# Semester Project and Explanation of Source Codes related to restuarnt

**Semester Project**

Author:

Supervisors: Prof. Dr. Osama Helmy

Submission Date: 15 December 2019

This is to certify that:

(i) the documentation comprises only my original work

(ii) due acknowledgement has been made in the text to all other material used

15 December, 2019

# Acknowledgments

After thanking Allah (SWT), I would like thank Prof. Dr. Osama Helmy for accepting me to work on his project. I would like also to thank him for introducing my project of our course Concepts Of Programming Language (CS312) and helping me getting through a lot of difficulties throughout the whole project.

# Abstract

In this work, we are going to explain restaurant that are related to our tutrials codes. Also, we describe the in detailed . we show how a data base works with the his staff, to make much more doable.

# Contents

**Acknowledgments** **III**

**Contents** **VI**

**1 Project Description** **1**
1.1 Overview and Motivation . . . . . 1
1.2 After Accessing the data (Cashier Mode Activated)) . . . . . 2

**2 Code Flow and How the Buttons Works** **5**
2.1 How The Data Base connect with Java code . . . . . 5
2.1.1 Insert new Item . . . . . 5
2.1.2 Update new Item . . . . . 8
2.1.3 Delete new Item . . . . . 11
2.1.4 Search new Item . . . . . 13

# Chapter 1

# Project Description

## 1.1 Overview and Motivation

As commonly known or almost problem solving programs that is known to solve a problems in real life. So, we think to prepare a data base for a restaurant to make a customers to receive a seats for their orders. And the one who can access the data base is the cashier, and he will meet the customers orders.

**So, At first. There's a login form which the cashier can access it with his a password to explain how the process will be going to process:**


Figure 1.1: **How the cashier can access the data base**

## 1.2 After Accessing the data (Cashier Mode Activated))

Abridged of the screen: At this one, the cashier is going to write the customer order and he will get a seat to meet the customer needs. So, at this screen the cashier write ID of the meal, the name of the meal and the purchase of the meal. And it the data base will be updated.

Cashiers are also responsible for counting the total of cash register drawer and keep track of all the credit and cash transactions at the end of each shift. Cashiers assist customers by recommending the products, responding to the inquiries and processing refunds or exchanges.

May also be responsible for inspecting the materials and reporting the stock quantities. Overall, you'll make sure that the transaction process runs smoothly and maximize customer satisfaction. Cashiers work under a manager or supervisor and abide by his/her instructions. Your knowledge of common practices in a particular field and friendly attitude plays a vital role in achieving customer satisfaction.


Figure 1.2:

We are looking for a courteous Cashier who possesses excellent communication and customer service skills. The Cashier has to manage the transactions with customers efficiently and accurately. You will scan the selected items by customers and ensure that the quantities and prices are appropriate, collect payments, return the change, issue receipts, answer their inquiries and monitor all the transactions. To be a successful Cashier, you must possess experience in the customer service field and you should have great accuracy and a strong work ethic.

You should also be willing to take weekend shifts and evening shifts occasionally. You should be committed to excellent service and be attentive to the customer inquiries and needs. The ideal candidate will have excellent interpersonal and communication skills and have a friendly attitude towards the customers.

# Chapter 2

# Code Flow and How the Buttons Works

## 2.1 How The Data Base connect with Java code

### 2.1.1 Insert new Item


Figure 2.1: **Inserting a new item**

When there is a new item is added. We take a new 4 variables num, name, type and cost. And we ask, is there an object by this values. If no, not doable! So, the operation not updated. If yes, we will take an object from class its name Meals. After that, the object will be added to the data base and it will be appeared a message that tell u that the object has been added.

```
ic void InsertDrank() {
try {
    int num = Integer.parseInt(numD.getText());
    String name = nameD.getText();
    String type = typeD.getSelectionModel().getSelectedItem().toString();
    int cost = Integer.parseInt(costD.getText());
    if (!DB.insertD(num, name, type, cost)) {
        listD.add(new Dranks(num, name, type, cost));
        doneD.setText("                                    ");
        doneD.setVisible(true);
        numDranks.setText(Integer.parseInt(numDranks.getText()) + 1 + "");
    }
} catch (Exception e) {
}
```

## Meal class

```
package restaurantes;

public class Meals {

    int id, cost;
    String name, type;

    public Meals(int id, String name, String type, int cost) {
        this.id = id;
        this.name = name;
        this.type = type;
        this.cost = cost;

    }

    public int getId() {
        return id;
    }

    public int getCost() {
        return cost;
    }

    public String getName() {
        return name;
    }

    public String getType() {
        return type;
    }

}
```

## How it will be connected to data base

After adding the object we were talking about. Now the role to make all the objects appear in the our interface. So, the follwing codes explain how the all the objects will be apperaed.

```
numTM.setCellValueFactory(new PropertyValueFactory<Meals, Integer>("id"));
nameTM.setCellValueFactory(new PropertyValueFactory<Meals, String>("name"));
typeTM.setCellValueFactory(new PropertyValueFactory<Meals, String>("type"));
costTM.setCellValueFactory(new PropertyValueFactory<Meals, Integer>("cost"));

listM = DB.getMeals();
tableM.setItems(listM);

numTD.setCellValueFactory(new PropertyValueFactory<Dranks, Integer>("id"));
nameTD.setCellValueFactory(new PropertyValueFactory<Dranks, String>("name"));
typeTD.setCellValueFactory(new PropertyValueFactory<Dranks, String>("type"));
costTD.setCellValueFactory(new PropertyValueFactory<Dranks, Integer>("cost"));

listD = DB.getDranks();
tableD.setItems(listD);
```

After that, the id of the table will be read the IDs of all objects of Drinks and Meals. listM -object from list data type from class Meal-, it will be

read all the obects data column by column. And the setItem is a function which will assign the items the items to the table -item which is the cell or the intersection of the row and the column-.

## Inserting into Data Base

The following sql code is to add into the Data Base

```
ic static boolean insertD(int id, String name, String type, int cost) {
Connection con = cannent();
try {
    PreparedStatement ps = con.prepareStatement("insert into Dranks Values('" + id + "
        ','" + name + "','" + type + "','" + cost + "')");
    return ps.execute();
} catch (Exception e) {

    JOptionPane.showMessageDialog(null, "
                                                ;

} finally {
    try {
        con.close();
    } catch (Exception e) {

    }
}

return true;
}
```

So, we have two previous source code. The first one, is to add into menu. And the second is to add into the data base.

### 2.1.2 Update new Item


Figure 2.2: **Modifying — updating a new item**

When there is a new item -either meals or drinks- is updated. We take a new 1 or more than variables num, name, type and cost. And we ask, is there an object by this values. If no, not doable! So, the operation not updated. If yes, we will take an object from class its name Meals. After that, the object will be added to the data base and it will be appeared a message that tell u that the object has been added.

```
public void updataMeals() {
        try {
            int num = Integer.parseInt(numM.getText());
            String name = nameM.getText();
            String type = typeM.getSelectionModel().getSelectedItem().toString();
            int cost = Integer.parseInt(costM.getText());

            if (DB.updateM("where numM=" + num, name, type, cost)) {
                listM.set(indexM, new Meals(num, name, type, cost));
                doneM.setText("                                    ");
                doneM.setVisible(true);
                clearM();

            }
        } catch (Exception e) {
        }

    }

    public void updataDrank() {
        try {
            int num = Integer.parseInt(numD.getText());
            String name = nameD.getText();
            String type = typeD.getSelectionModel().getSelectedItem().toString();
            int cost = Integer.parseInt(costD.getText());
            if (DB.updateD("where numD=" + num, name, type, cost)) {
                listD.set(indexD, new Dranks(num, name, type, cost));
                doneD.setText("                                    ");
                doneD.setVisible(true);
                clearD();

            }

        } catch (Exception e) {
        }

    }
```

# Meal — Drinks class

```
package restaurantes;

public class Meals {
```

```
    int id, cost;
    String name, type;

    public Meals(int id, String name, String type, int cost) {
        this.id = id;
        this.name = name;
        this.type = type;
        this.cost = cost;

    }

    public int getId() {
        return id;
    }

    public int getCost() {
        return cost;
    }

    public String getName() {
        return name;
    }

    public String getType() {
        return type;
    }

}

public class Dranks {

int id, cost;
String name, type;

public Dranks(int id, String name, String type, int cost) {
    this.id = id;
    this.name = name;
    this.type = type;
    this.cost = cost;
}

public int getId() {
    return id;
}

public int getCost() {
    return cost;
}

public String getName() {
    return name;
}

public String getType() {
    return type;
}
```

## How it will be connected to data base

After adding the object we were talking about. Now the role to make all the objects appear in the our interface. So, the follwing codes explain

how the all the objects will be apperaed.

```
numTM.setCellValueFactory(new PropertyValueFactory<Meals, Integer>("id"));
nameTM.setCellValueFactory(new PropertyValueFactory<Meals, String>("name"));
typeTM.setCellValueFactory(new PropertyValueFactory<Meals, String>("type"));
costTM.setCellValueFactory(new PropertyValueFactory<Meals, Integer>("cost"));

listM = DB.getMeals();
tableM.setItems(listM);

numTD.setCellValueFactory(new PropertyValueFactory<Dranks, Integer>("id"));
nameTD.setCellValueFactory(new PropertyValueFactory<Dranks, String>("name"));
typeTD.setCellValueFactory(new PropertyValueFactory<Dranks, String>("type"));
costTD.setCellValueFactory(new PropertyValueFactory<Dranks, Integer>("cost"));

listD = DB.getDranks();
tableD.setItems(listD);
```

After that, the id of the table will be read the IDs of all objects of Drinks and Meals. listM -object from list data type from class Meal-, it will be read all the obects data column by column. And the setItem is a function which will assign the items the items to the table -item which is the cell or the intersection of the row and the column-.

## modifying — updating into Data Base

The following sql code is to add into the Data Base

```
lic static boolean updateD(String where, String name, String type, int cost){
 Connection con = cannent();
// String sql;
 //sql=;
 try {
      PreparedStatement ps = con.prepareStatement("update Dranks set nameD='"+name+"',
          typeD='"+type+"',costD='"+cost+"'"+where);
      return !ps.execute();
 } catch (Exception e) {
      JOptionPane.showMessageDialog(null, "
                                                 ");
 }finally{
     try {
         con.close();
     } catch (Exception e) {
     }
 }
return true;
```

So, we have two previous source code. The first one, is to update into menu. And the second is to update into the data base.

### 2.1.3 Delete new Item


Figure 2.3: **Deleting a new item**

By simply, when we indexed the row the function will deleteDarks() will know the row number as an indexed

```
ic void deleteDarks() {
if (indexD == -1) {
    return;
}
if (DB.deleteD("numD=" + numTD.getCellData(indexD))) {

    doneD.setText("                  ");
    doneD.setVisible(true);
    numDranks.setText(Integer.parseInt(numDranks.getText()) + -1 + "");
    listD.remove(indexD);
    indexD = -1;
    clearD();
}
```

## Deleting from data base

After adding the object we were talking about. Now the role to make all the objects appear in the our interface. So, the follwing codes explain how the all the objects will be apperaed.

```
lic static boolean deleteM(String where){
 Connection con = cannent();

 try {

     PreparedStatement ps = con.prepareStatement("delete from Meals where "+where);
           return !ps.execute();

 } catch (Exception e) {

 }finally{
     try {
         con.close();
     } catch (Exception e) {
     }
 }
return true;
```

## modifying — updating into Data Base

The following sql code is to add into the Data Base

```
lic static boolean updateD(String where, String name, String type, int cost){
 Connection con = cannent();
// String sql;
 //sql=;
```

```
 try {
      PreparedStatement ps = con.prepareStatement("update Dranks set nameD='"+name+"',
          typeD='"+type+"',costD='"+cost+"'"+where);
      return !ps.execute();
 } catch (Exception e) {
      JOptionPane.showMessageDialog(null, "
                                                  ");
 }finally{
     try {
         con.close();
     } catch (Exception e) {
     }
 }
return true;
```

So, we have two previous source code. The first one, is to update into menu. And the second is to update into the data base.

### 2.1.4 Search new Item

# Searching into Data Base


Figure 2.4: **Searching a new item**

```
c void searchMeals() {
searchM.textProperty().addListener(new InvalidationListener() {
    @Override
    public void invalidated(Observable o) {
        if (searchM.textProperty().get().isEmpty()) {
            tableM.setItems(listM);
            return;
        }

        ObservableList<Meals> tableitems = FXCollections.observableArrayList();
        ObservableList<TableColumn<Meals, ?>> cols = tableM.getColumns();
        for (int i = 0; i < listM.size(); i++) {
            for (int j = 0; j < cols.size(); j++) {
                TableColumn col = cols.get(j);
                String cellValue = col.getCellData(listM.get(i)).toString();
                cellValue = cellValue.toLowerCase();
                if (cellValue.contains(searchM.getText().toLowerCase()) && cellValue.
                    startsWith(searchM.getText().toLowerCase())) {
                    tableitems.add(listM.get(i));
                    break;
                }
            }

        }
        tableM.setItems(tableitems);
    }
});



ic void searchDranks() {
try {
    searchD.textProperty().addListener(new InvalidationListener() {
        @Override
        public void invalidated(Observable o) {
            if (searchD.textProperty().get().isEmpty()) {
                tableD.setItems(listD);
                return;
            }

            ObservableList<Dranks> tableitems = FXCollections.observableArrayList();
            ObservableList<TableColumn<Dranks, ?>> col = tableD.getColumns();
            for (int i = 0; i < listD.size(); i++) {
                for (int j = 0; j < col.size(); j++) {
                    TableColumn coll = col.get(j);
                    String cellValue = coll.getCellData(listD.get(i)).toString();
                    cellValue = cellValue.toLowerCase();
                    if (cellValue.contains(searchD.getText().toLowerCase()) &&
                        cellValue.startsWith(searchD.getText().toLowerCase())) {
```

```
                        tableitems.add(listD.get(i));
                        break;
                    }
                }

            }
            tableD.setItems(tableitems);
        }
    });

} catch (Exception e) {

}
```