

**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Assignment Cover Letter**

**(Individual Work)**

|  |
| --- |
| **Student Information: Surname Given Name Student ID number**   1. **Vohra Naman 2201798420** |
| **Course Code : COMP6510 Course Name : Programming Language** |
| **Class : B2BC Name of Lecturer(s) : Minaldi Loeis** |
| **Major : Computer Science** |
| **Title of Assignment : Java Final Project Report (Hotel Management System)**  **(if any)** |
| **Type of Assignment : Final Project Report** |
| **Submission Pattern**  **Due Date : 01-07-2019 Submission Date : 01-07-2019** |

**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)

1 Naman Vohra

1. **DESCRIPTION**

This is a hotel management system project. It is designed to provide the owner or hotel staff the ability to manage (insert, delete, edit) and preserve data. For example, the staff can insert the customer details when the customer checks in, delete a dish information from menu or edit the price of a room, etc. Data can also be preserved with the help of database. Furthermore, the implementation of GUI makes the program user friendly.

**II. Explanation of Functions**

**Main\_Window.java**

* **Outside Main\_Window Class**

-Importing libraries such as java.sql.\*, java.text.SimpleDateFormat, java.util.ArrayList, java.util.logging.Level, java.util.logging.Logger, javax.swing.JOptionPane, javax.swing.table.DefaultTableModel

* **Inside Main\_Window Class**

public Main\_Window()

**-**initComponents() is a method used to initialise components (setting the default values, etc).

**--**setExtendedState(java.awt.Frame.MAXIMIZED\_BOTH) is to maximize the window automatically whenever the user runs this file

public Connection getConnection()

**--**Uses the given database URL to establish connection to the database. If no problem exists, the function will return the connection to URL, else it will return null.

public Boolean inputCheck()

**--**Checks whether the necessary text fields (id, first name, last name, address, etc) are filled by the user. If filled, the function will return true else it will return false.

public ArrayList<CheckInDetails> getCheckInDetails()

**--**This method fills an ArrayList with objects of type CheckInDetails (which requires 10 parameters to be filled when creating its object; id, first\_name, last\_name, address, contact\_no, personal\_id\_no, room\_type, room\_no, check\_in\_date, check\_out\_date, and total\_cost) and then returns it. The SQL query = “SELECT \* FROM checkindetails” will select all the data present in this table. To achieve this using Java, Statement and ResultSet objects need to be created. createStatement() creates Statement object which is used to send SQL statements to the database. executeQuery() returns ResultSet object. ResultSet object contains the data retrieved (from database) which we get by executing the query. The data stored in this object can be accessed and retrieved through a set of get methods (getInt, getString, etc).

**--**ResultSet has a cursor that points to current row of data. The next method moves the cursor to the next row and returns true if there is row in the table in database. It returns false if there is no row left in the table. With the help of while, we can iterate from the first row to the last row and store the data in each row in CheckInDetails object. We then append this object to ArrayList.

**--**This ArrayList (cIn\_details\_list) is then returned.

public void ShowCheckInDetailsInTable()

--As the method name suggests, it fills the JTable (in this case cInDetails) with the desired data from database.

**--**A new ArrayList<CheckInDetails> object (named list) is created to store the ArrayList which is returned from the getCheckInDetails() function. An object model of type DefaultTableModel is created. This refers to cInDetails JTable.

**--**An array (named row) of type Object and size 11 is created. For loop iterates through list (which stores CheckInDetails object). CheckInDetails variables are accesed using getter methods which are defined in CheckInDetails class. Each info is then stored in a position in row. For example: list.get(i).getId() returns Id which will be stored in row[0], list.get(i).getFirstName() returns first name which will be stored in row[1] and so on. After filling all the 10 data in row, we add the row to JTable (model). This process is repeated until there is no more row left in the table in database.

public ArrayList<Transactions> getTransactionsDetails()

**--**Uses the same concept as in function ArrayList<CheckInDetails> getCheckInDetails(), but the difference is that this method deals with Transactions data type instead of CheckInDetails.

public void ShowTransactionDetailsInTable()

**--**Uses the same concept as in ShowCheckInDetailsInTable() but deals with Transactions objects and the data is shown in JTable transaction\_list.

private void totalCostActionPerformed(java.awt.event.ActionEvent evt)

If the user clicks **Total Cost** button:

**--**Calculates each of the customer’s bill.

**--**PreparedStatement object is used two times in this function. It represents a precompiled SQL statement meaning that the SQL statement is compiled in the initial run itself thus making it able to be run multiple times without compiling it again and again. The first preparedStatement object stores and compiles the SQL *Statement “UPDATE checkindetails INNER JOIN room ON checkindetails.room\_no = room.room\_no SET checkindetails.total\_cost = TIMESTAMPDIFF(DAY, check\_in\_date, check\_out\_date) \* room.cost”*. This statement means that at first, the table checkindetails will be connected to room through a common field (room\_no) and the total\_cost field in table checkindetails will be given a value which is the number of days the customer stayed in hotel multiplied by the room cost (room cost depends on the room id and type).

**--**The second PreparedStatement object stores the SQL statement *“UPDATE checkindetails INNER JOIN (transactions INNER JOIN menu ON transactions.food\_id = menu.id) INNER JOIN room ON transactions.id = checkindetails.id AND checkindetails.room\_no = room.room\_no SET checkindetails.total\_cost = transactions.quantity \* menu.cost +checkindetails.total\_cost WHERE checkindetails.room\_no = room.room\_no AND transactions.food\_id = menu.id”*. This statement means that we update the table checkindetails which is connected to table transactions (notice that table transactions is connected to table menu through field food\_id (transactions) which is equated to field id (menu)) based on field id (transactions) which equals to field id (checkindetails) and room\_no (checkindetails) which is equated to room\_no (room). It then sets total\_cost field in checkindetails table to quantity (transactions) \* cost (menu) + total\_cost (checkindetails). Quantity (transactions) refers to the number of dishes the customer ordered, and cost is the price of the dish. Note that total\_cost (previous value in field) is added as SET resets the value and not update it. We don’t want the cost of the room to be not calculated in the bill.

**--**ps.executeUpdate() and ps2.executeUpdate() simply executes the SQL statements.

**--**An object named cInT (refers to JTable which stores the check in details) of type DefaultTableModel is created and the data inside it cleared (cInT.setRowCount(0)) before ShowCheckInDetailsInTable() is called. The reason is that we don’t want the same data from table checkindetails to be showed again in the JTable every time the user clicks the button **Total Cost** and that we want the user to be able to view the updated version of data.

private void INSERTActionPerformed(java.awt.event.ActionEvent evt)

If the user clicks **Check In** button:

**--**Inserts check in details into the checkindetails table in database. Updates the availability field in a specific record in table room. The record which will be updated depends on the room number which is chosen by the customer. For example, if the customer chooses room number 1 it means the availability field in the record where room\_no = 1 will be changed from ‘A’ (short term for Available) to ‘NA’ (Not Available). Inserts into field id (transactions table) the id of the new customer (same as id in checkindetails table).

**--**Uses .prepareStatement and .executeUpdate to run the SQL statements just as in function **private void totalCostActionPerformed(java.awt.event.ActionEvent evt)** (For explanation of these methods check this function). The only difference is that customer inputs are needed here, thus keyword VALUES(?, ?, ?, ?, ?...so on…) is seen in the SQL statements. For example *INSERT INTO checkindetails (first\_name, last\_name, address, contact\_no, personal\_id\_no, room\_type, room\_no, check\_in\_date)* ***VALUES (?, ?, ?, ?, ?, ?, ?, ?).*** The VALUES command specifies the values that the user wants to insert into table using the INSERT INTO keyword. For example, ps.setString(1, fname.getText()) takes the first name from fname text field and stores it to first\_name text field in table checkindetails. 1 refers to first\_name as first\_name is the first field into which value is going to be inserted.

**-**-SimpleDateFormat is used to parse and format the date according to the pattern given by the user.

**--**Similar to function **totalCostActionPerformed(java.awt.event.ActionEvent evt)** an object named cInT (refers to JTable which stores the check in details) of type DefaultTableModel is created and the data inside it cleared (cInT.setRowCount(0)) before ShowCheckInDetailsInTable() is called. One thing is added here, ShowTransactionDetailsInTable() which implements same mechanism as in ShowCheckInDetailsTable() except that it deals with Transactions object and shows the transaction details (customer id, food id, and quantity) in transactions table.

private void menu\_and\_roomActionPerformed(java.awt.event.ActionEvent evt)

**--**Whenever the user clicks the button Menu and Room, a new window will pop up. This window contains information about menu and food.

**--**We create an object of type MenuandRoom (JFrame that displays information about menu and room) and the make this object visible (second\_frame.setVisible(true)).

private void refreshActionPerformed(java.awt.event.ActionEvent evt)

**--**Clears table cInT and transactions before calling ShowCheckInDetailsInTable and ShowTransactionDetailsInTable().

private void rtActionPerformed(java.awt.event.ActionEvent evt)

**--**Shows all the room number of specific type (according to customer input) in the rn combo box. For example, if customer wants a hotel room of type Single then only room numbers available of this type will show in up rn combo box, which is a combo box to store room numbers. The SQL statement is "SELECT room\_no FROM room WHERE room.availability = 'A' AND room.room\_type = ?".

**--**As usual, this is achieved by using PreparedStatement and ResultSet object. ResultSet retrieves the values from database and add it to combo box. Before that all the values in rn combo box are cleared.

private void cOut2ActionPerformed(java.awt.event.ActionEvent evt)

If the user clicks the Check Out button:

**--**Fills the check\_out\_date field which was null earlier in checkindetails table. (*UPDATE checkindetails SET check\_out\_date = ? WHERE checkindetails.id = ?"*).

**--**Changes the room’s availability (the room where the customer stayed) from ‘NA’ to ‘A’ (*UPDATE checkindetails INNER JOIN room ON checkindetails.room\_no = room.room\_no SET room.availability = 'A'*). If the customer left the room, it will be available for use.

**--**ShowCheckInDetailsInTable() is called. Before that cInT (JTable which stores check in details) is cleared.

**MenuandRoom.java**

* **Outside MenuandRoom Class**

**--**Importing libraries such as java.sql.\*, java.text.SimpleDateFormat, java.util.ArrayList, java.util.logging.Level, java.util.logging.Logger, javax.swing.JOptionPane, javax.swing.table.DefaultTableModel and java.awt.Color.

* **Inside MenuandRoom Class**

public MenuandRoom()

**---**initComponents() is a method used to initialise components (setting the default values, etc)

**--**getContentPane().setBackground(Color.cyan) sets the JFrame background colour to cyan.

public Connection getConnection()

**--**Exactly the same as the function public Connection getConnection() in Main\_Window class.

public boolean inputCheck()

**--**Returns true if all the necessary text fields are filled by the user. Else return false.

public ArrayList<RoomInfo> getRoomDetails() and public ArrayList< MenuInfo> getMenuDetails()

**--**Uses the same mechanism as in function public ArrayList<CheckInDetails> getCheckInDetails() but this one deals with RoomInfo and MenuInfo objects respectively.

public void ShowRoomDetailsInTable() and public void ShowMenuDetailsInTable() **--**Uses the same concept as in ShowCheckInDetailsInTable() but deals with deals with RoomInfo and MenuInfo objects and the data is shown in JTable rInfos (which stores room details) and JTable menus (which stores details about menu).

private void main\_pageActionPerformed(java.awt.event.ActionEvent evt) If the user presses the Main Page button:

**--**It will close this window (MenuandRoom) (this.setVisible(false)).

private void purchaseActionPerformed(java.awt.event.ActionEvent evt)

If the user clicks the purchase button:

private void room\_editcostActionPerformed(java.awt.event.ActionEvent evt) If Edit Cost button is pressed:

**--**Firstly, the user have to fill the room\_cost and room\_no text fields. Room\_cost refers to the cost of the room and room\_no means the room which the user wants the price to be changed.

**--**If the necessary text fields are filled, this SQL statement will be executed: *"UPDATE room SET room.cost = ? WHERE room.room\_no = ?".* The SQL statement will now update table menu and change the value of field quantity in a record where the id is specified by the user.

--Calls ShowRoomDetailsInTable which fills the rInfos table with the updated data from database. But the JTable is cleared before.

private void room\_insertActionPerformed(java.awt.event.ActionEvent evt) **--**If the owner wants to insert details of a new room, then he should make sure that he fills the room\_type (refers to the type of the room), room\_cost (refers to the cost of the room), and room\_availability text fields (refers to the status of room, for example, whether it is available or not).

--If all the required text fields are filled and the Insert button is pressed, the following SQL statement will be executed: *“INSERT INTO room(room\_type, cost, availability) VALUES (?, ?, ?)".* It means that it will insert values (according to user inputs) into fields room\_type, cost, and availability which is in table room.

**--**Calls ShowRoomDetailsInTable which fills the rInfos table with the updated data from database. But the JTable is cleared before function ShowRoomDetailsInTable is called.

private void refreshActionPerformed(java.awt.event.ActionEvent evt) **--**Calls ShowRoomDetailsInTable which fills the rInfos table with the updated data from database. But the JTable is cleared before function ShowRoomDetailsInTable is called.

private void purchaseActionPerformed(java.awt.event.ActionEvent evt)

**--**The necesarry text fields which must be filled are checkIn\_id (refers to id given to the customer when the customer checks in), menu\_id (refers to the food that the customer wants to order) and menu\_quantity (the number of dish that the customer wants to order).

--If the user clicks the purchase button, the following SQL statements will be executed in the following order:

*1.) "SELECT quantity FROM menu WHERE id = ?"*

*2.) "UPDATE menu SET quantity = ? WHERE id = ?"*

*3.) "UPDATE transactions SET food\_id = ?, quantity = ? WHERE id = ?"*

The first one selects the quantity of the selected food. The value is then retrieved with the help of ResultSet object. We need this value as we want to make sure that the quantity of food ordered by the customer does not exceed the amount of food available. The second one updates the quantity of the food chosen by the owner. This is needed as the available quantity of the food decreases whenever customer orders it. The third one updates the food\_id and quantity fields (in table transactions) of a record which represents the customer who ordered the food. The reason that another table is to avoid redundant data. For example, details of food are not required to be repeated again as it can be represented by id (food\_id in transactions table is equated with the menu\_id in table menu).

**--**Calls ShowMenuDetailsInTable which fills the menus table with the updated data from database. But the JTable is cleared before function ShowMenuDetailsInTable is called.

private void refresh2ActionPerformed(java.awt.event.ActionEvent evt)

**--**Calls ShowMenuDetailsInTable which fills the JTable menus with the updated data from database. But the JTable is cleared before function ShowMenuDetailsInTable is called.

private void menu\_insertActionPerformed(java.awt.event.ActionEvent evt)

**--**Calls the inputCheck() function to make sure that all the required text fields are filled. The text fields are menu\_id (refers to the food name), menu\_foodname (the food name), menu\_cost (the price of the food), and menu\_quantity ( In this case, the quantity of the new food that is going to be inserted).

--The SQL statement used in this function: *“INSERT INTO menu(food\_name, quantity, cost) VALUES(?, ?, ?)”*. If the user clicks the INSERT button, this statement will be executed. It simply inserts the name, quantity, and cost of the food to the menu table.

private void menu\_deleteActionPerformed(java.awt.event.ActionEvent evt)

--In order for the SQL statement in this function to be executed, the menu\_id text field should not be null.

**--**The following SQL statement will be executed: *"DELETE FROM menu where id = ?".* It simply deletes the record with id specified by user (In other words, deletes the dish details that the owner wishes to delete).

--Calls ShowMenuDetailsInTable which fills the JTable menus with the updated data from database. But the JTable is cleared before function ShowMenuDetailsInTable is called.

private void menu\_editCostActionPerformed(java.awt.event.ActionEvent evt)

**--**Edits the price of a particular dish. In order for the following SQL statement *(“UPDATE menu SET menu.cost = ? WHERE menu.id = ?”)* to be executed, the tex fields menu\_cost and menu\_id should not be null.

--Calls ShowMenuDetailsInTable which fills the JTable menus with the updated data from database. But the JTable is cleared before function ShowMenuDetailsInTable is called.

private void menu\_editQuantityActionPerformed(java.awt.event.ActionEvent evt) **--**Edits the quantity of a particular dish in the menu. In order for the following SQL statement *(“UPDATE menu SET menu.quantity = ? WHERE menu.id = ?”)* to be executed, the text fields menu\_quantity and menu\_id should not be null.

--Calls ShowMenuDetailsInTable which fills the JTable menus with the updated data from database. But the JTable is cleared before function ShowMenuDetailsInTable is called.

**MenuInfo.java**

* **Inside MenuInfo**

Public Members:

Id: int

Food\_name: String

Quantity: int

Cost: int

public int getId()

**--**returns id

public String getFoodName()

**--**returns food\_name

public int getQuantity()

**--**returns quantity

public int getCost()

**--**returns cost

**Transactions.java**

* **Inside Transactions implements Transaction**

Public Members:

Id: int

Food\_id: int

Quantity: int

public int getId()

**--**returns id

public String getFoodId()

**--**returns food\_name

public int getQuantity()

**--**returns quantity

**RoomInfo.java**

* **Inside Transactions**

Public Members:

Room\_no: int

Food\_id: String

Cost: int

Availability: String

public int getRoomNo()

**--**returns room\_no

public String roomType()

**--**returns room\_type

public int getCost()

**--**returns cost

public String getAvailability()

**--**returns availability

**CheckInDetails.java**

* **Inside Transactions**

Public Members:

Id: int

First\_name: String

Last\_name: int

address: String

contact\_no: int

personal\_id\_no: int

room\_type: String

room\_no: int

check\_in\_date: String

check\_out\_date: String

total\_cost: int

public int getId()

**--**returns id

public String getFirstName()

**--**returns first\_name

public String getLastName()

**--**returns last\_name

public String getAddress

**--**returns address.

public int getContactNo()

**--**returns contact\_no

public int getPersonalIdNo()

**--**returns personal\_id\_no

public String getRoomType()

**--**returns room\_type

public String getRoomNo()

**--**returns room\_no

public String getCheckInDate()

**--**returns check\_in\_date

public String getCheckOutDate()

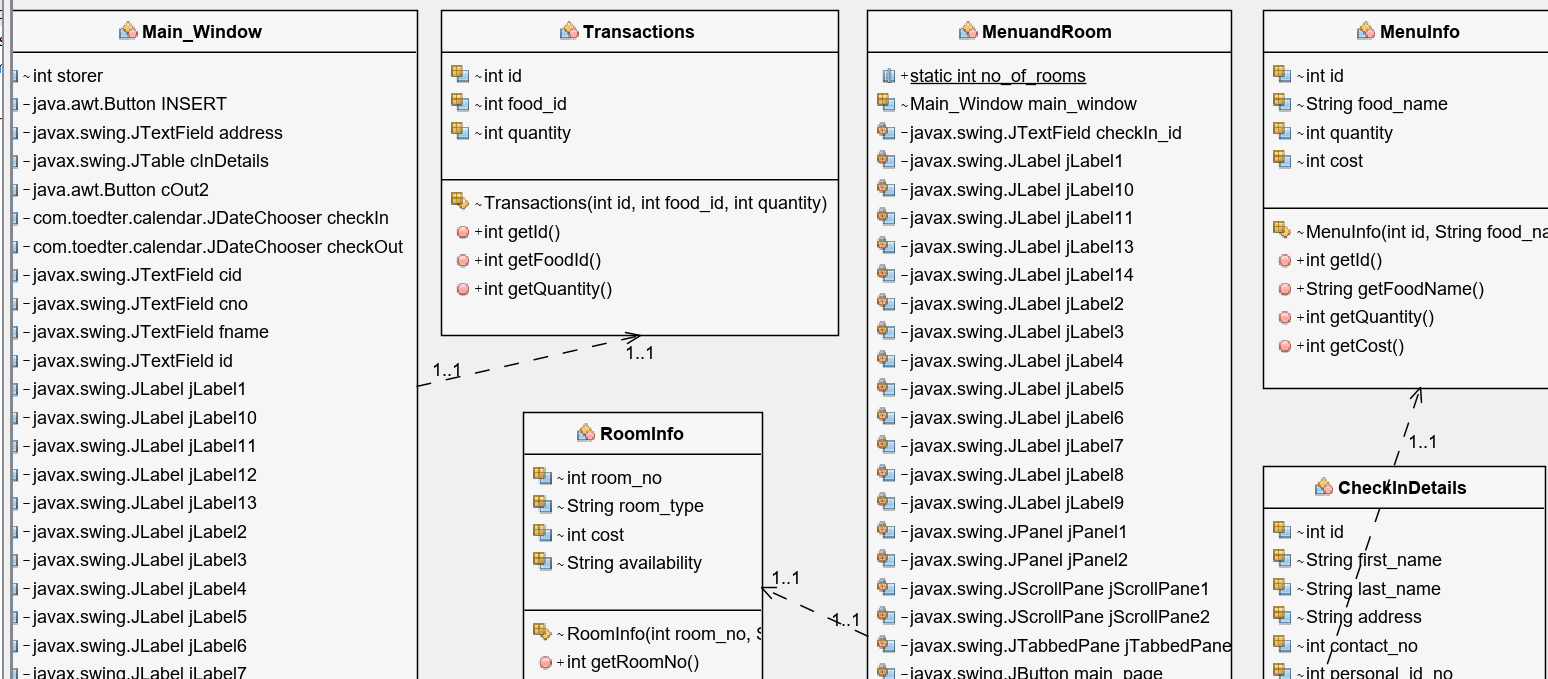
**--**returns check\_out\_date

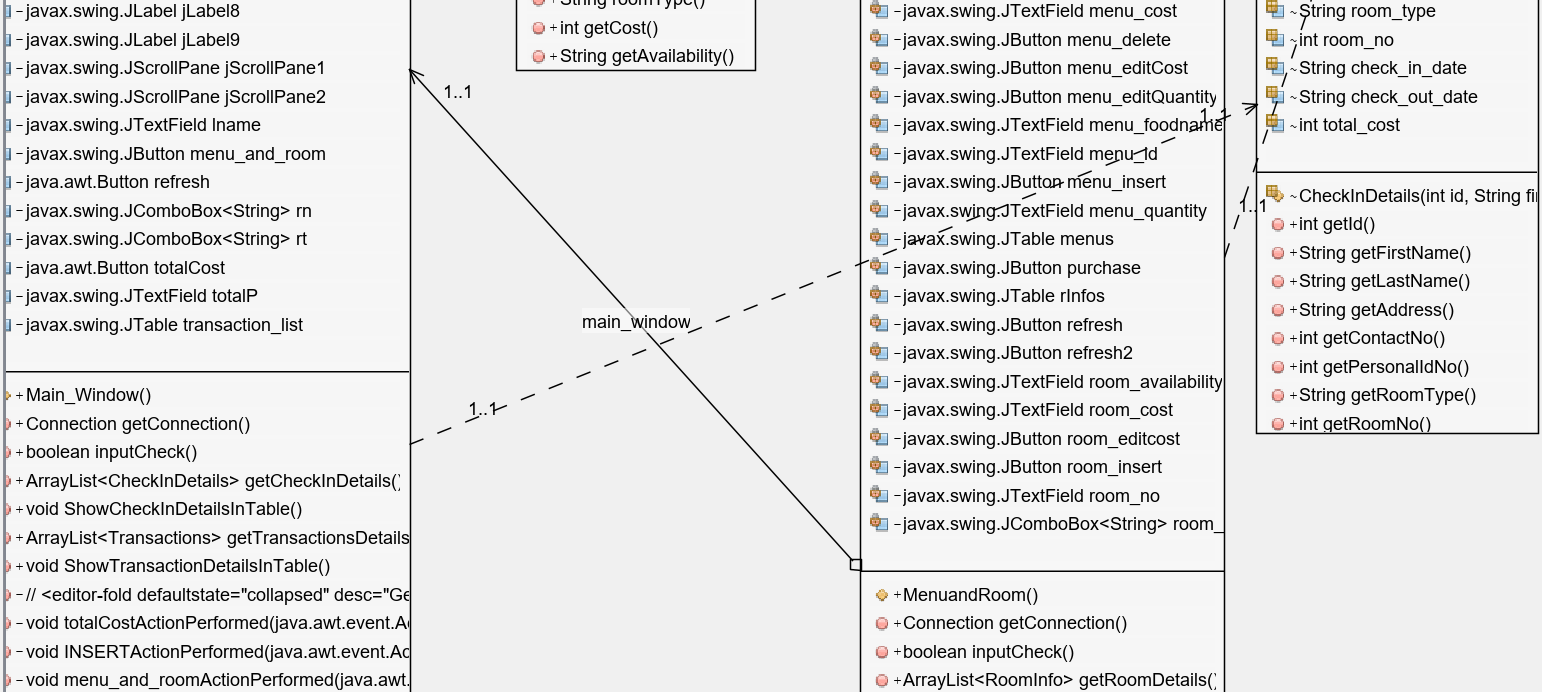
public String getTotalCost()

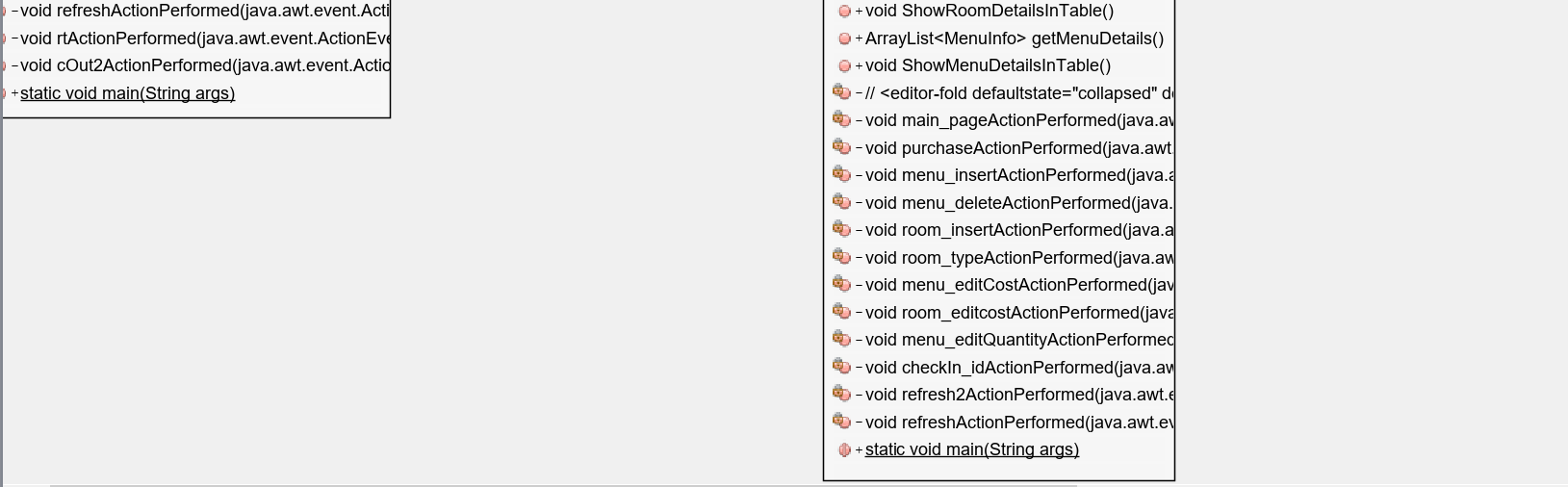
**--**returns total\_cost

1. **Diagrams**

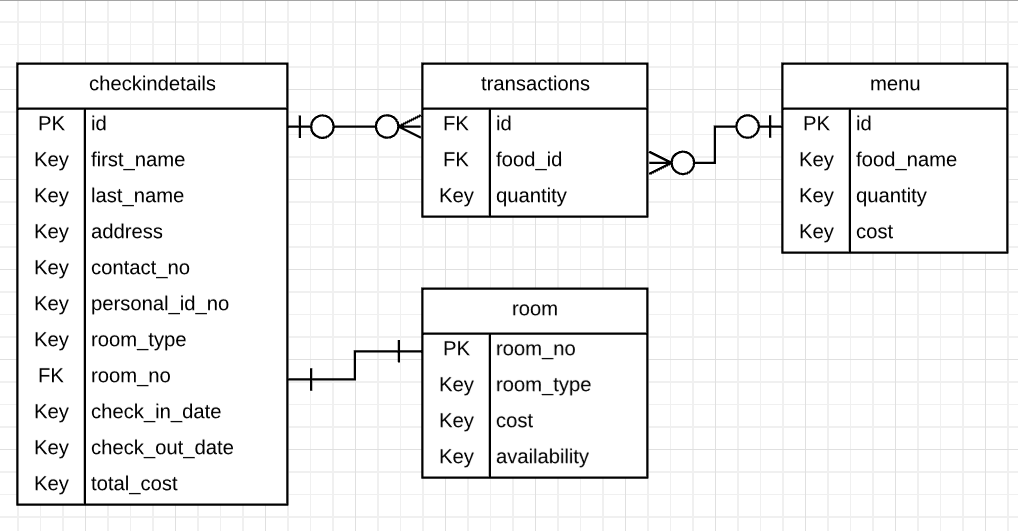
**III. a. CLASS DIAGRAM**







1. **b ER Diagram**



**IV. Lessons That have Been Learned**

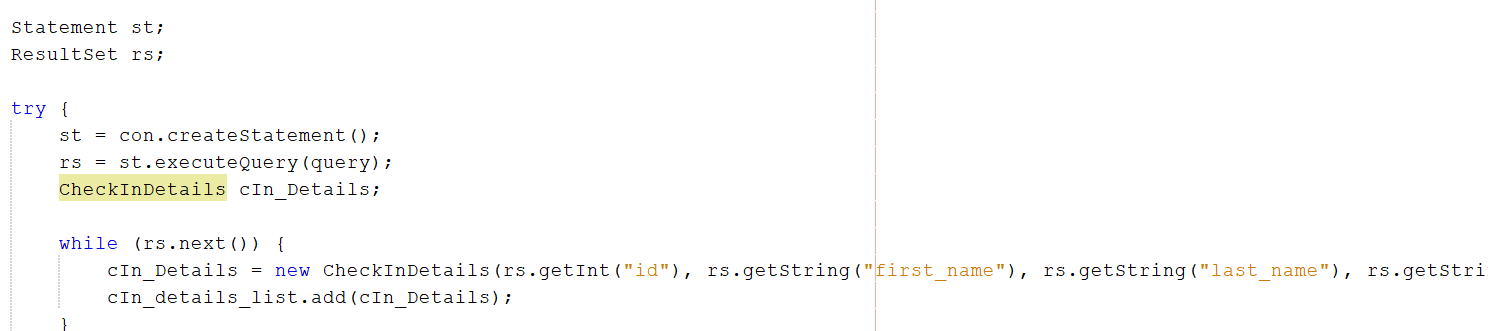
1. **Establishing a Connection to the Database**



The **DriverManager** class has a function called the **getConnection(String url)** which establishes connection / connects Java to the database by using the given database URL.

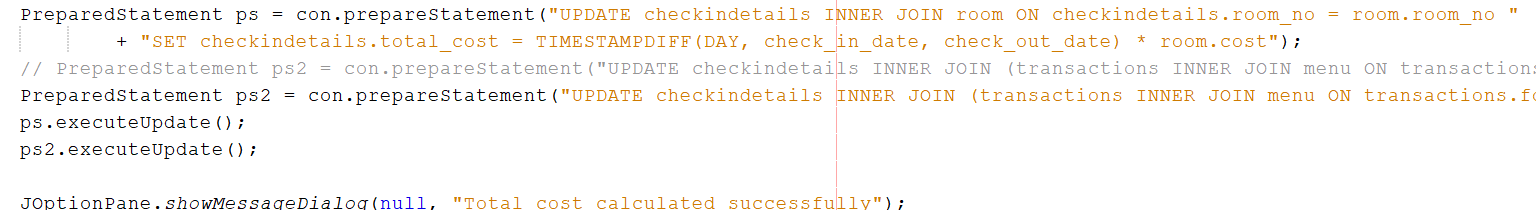
This is the first step which is needed to acquire the ability to carry out operations like insert, delete, edit, etc with the tables of the hotel management system database.

1. **The Use of Statement and ResultSet Objects to Retrieve Data from Database**



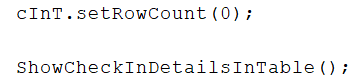
The .createStatement() method returns a Statement object which is needed to execute SQL queries (st.executeQuery(query)). The ResultSet object contains the data which results from executing the SQL query. In the above picture, the objects are being used to retrieve data from the checkindetails table so that later it can be shown in the JTable (The user do not have to switch between netbeans and mySQL to look at updates, etc).

1. **Using Inner Join to Connect Tables and Carry Out Operations**



Inner join can be used to combine two tables if there is a certain field which exists in both of the tables (a common field). Now let us look and understand the SQL statement in the above picture: “UPDATE checkindetails INNER JOIN room ON checkindetails.room\_no = room.room\_no SET checkindetails.total\_cost = TIMESTAMPDIFF(DAY, check\_in\_date, check\_out\_date) \* room.cost”). Firstly, we can see that the checkindetails is the table that is going to be updated. We then join the checkindetails to the room table **ON THE BASIS** **OF** a common field which exists in both of them; room\_no. The reason is that the table checkindetails contains only the id of the room (room\_no), but not the details. The details of the room are stored in table room. Thus, to gain access to those details, we need to use INNER JOIN and have a common field between the tables. Afterwards, we can see that the total\_cost field in table checkindetails will be given a value which is the the number of days the customer stayed in the hotel ((TIMESTAMPDIFF(DAY, check\_in\_date, check\_out\_date) \* room.cost) multiplied by the price of the room per night.

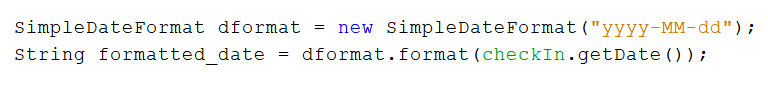
1. **The Use of Functions Like format(), removeAllItems(), and getModel()**



setRowCount(0) is used to clear the cInT (refers to the JTable which stores the check in details). This is used when buttons such as edit, purchase, etc is used (When such buttons are pressed, all data from a particular table will be retrieved and shown in its respective JTable). We want the JTable to store updated information. Moreover, we don’t want redundant records to be displayed in JTable. Hence, we clear the JTable’s contents first before calling ShowCheckInDetailsInTable (in this case).



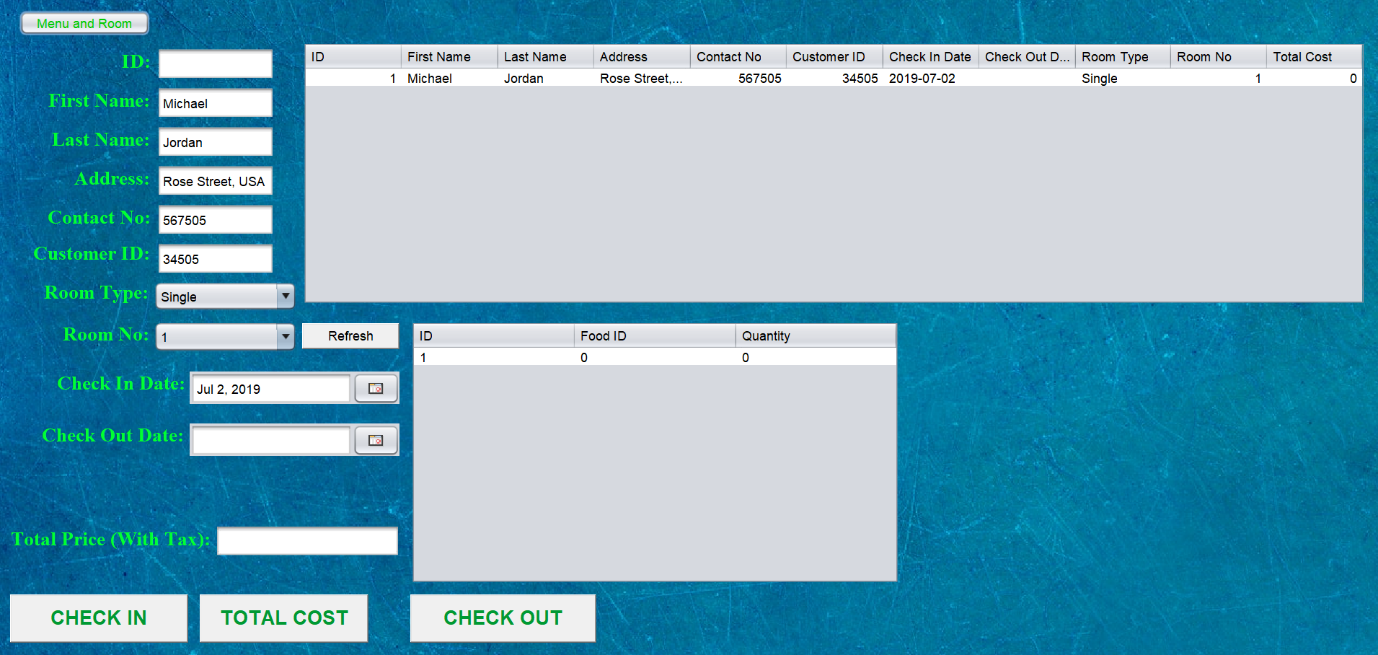
Removes all items in comboBox.



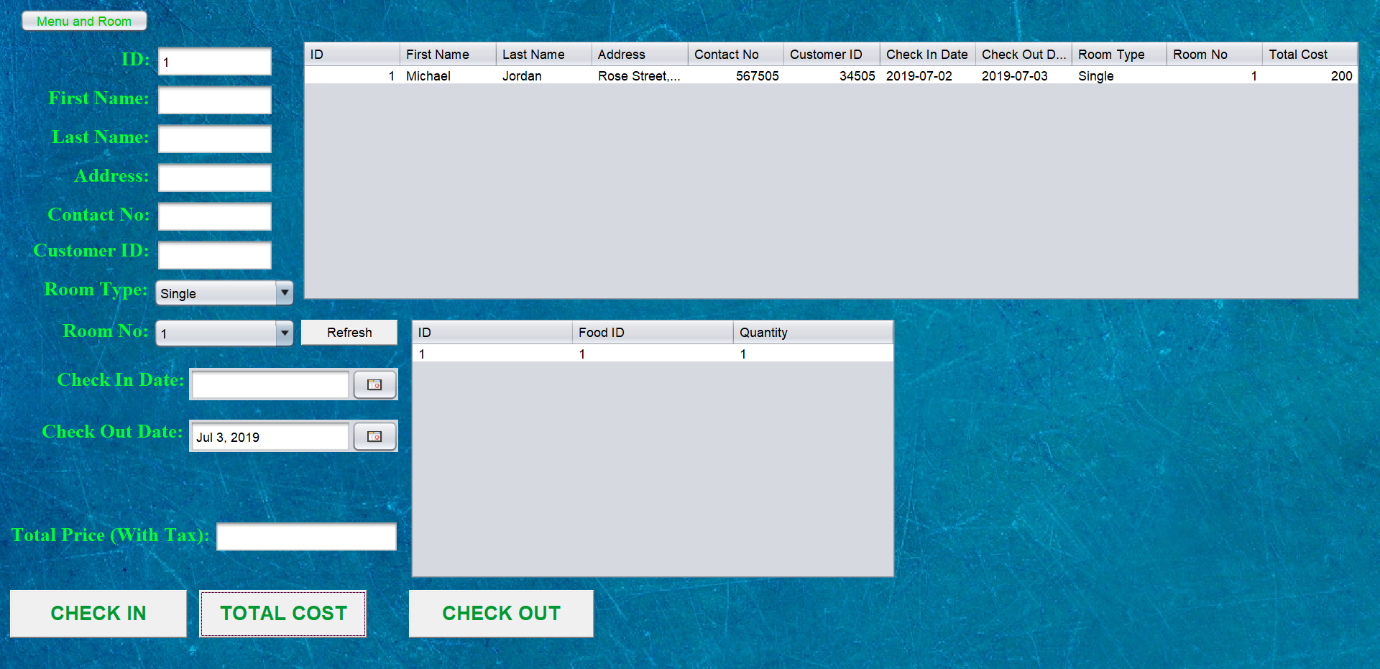
Used to format and parse the dates according to the pattern given (In this case, “yyyy-MM-dd”).

**V.Evidence of Working Program (Screenshots)**

**After pressing Check In button**



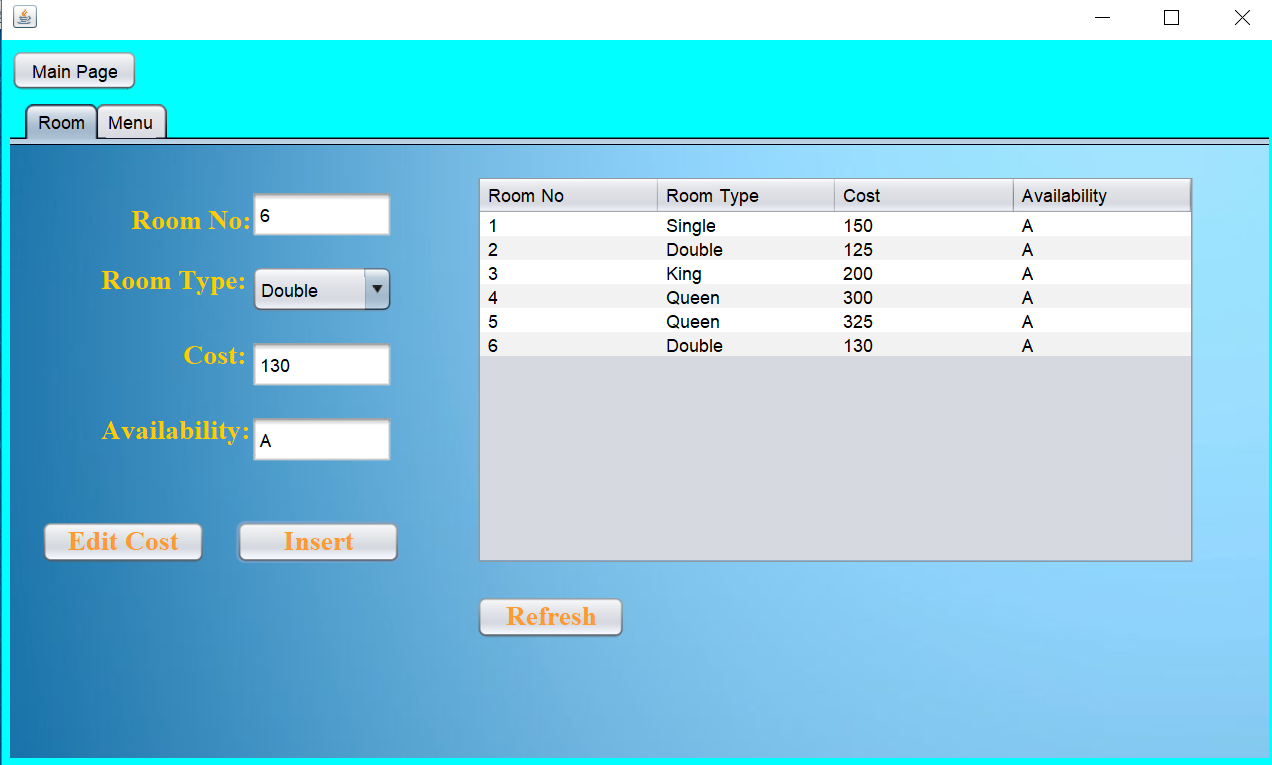
**After pressing Check Out button**



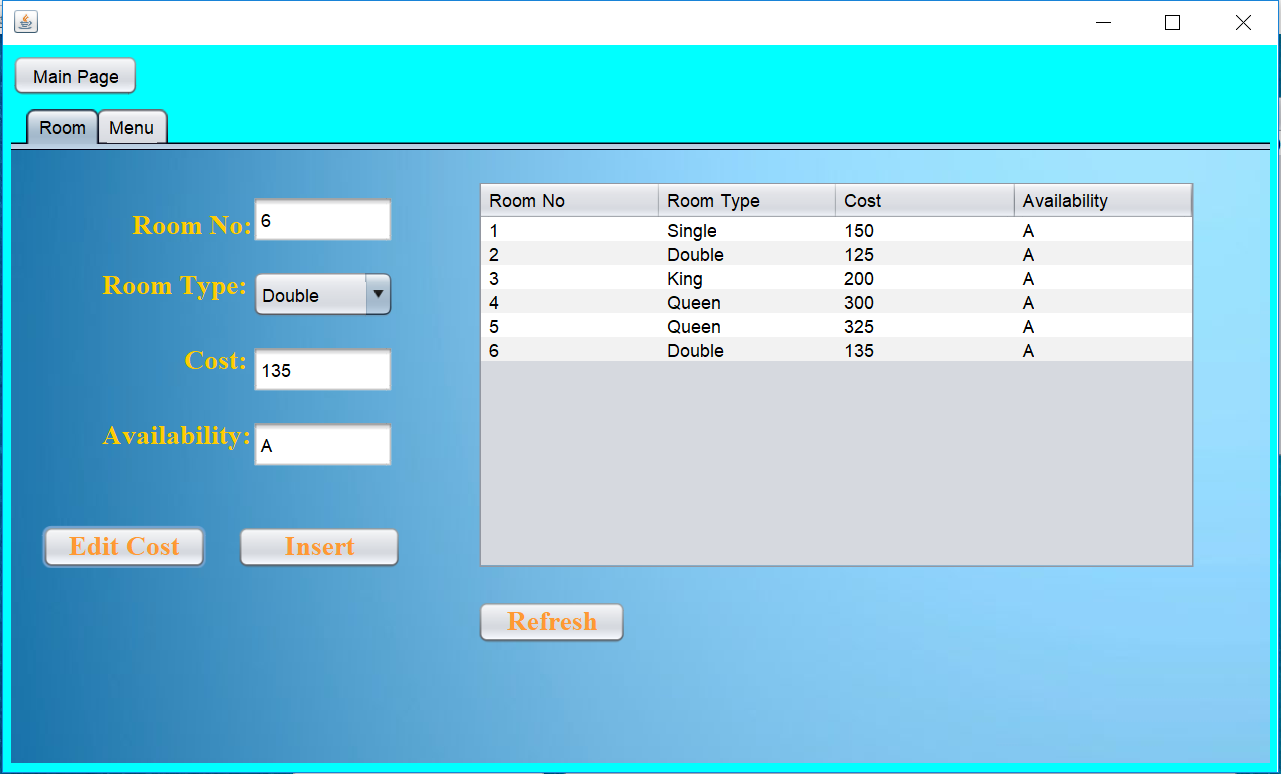
**After pressing Total Cost button**



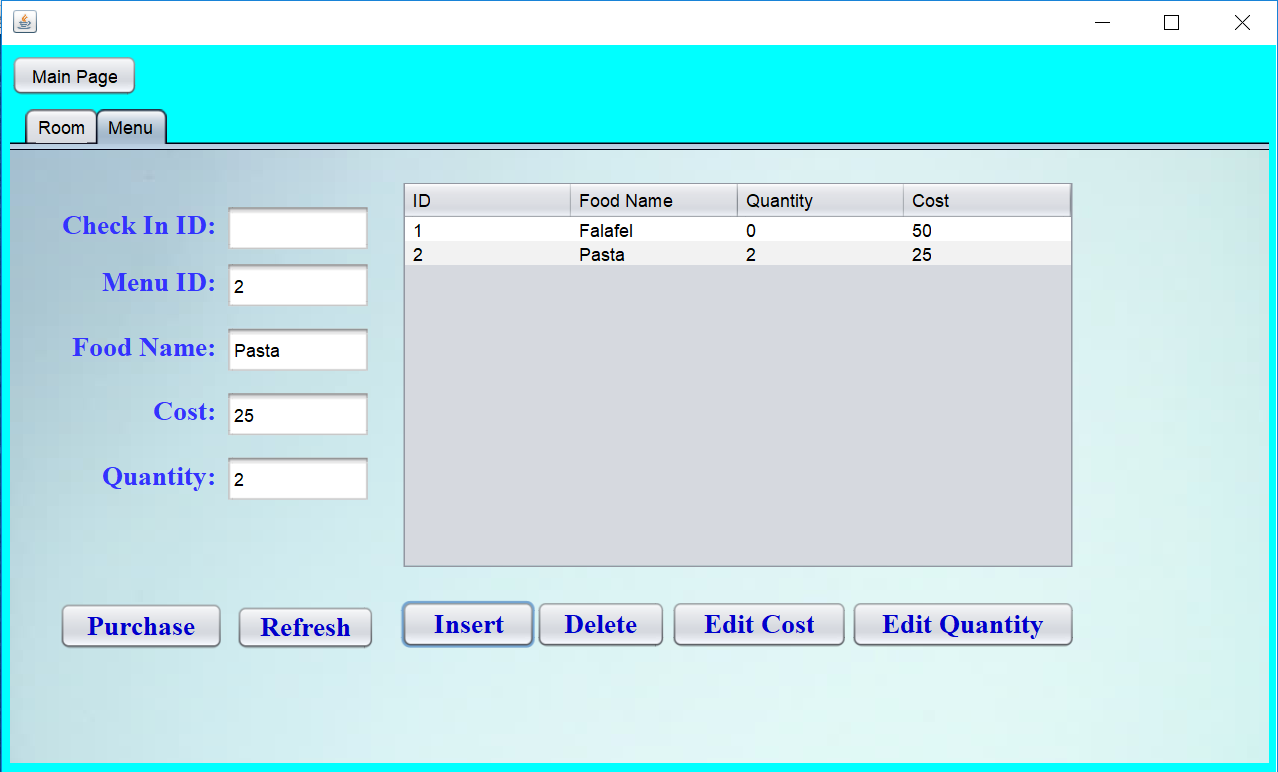
**After inserting new type of room (Pressing Insert Button)**



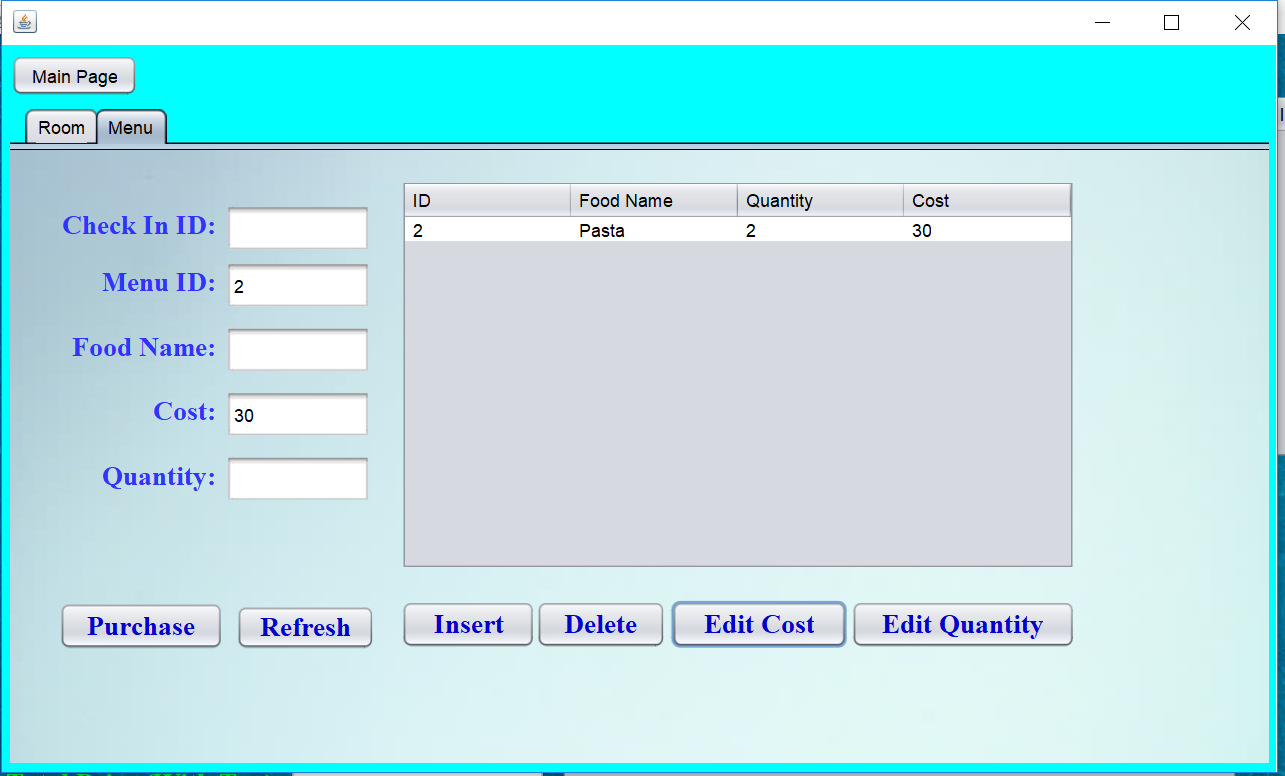
**Changing the price of the room inserted above (Pressing Insert Button)**



**Inserting a new dish (Pressing Insert Button)**



**Changing price and quantity of dish inserted above (Pressing Edit Cost then Edit Quantity Button)**





**VI. References**

[***https://www.javatpoint.com/java-drivermanager-getconnection-method***](https://www.javatpoint.com/java-drivermanager-getconnection-method)

[***http://tutorials.jenkov.com/java-internationalization/simpledateformat.html***](http://tutorials.jenkov.com/java-internationalization/simpledateformat.html)

<https://www.youtube.com/watch?v=vtTUKLE_SWE&t=106s>