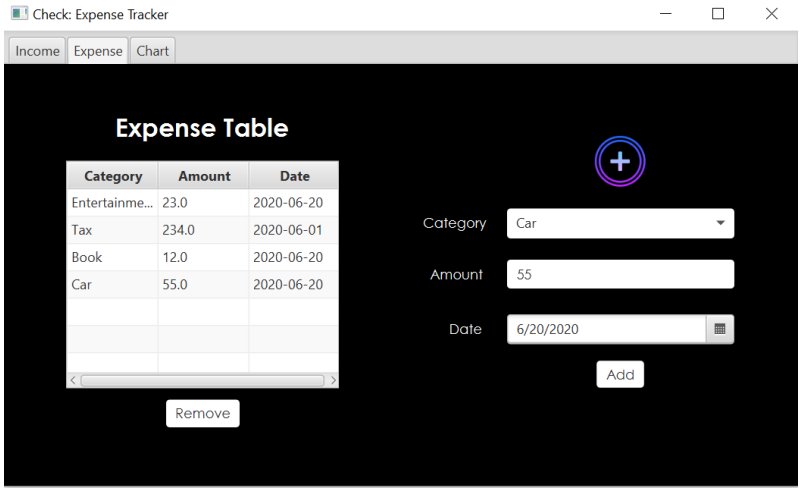


The expense and income tabs are very similar. The user inputs the data, if the category and amount fields are empty, a text appears to show the error. However, for the case of the datepicker, leaving it empty will insert a default date which is the current date.

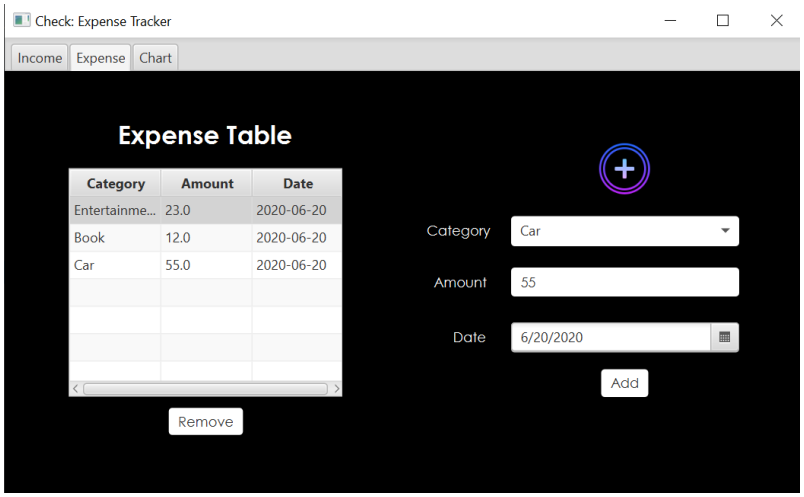
```
void handleButtonAction(ActionEvent event) {
    if (event.getSource() == addIncome){
        String cat = categoryChoiceIncome.getValue();
        String amountIncomeString = amountTextIncome.getText();
        String date;
        if (datePickIncome.getValue() == null){
            // default date if user does not input the date themselves
            datePickIncome.setValue(LocalDate.now());
            date = datePickIncome.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
        }
        if (categoryChoiceIncome.getSelectionModel().isEmpty() || amountIncomeString.isEmpty()){
            incomeErrorLabel.setText("Please fill the empty fields");
        }
        else{
            date = datePickIncome.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
            Income newData = new Income(cat,Double.parseDouble(amountIncomeString), date);
            incomeTable.getItems().add(newData);
        }
    }
```



If all details are correctly filled, this is what can be seen in the table view.



It is also possible to delete single or multiple rows. In this example, the second row has been deleted.



First we create objects, incomeAll is a variable that holds all of the items inside the table view while incomeRows hold the items in the selected row(s). Then the desired rows that the user wishes to be deleted is removed using the removeAll() method, which is able to delete one or multiple rows.

```
public void removeIncomeData(ActionEvent actionEvent) {
    ObservableList<Income> incomeRows, incomeAll;
    incomeAll = incomeTable.getItems();
    incomeRows = incomeTable.getSelectionModel().getSelectedItems(); // gets selected rows
    incomeAll.removeAll(incomeRows);
}
public void removeExpenseData(ActionEvent actionEvent) {
    ObservableList<Expense> expenseRows, expenseAll;
    expenseAll = expenseTable.getItems();
    expenseRows = expenseTable.getSelectionModel().getSelectedItems(); // gets selected rows to be removed
    expenseAll.removeAll(expenseRows);
}
```

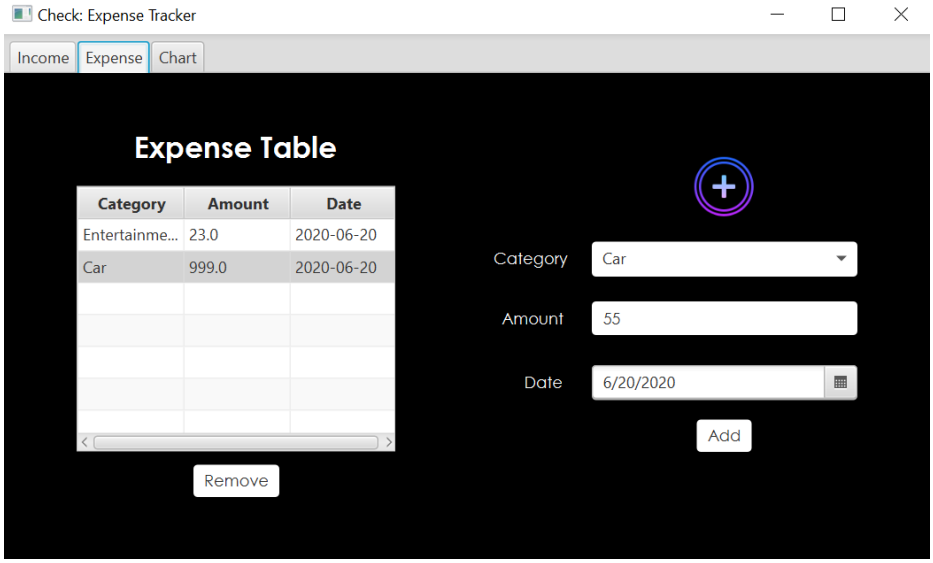
A cell value can also be edited on the table view itself. When initializing the page, the table views must be set to editable.

```
// Editable income table view
incomeTable.setEditable(true);
catIncomeT.setCellFactory(TextFieldTableCell.forTableColumn());
amtIncomeT.setCellFactory(TextFieldTableCell.forTableColumn(new DoubleStringConverter())); // String input -> Double
dateIncomeT.setCellFactory(TextFieldTableCell.forTableColumn());

// Editable expense table view
expenseTable.setEditable(true);
catExpenseT.setCellFactory(TextFieldTableCell.forTableColumn());
amtExpenseT.setCellFactory(TextFieldTableCell.forTableColumn(new DoubleStringConverter())); // String input -> Double
dateExpenseT.setCellFactory(TextFieldTableCell.forTableColumn());
```

These edit functions call the set methods from the Income class to set an instance variable.

```
public void catIncomeEdit(TableColumn.CellEditEvent<Income, String> incomeStringCellEditEvent) {
    Income income = incomeTable.getSelectionModel().getSelectedItem();
    income.setCategory(incomeStringCellEditEvent.getNewValue());
}
public void amtIncomeEdit(TableColumn.CellEditEvent<Income, Double> incomeDoubleCellEditEvent) {
    Income income = incomeTable.getSelectionModel().getSelectedItem();
    income.setAmount(incomeDoubleCellEditEvent.getNewValue());
}
public void dateIncomeEdit(TableColumn.CellEditEvent<Income, String> incomeStringCellEditEvent) {
    Income income = incomeTable.getSelectionModel().getSelectedItem();
    income.setDate(incomeStringCellEditEvent.getNewValue());
}
```

Originally, the car's amount was 55, now it displays 999.0.



The chart tab is unfinished, however the idea was to show data from the expense tableview and add that to the piechart. This was the hardest functions to figure out, so at the end, a dummy data has been placed instead.

This is the link to my github:

https://github.com/rachelwijaya/Check-Expense-Tracker