INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

AAPP004-4-2-JP

JAVA PROGRAMMING

HAND OUT DATE: 10th JANUARY 2022

HAND IN DATE: 10th APRIL 2022

WEIGHTAGE: 60%

INSTRUCTIONS TO CANDIDATES:

1 **Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing).**

2 **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**

3 **Cases of plagiarism will be penalised.**

4 **The assignment should be bound in an appropriate style (comb bound or stapled).**

5 **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**

Student Name: LAI JUN JACK

Student TP Number: TP059540

Table of Contents

1.0 Introduction

This report is regarding the design and development of a resort room booking system made with stand-alone GUI application using JAVA concepts. This report will demonstrate the use of Java concept and their functionalities in the entire resort room booking system. The purpose of this resort room booking system is to simulate room booking in a week where it consists of two different type of room views (Jungle, Sea). Both type of views consists of 10 different rooms. A room is charged RM350.00 per night however, taxes will be charged differently to different type of citizenship. This room booking system will only be used by the resort staff.

In this report, we will cover how the Java application is built and ways to use the functions available like, login, adding, modifying, deleting, and searching. At the very end, only correct login credential allow user to access the system and every booking will be unique so it will not crash with one another. The system will provide data validation, for customer details and room booked for the certain day and time with automated price calculation. A receipt will be provided for every room booking made for the customer. The system will continue running until an exit command is issued. What's more, we will provide deep explanation to why and how we configure the code to be implemented into the system and additional features we have added. All concepts and error measures will be discussed in detailed.

2.0 Source Code

2.1 Header / Import Files

1. **import java.io.*;**

Java.io* imports the class that involves input and output functions that is used in the file operations. It adds all the Library/Package classes and methods into the java file for all input/output operations for all types of objects, data types, characters, and files to execute I/O operation.

2. **import java.text.SimpleDateFormat;**

It is a concrete class to format and parsing a date and time in Java. Able to change the format of time from date to string.

3. **import java.util.ArrayList;**

It is a class that allows resizable array in Java. Compared to built-in array in java, the built-in array size cannot be modified accordingly. Elements in ArrayList can be added and removed whenever.

4. **import java.util.Calendar;**

An abstract class that provides functions to convert and manipulate time and calendar field like year, month, day, hour, secs. Functions available like getting the date of any day with implementing calendar and defining the range of values.

5. **import java.util.Date;**

This class represent date and time in Java where constructions and methods are available to deal with date and time.

6. **import java.util.Scanner;**

Scanner class is to read user input with different methods available like nextLine() method by creating an object of the class to read Strings in data file.

7. **import javax.swing.JOptionPane;**

This Java API class is used to create dialog box to appear on the user's computer desktop to either be able to request input from or prompt a dialog box to display messages.

8. **import java.io.BufferedReader;**

This is a Java public class that have methods to read large amount of text input stream. It is usually used for demanding task like read large files and the buffering allows high efficiency for text readings.

9. **import java.io.File;**

It is an abstract representation of file and directory pathnames. It tells the location of external classes to the compiler.

10. **import java.io.FileNotFoundException;**

This class provides signal to the user when there is a failed attempt to open a file through a specified pathname.

11. **import java.io.FileReader;**

This class is used to read data characters from the specified files.

12. **import java.io.FileWriter;**

This class is used to write data characters to the specified files.

13. **import java.io.IOException;**

This class provides signal to the user when there is failure for input / output operation. For example, failure during writing, reading, searching file directory operation.

14. **import javax.swing.table.DefaultTableModel;**

This class is to import the table model that store the cell value objects where it returns column and rows classes of the object.

15. **import java.util.Scanner;**

This class is to read string input, int input, double input in Java.

16. **import java.awt.event.WindowEvent;**

This class prompt interface and classes to deal with different type of event fired by AWT component.

17. import javax.swing.ListSelectionModel;

This class provide List selection model interface to enable the input methods that is used during Java runtime.

18. import javax.swing.RowFilter;

This class is used to filter out the table model entries from every row that will not be shown in the view.

19. import javax.swing.table.TableRowSorter;

This class is an implementation of row sorter for table that provides function for sorting and filtering result from a table model.

2.2 Variables

```
//Create date variable to get time.
Date mindate = cal.getTime();
Date maxdate = cal.getTime();
```

Figure 2.2.1 Date variable

mindate and maxdate are date variables with date format that are created to allow input from the system through calendar format with cal.getTime() function. Date data is considered a class in Java that includes time, year, name of the day of the week and the time zone.

```
//Get input from textbox
String name = jtxtname.getText();
String citizenship = jtxtcitizenship.getText();
String ic = jtxtic.getText();
String contact = jtxtnumber.getText();
String email = jtxtemail.getText();
String roomtype = jtxtroomtypel.getText();
String roomid = jtxtroomid.getText();
String price = jtxtprice.getText();
```

Figure 2.2.2 String variable

Variable string is created to allows user input from the Java application. For example jtxtname.getText() is to receive text from the system and convert into a string under the variable name.

```
//Set date format
SimpleDateFormat DateFormat = new SimpleDateFormat("yyyy/MM/dd");
//Get start and end date entered by the users.
String startdate = DateFormat.format(jtxtstartdate.getDate());
String enddate = DateFormat.format(jtxtenddate.getDate());
```

Figure 2.2.3 String Date variable converted to String

String variables like startdate and enddate is created to convert date format input to string from the system.

```
//Validation Details with input
int error = Booking.validatebooking(jtxtname.getText(), jtxtic.getText(), jtxtnumber.getText(), jtxtemail.getText(), jLabel3.getText());
```

Figure 2.2.4 Integer variable

Integer variable is created to store values from the function made. Each element in an array of type int is a variable of type int where it stores integers without decimals.

```
//List out arraylist from booking start and end date arraylist.
ArrayList<String> bookingstartdate = LoadData.bookingstartdate();
ArrayList<String> bookingenddate = LoadData.bookingenddate();
```

Figure 2.2.5 ArrayList String variable

String bookingstartdate and bookingenddate variables created with the class function of ArrayList where it stores the data accordingly from the function enable.

```
//Set date format to string
Date startdate = null, enddate = null;
```

Figure 2.2.6 Date variable to string variable

Set date format variable into string variable with null.

```
public static int validatebooking(String name, String idnum, String connum, String email, String idtype) {
    int error = 0;
    //error 1 = input error
    //error 2 = IC number invalid
    //error 0 = no error

    //Data validation for name, contact number, email
    if (name.length() < 2 || name.length() > 50 || name.matches("[+-]?\\d*(\\.\\d+)?") == true
            || connum.length() < 10 || connum.length() > 11 || connum.matches("[+-]?\\d*(\\.\\d+)?") == false
            || email.contains("@") == false || email.contains(".com") == false) {
                //Display error 1
                error = 1;
    }

    //Data validation for IC / Passport
    if (idtype.equals("IC / Passport :")) {
        if (idnum.length() != 12 || idnum.matches("[+-]?\\d*(\\.\\d+)?") == false) {
            //Display error 2
            error = 2;
        }
    }
    //If no eeror Display error 0
    return error;
}
```

Figure 2.2.7 Integer variable

Integer ‘error’ variable is created for error because of the value it holds is a whole number integer.

```
//Split string tempdata into an array under string eachdata
String[] eachdata = tempdata.split("\n");
```

Figure 2.2.8 String array variable

String array ‘eachdata’ is created to store values in arraylist.

```
//Get the number of days booked by minusing enddate and startdate
long difference = enddate.getTime() - startdate.getTime();
int numofday = (int)difference/86400000 + 1;
```

Figure 2.2.9 Long variable

Long variable is a numeric data type in Java that is primitive. It is used to extend size variables for number storage.

```
Object name;
name = roomidcombobox.getSelectedItem();
jtxtroomid.setText((String) name);
```

Figure 2.2.10 Object variable

Set variable ‘name’ as an object instead of string to uphold the value of an object in a container that hols reference. A universal holder.

2.3 Control Structure

```
if(jComboBox1.getSelectedItem().equals("Jungle View"))
{
    jtxtroomtype1.setText("Jungle View");
}
if(jComboBox1.getSelectedItem().equals("Sea View"))
{
    jtxtroomtype1.setText("Sea View");
}
```

Figure 2.3.1 If statement for Combo Box

If the selected item in the combo box is “Jungle View” then it will set text in the ‘jtxtroomtype1’ text box to “Jungle View”. On the other hand, if the selected item in the combo box is “Sea View” then it will set text in the ‘jtxtroomtype1’ text box to “Sea View”. End if.

```
public class Booking {
    //Validate booking function
    public static int validatebooking(String username, String icnumber, String contactnumber, String email, String idname) {
        int error = 0;
        //error 1 = input error
        //error 2 = IC number invalid
        //error 0 = no error

        //Data validation for name, contact number, email
        if (username.length() < 2 || username.length() > 50 || username.matches("[+-]?\\d*(\\.\\d+)?") == true
                || contactnumber.length() < 10 || contactnumber.length() > 11 || contactnumber.matches("[+-]?\\d*(\\.\\d+)?") == false
                || email.contains("@") == false || email.contains(".com") == false) {
                    //Display error 1
                    error = 1;
        }

        //Data validation for IC / Passport
        if (idname.equals("IC / Passport :")) {
            if (icnumber.length() != 12 || icnumber.matches("[+-]?\\d*(\\.\\d+)?") == false) {
                //Display error 2
                error = 2;
            }
        }
        //If no eeror Display error 0
        return error;
    }
```

Figure 2.3.2 If statement for Booking Validation

If ‘validatebooking’ function is used, string username, IC number, contact number, email and id name is collected. Initial error is 0. If name length is less than 2 and more than 50 and includes special characters, if length contact number inserted is not 10 and includes special characters, if email does not contain “@” and “.com” then it will lead to result true that will change integer error =1. If ‘idname’ equals to the following then check if the length of ic number is 12, if it isn’t then display error = 2. Once all statement is determined then it will return error integer value.

```
private void jbtnloginActionPerformed(java.awt.event.ActionEvent evt) {
    //Get input from textbox
    String password = jtxtpassword.getText();
    String username = jtxtusername.getText();

    //Check if textbox input contains both admin username and admin password
    if (password.contains("admin") && (username.contains("admin")))
    {
        jtxtusername.setText(null);
        jtxtpassword.setText(null);
        JOptionPane.showMessageDialog(this, "Successfully Login.");
        systemExit();

        //If both input is correct then open to room booking main page
        RoomBookingMain Info = new RoomBookingMain();
        Info.setVisible(true);
        dispose();
    }
    else
    {
        //Display Error if input is invalid
        JOptionPane.showMessageDialog(null,"Invalid Login Details", "Login Error",JOptionPane.ERROR_MESSAGE);
        jtxtpassword.setText(null);
        jtxtusername.setText(null);
    }
}
```

Figure 2.3.3 If-else statement for login validation

Check if the string contained in the textbox contains the login credentials. If it meets both the requirements then empty the textbox section and prompt user the Successfully login panel, exit system, direct user to the main room booking page. Else, prompt user login error panel with Invalid login details stated and set both textbox into null.

```
private void jbtnreceiptActionPerformed(java.awt.event.ActionEvent evt) {
    //Get row selection
    int row = bookingtable.getSelectedRow();
    if (row != -1) {
        //Get username in row Index 0 and convert to string. Open Receipt Form with the username as selectedid.
        String selectedid = bookingtable.getValueAt(row, 0).toString();
        ReceiptForm mbf = new ReceiptForm(selectedid);
        mbf.setLocationRelativeTo(null);
        mbf.setVisible(true);
        dispose();
    } else {
        JOptionPane.showMessageDialog(this, "No record is selected", "Message", JOptionPane.ERROR_MESSAGE);
    }
}
```

Figure 2.3.4 If-else statement for receipt function

Set the selected row by the user into an integer variable. If the row is not equal to -1 then do the following activities. Create string variable for the selected row username and open receipt form with the selected username string. Else, display no record is selected by the user.

```
private void jbtnviewActionPerformed(java.awt.event.ActionEvent evt) {
    //Validation if none of the user is selected and prompt user the error.
    if (jtxtname.getText().equals("")){
        JOptionPane.showMessageDialog(this, "Please Enter a Customer's Name, RoomID", "Message", JOptionPane.ERROR_MESSAGE);
    } else {
        //If user is selected then get user name or room ID text input with row filter. Index 0 and 6 from the booking details data.
        TableRowSorter rowSorter = new TableRowSorter((DefaultTableModel) bookingtable.getModel());
        rowSorter.setRowFilter(RowFilter.regexFilter(jtxtname.getText(), 0, 6));
        bookingtable.setRowSorter(rowSorter);
    }
}
```

Figure 2.3.5 If-else statement for search booking

If the textbox is empty, prompt user the error message to make user to enter username or room ID. Else, get username or room ID from the table data index 0 and 6, compare result and display the table row to the user.

```
//Calculate the price for different type of customer like citizen or non-citizen.
if (jtxtcitizenship.getText().equals("Citizen")) {
    int price = ((Integer) jspinday.getValue() * 350) * 110 / 100;
    jtxtprice.setText("RM " + String.valueOf(price));
} else {
    int price = (((Integer) jspinday.getValue() * 350) * 110 / 100) + ((Integer) jspinday.getValue() * 10);
    jtxtprice.setText("RM " + String.valueOf(price));
}
```

Figure 2.3.6 If-else statement for price calculation

If the textbox string is equal to “Citizen” then create integer variable and get value of the booking days times the amount per day and 10% tax chargers. Display the integer variable with ‘RM’ string. Else, proceed the same step but include an additional charge to the user and display the integer variable.

2.4 Selection Control Structure

```
public static int validatebooking(String name, String idnum, String connum, String email, String idtype) {
    int error = 0;
    //error 1 = input error
    //error 2 = IC number invalid
    //error 0 = no error

    //Data validation for name, contact number, email
    if (name.length() < 2 || name.length() > 50 || name.matches("[+-]?\\d*(\\.\\d+)?") == true
            || connum.length() < 10 || connum.length() > 11 || connum.matches("[+-]?\\d*(\\.\\d+)?") == false
            || email.contains("@") == false || email.contains(".com") == false) {
                //Display error 1
                error = 1;
    }

    //Data validation for IC / Passport
    if (idtype.equals("IC / Passport :")) {
        if (idnum.length() != 12 || idnum.matches("[+-]?\\d*(\\.\\d+)?") == false) {
            //Display error 2
            error = 2;
        }
    }
    //If no eeror Display error 0
    return error;
}
```

Figure 2.4.1 Switch case part 1.

This section is for validating booking details where if-else statement is used to determine the value of integer variable ‘error’. Different integer value determines different result. For example, 0 means no error, 1 means no input error, and 2 means IC number invalid. Refer to Figure 2.3.2 for more if-else explanation.

```
//Validation Details with input
int error = Booking.validatebooking(jtxtname.getText(), jtxtic.getText(), jtxtnumber.getText(), jtxtemail.getText(), jLabel3.getText());

//If there is error from validation
switch (error) {
    case 1:
        JOptionPane.showMessageDialog(this, "Invalid Customer details\nPlease check on customer's name, contact number, and email.", "Message", JOptionPane.ERROR_MESSAGE);
        break;
    case 2:
        JOptionPane.showMessageDialog(this, "Invalid Identification number.", "Message", JOptionPane.ERROR_MESSAGE);
        break;
    case 0:
        //If no error then proceed to this section.
        try
        {
            //Write input from textbox into the booking details file.
            File f = new File("bookingdetails.txt");
            FileWriter fw = new FileWriter(f,true);
            fw.write("\n"+name+"\n"+citizenship+"\n"+ic+"\n"+contact+"\n"+email+"\n"+roomtype+"\n"+roomid+"\n"+startdate+"\n"+enddate+"\n"+price);
            fw.close();

            //Once all data are insert then set all data into null (empty values).
            jtxtname.setText(null);
            jtxtic.setText(null);
            jtxtnumber.setText(null);
            jtxtemail.setText(null);
            jtxtcitizenship.setText(null);
            jtxtroomtype1.setText(null);
            jtxtroomid.setText(null);
            jtxtprice.setText(null);
            JOptionPane.showMessageDialog(null,"Successfully Added.");

            this.setVisible(false);
            SearchBookingListForm Info =new SearchBookingListForm();
            Info.setVisible(true);
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null,"System Error");
        }
}
```

Figure 2.4.2 Switch case part 2.

Receive integer variable from booking validation from Figure 2.4.1. Switch case is used to determine different result with different integer value from ‘error’ variable. In case scenario where ‘error’ variable = 1 then prompt user an error where there is invalid customer detail for name, contact number or email address. In case scenario where ‘error’ variable = 2 then prompt user an error message where there is an invalid identification number inserted to the textbox. In case scenario where ‘error’ variable = 0 which is no error, then proceed the process for adding the booking details into the data file.

2.5 Lopping Structure

```
//Delete room booking function
public static void deletebooking(String name) {
    //Create new arraylist and insert room booking data into the arraylist
    ArrayList<String> bookingdata = LoadData.bookingdata();

    //Create new string and insert all data except the deleted data by looking through the username string
    String newdata = ""; int count = 0;
    while (count < bookingdata.size()) {
        if (bookingdata.get(count).equals(name)) {
            count = count + 9;
        } else {
            //Every data line once complete skips to next line with \n
            newdata = newdata + bookingdata.get(count) + "\n";
        }
        count++;
    }

    //Rewrite the room booking data in the bookingdetails.txt
    File bookingfile1 = new File("bookingdetails.txt");
    try {
        FileWriter bookingfile2 = new FileWriter(bookingfile1, false);
        bookingfile2.write(newdata);
        bookingfile2.close();
    } catch (IOException ioe) {
        JOptionPane.showMessageDialog(null,"System Error");
    }
}
```

Figure 2.5.1 While Loop for Deleting Room Booking data.

Variable String ‘bookingdata’ array list is created to contain the data from load data class booking data function. Another empty variable string is created as ‘newdata’ and integer variable is created as ‘count’ with value 0. While the size for ‘bookingdata’ is more than the value of ‘count’ it will loop the process of receiving data from the ‘bookingdata’ array list will continue till the ‘count’ value is more than the ‘bookingdata’ size. Every process will keep adding the value of ‘count’ by one and once the ‘count’ value is more than the ‘bookingdata’ size then ‘newdata’ variable will get the arraylist from ‘bookingdata’.

```
//Modify room booking function
public static void modifybooking(String name, String citizen, String ic, String contact, String email, String roomtype, String roomid, String startdate, String enddate, String price) {

    //Create new string array to store data from room booking textbox input
    ArrayList<String> modifydata = new ArrayList<String>();
    modifydata.add(name);
    modifydata.add(citizen);
    modifydata.add(ic);
    modifydata.add(contact);
    modifydata.add(email);
    modifydata.add(roomtype);
    modifydata.add(roomid);
    modifydata.add(startdate);
    modifydata.add(enddate);
    modifydata.add(price);

    //Insert all room booking data into an arraylist
    ArrayList<String> bookingdata = LoadData.bookingdata();

    //Create new string
    String newdata = "";
    int count = 0;
    //Insert new data by looking through the username index 0.
    while (count < bookingdata.size()) {
        if (bookingdata.get(count).equals(modifydata.get(0))) {
            for (int i = 0; i != 10; i++) {
                //Every data line once complete skips to next line with \n
                newdata = newdata + modifydata.get(i) + "\n";
                if (i < 9) {
                    count++;
                }
            }
        } else {
            newdata = newdata + bookingdata.get(count) + "\n";
        }
        count++;
    }

    //Rewrite the room booking data in bookingdetails.txt
    File bookingfile1 = new File("bookingdetails.txt");
    try {
        FileWriter bookingfile2 = new FileWriter(bookingfile1, false);
        bookingfile2.write(newdata);
        bookingfile2.close();
    } catch (IOException ioe) {
        JOptionPane.showMessageDialog(null,"System Error");
    }
}
```

Figure 2.5.2 While and for loop with modifying room booking

While loop is used to insert new data by looking through the username index 0 received from the string data received. Integer ‘count’ is created with value 0. Size of the array list from ‘bookingdata’ is compared with ‘count’ value. While ‘count’ value is lesser than the ‘bookingdata’ size then it will proceed the process below to check the username index 0 from the ‘bookingdata’ array list and if it is true then a for loop will occur. The for loop is used to add data from the ‘modifying.get’ function from every index and store into the new ‘newdata’ string. For every process, the ‘i’ integer variable will increase by one till it reaches the maximum of ‘10’. Once the for loop is completed to store the data into the array then it will add one value to the ‘count’ integer variable. Once the ‘count’ integer variable reaches 10 then the while loop will stop.

```
//Load room booking data into table function
public static void LoadBookingTable(DefaultTableModel model) {
    //Load booking details data from bookingsdetails.txt into the table.
    //Count number of line/data in bookingdetails.txt
    int line = 0;
    try (BufferedReader reader = new BufferedReader(new FileReader("bookingdetails.txt"))) {
        while (reader.readLine() != null) {
            line++;
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null,"System Error");
    }

    //Every 10 data = 1 room booking
    int bookings = (line / 10);

    //Setting up the string (tempdata) to input all the data which is required for the table
    String tempdata = "";

    //Set number of table rows.
    model.setRowCount(bookings);

    //Insert data to tempdata from bookingdetails.txt
    File bookingfile = new File("bookingdetails.txt");
    try {
        Scanner myReader = new Scanner(bookingfile);
        for (int row = 0; row != bookings; row++) {
            for (int i = 0; i != 12; i++) {
                while (myReader.hasNextLine() && i != 0) {
                    String data = myReader.nextLine();
                    tempdata = tempdata + data + "\n";
                    break;
                }
            }
        }
    } catch (FileNotFoundException e) {
        System.out.println("File not found.");
    }

    //Split string tempdata into an array under string eachdata
    String[] eachdata = tempdata.split("\n");

    //Load the array into table
    int i = 0;
    //10 column and row = total number of room bookings
    for (int row = 0; row != bookings; row++) {
        for (int col = 0; col != 10; col++) {
            model.setValueAt(eachdata[i], row, col);
            i++;
        }
    }
```

Figure 2.5.3 For and While loop for displaying data on booking table.

For loop is used to repeat the function the get all the data onto the booking row by row with data from ‘bookingdetails.txt’. While loop is used to read and enter every data line by line into the temporary data variable and stop the process once the value is not met. For loop will stop processing once row = number of booking data.

Another for loop is used to enter all data stored in temporary data array list into the table row and column. This loop will stop once the number of rows is equal to the total number of booking data available.

```
//Load only booking start date function from all booking data
public static ArrayList<String> bookingstartdate() {
    //Set new arraylist from data in bookingdata() arraylist.
    ArrayList<String> bookingdata = bookingdata();

    //Separate every start date to another arraylist
    ArrayList<String> bookingstartdate = new ArrayList<String>();
    //Start date data is contained in every index count 7.
    int index = 7;
    //Store every index count 7 data into the arraylist.
    while (index < bookingdata.size()) {
        bookingstartdate.add(bookingdata.get(index));
        index = index + 10;
        if (bookingdata.size() <= index) {
        }
    }
    //return bookingstartdate arraylist.
    return bookingstartdate;
}
```

Figure 2.5.4 Load booking start date from the booking details file

While Loop is used to receive room booking starting date from the booking details file where it is located on index 7. So, while loop will receive all data for index 7 in all data from the booking details file and contain the date into the variable of ‘bookingstartdate’. The while loop will stop once the index value is more than the ‘bookingdata’ array size.

```
//Load all booking data function
public static ArrayList<String> bookingdata() {
    //Appending every data in bookingdetails.txt to an arraylist
    ArrayList<String> bookingdata = new ArrayList<String>();
    try {
        //Open new file for the bookingdetails.txt
        File bookingfile = new File("bookingdetails.txt");
        //Extract content from the bookingdetails.txt
        Scanner myReader = new Scanner(bookingfile);
        //Read every line and add into booking data arraylist
        while (myReader.hasNextLine()) {
            String data = myReader.nextLine();
            bookingdata.add(data);
        }
        myReader.close();
    //Call error if file is not found.
    } catch (FileNotFoundException e) {
        System.out.println("File Not found");
    }
    //return bookingdata arraylist.
    return bookingdata;
}
```

Figure 2.5.5 While loop for data gathering for booking data

While loop is used to read every line of code in ‘bookingdetails.txt’ and store the data into an string array list.

```
public ModifyBookingForm(String selectedid) {
    initComponents();
    setLocationRelativeTo(null);

    //Set date format to yyyy/mm/dd to maintain an correct order throughout entire process
    jtxtstartdate.setDateFormatString("yyyy/MM/dd");

    //Set date range for user
    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.YEAR, 1);
    //Set date variable to get time input
    Date mindate = cal.getTime();
    Date maxdate = cal.getTime();
    jtxtstartdate.setDate(mindate);
    jtxtstartdate.setSelectableDateRange(mindate, maxdate);

    //Load and store all the booking record data into an arraylist
    ArrayList<String> bookingdata = LoadData.bookingdata();

    //Create an arraylist to append the new selected booking data
    ArrayList<String> data = new ArrayList<String>();

    //Load out data of the booking which has the same room id with the String selectedid
    int index = 0, count = 0;
    while (index < bookingdata.size()) {
        if (bookingdata.get(index).equals(selectedid)) {
            count = index;
            break;
        }
        index++;
    }
    //One Room booking data contains 10 different data so repeat steps 10 times.
    for (int i = 0; i != 10; i++) {
        data.add(bookingdata.get(count));
        count++;
    }
```

Figure 2.5.6 While and for loop for modifying booking form

While loop is used to automate task to load out data from the bookingdetails.txt and list out the room id data where it has the same username as the customer data. For loop is to repeat entering data into the variable selected.

2.6 OOPS Concepts

2.6.1 Class

```
public class AddBookingForm extends javax.swing.JFrame {

    /**
     * Creates new form AddBooking
     */
    public AddBookingForm() {
        initComponents();
        setLocationRelativeTo(null);

        //Set date format to yyyy/mm/dd to maintain an correct order throughout entire process
        jtxtstartdate.setDateFormatString("yyyy/MM/dd");
        jtxtenddate.setDateFormatString("yyyy/MM/dd");

        //Set date range for users
        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.DATE, 3);
        cal.add(Calendar.YEAR, 1);
        //Create date variable to get time.
        Date mindate = cal.getTime();
        Date maxdate = cal.getTime();
        //Set minimum and maximum date range
        jtxtstartdate.setDate(mindate);
        jtxtstartdate.setSelectableDateRange(mindate, maxdate);
        jtxtenddate.setDate(mindate);
        jtxtenddate.setSelectableDateRange(mindate, maxdate);
    }
```

Figure 2.6.1.1 AddBookingForm class

Add booking form class is a public class that have extension to javax.swing.JFrame which opens a window panel for input/output operations like adding room booking operation.

```
public class Booking {
    //Validate booking function
    public static int validatebooking(String name, String idnum, String connum, String email, String idtype) {
        int error = 0;
        //error 1 = input error
        //error 2 = IC number invalid
        //error 0 = no error

        //Data validation for name, contact number, email
        if (name.length() < 2 || name.length() > 50 || name.matches("[+-]?\\d*(\\.\\d+)?") == true
                || connum.length() < 10 || connum.length() > 11 || connum.matches("[+-]?\\d*(\\.\\d+)?") == false
                || email.contains("@") == false || email.contains(".com") == false) {
                    //Display error 1
                    error = 1;
        }

        //Data validation for IC / Passport
        if (idtype.equals("IC / Passport :")) {
            if (idnum.length() != 12 || idnum.matches("[+-]?\\d*(\\.\\d+)?") == false) {
                //Display error 2
                error = 2;
            }
        }
        //If no eeror Display error 0
        return error;
    }
```

Figure 2.6.1.2 Booking class

Booking class is a public class that contains methods like validatebooking(), bookedroomid(), LoadBookingTable(), modifybooking(), and deletebooking().

```java
public class LoadData {
    //Load all booking data function
    public static ArrayList<String> bookingdata() {
        //Appending every data in bookingdetails.txt to an arraylist
        ArrayList<String> bookingdata = new ArrayList<String>();
        try {
            //Open new file for the bookingdetails.txt
            File bookingfile = new File("bookingdetails.txt");
            //Extract content from the bookingdetails.txt
            Scanner myReader = new Scanner(bookingfile);
            //Read every line and add into booking data arraylist
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                bookingdata.add(data);
            }
            myReader.close();
        //Call error if file is not found.
        } catch (FileNotFoundException e) {
            System.out.println("File Not found");
        }
        //return bookingdata arraylist.
        return bookingdata;
    }

    //Load only room ID function from all booking data
    public static ArrayList<String> roombookingid() {
        //Set new arraylist from data in bookingdata() arraylist.
        ArrayList<String> bookingdata = bookingdata();

        //Separate every booking room id to another arraylist.
        ArrayList<String> roombookingid = new ArrayList<String>();
        //Booking room id data is contained in every index count 6.
        int index = 6;
        //Store every index count 6 data into the arraylist.
        while (index < bookingdata.size()) {
            roombookingid.add(bookingdata.get(index));
            index = index + 10;
            if (bookingdata.size() <= index) {
            }
        }
        //return roombookingid arraylist.
        return roombookingid;
    }
```

Figure 2.6.1.3 LoadData class

Load date class is a public class that contains methods like loading all booking data, load room booking id, booking start date, booking end date.

Bookingdata(), roombookingid(), bookingstartdate(), bookingenddate().

```java
public class LoginForm extends javax.swing.JFrame {

    /**
     * Creates new form TestingForm
     */

    public LoginForm() {
        initComponents();
        setLocationRelativeTo(null);
    }
```

Figure 2.6.1.4 LoginForm class

Login Form class is a public class that have extension to javax.swing.JFrame which opens a window panel for input/output operations like login operation.

```
public class ModifyBookingForm extends javax.swing.JFrame {

    /**
     * Creates new form ModifyBooking
     * @param selectedid
     */
    public ModifyBookingForm(String selectedid) {
        initComponents();
        setLocationRelativeTo(null);

        //Set date format to yyyy/mm/dd to maintain an correct order throughout entire process
        jtxtstartdate.setDateFormatString("yyyy/MM/dd");

        //Set date range for user
        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.YEAR, 1);
        //Set date variable to get time input
        Date mindate = cal.getTime();
        Date maxdate = cal.getTime();
        jtxtstartdate.setDate(mindate);
        jtxtstartdate.setSelectableDateRange(mindate, maxdate);

        //Load and store all the booking record data into an arraylist
        ArrayList<String> bookingdata = LoadData.bookingdata();

        //Create an arraylist to append the new selected booking data
        ArrayList<String> data = new ArrayList<String>();
```

Figure 2.6.1.5 ModifyBookingForm class

Modify Booking Form class is a public class that have extension to javax.swing.JFrame which opens a window panel for input/output operations like modifying and deleting room booking operation. It is opened by the ‘selectedid’ string which is the username from previous forms.

```
public class ReceiptForm extends javax.swing.JFrame {

    /**
     * Creates new form ReceiptForm
     * @param selectedid
     */
    public ReceiptForm(String selectedid) {
        initComponents();
        setLocationRelativeTo(null);

        //Set date format to yyyy/mm/dd to maintain an correct order throughout entire process
        jtxtstartdate.setDateFormatString("yyyy/MM/dd");

        //Date range
        Calendar cal = Calendar.getInstance();
        Date mindate = cal.getTime();
        cal.add(Calendar.YEAR, 1);
        Date maxdate = cal.getTime();
        jtxtstartdate.setDate(mindate);
        jtxtstartdate.setSelectableDateRange(mindate, maxdate);

        //Load and store all the booking record data into an arraylist
        ArrayList<String> bookingdata = LoadData.bookingdata();

        //Create an arraylist to append the new selected booking data
        ArrayList<String> data = new ArrayList<String>();

        //Load out all the data of the booking which has the same id with the String selectedid
        int index = 0, count = 0;
        while (index < bookingdata.size()) {
            if (bookingdata.get(index).equals(selectedid)) {
                count = index;
                break;
            }
            index++;
```

Figure 2.6.1.6 ReceiptForm class

Receipt Form class is a public class that have extension to javax.swing.JFrame which opens a window panel for input/output operations like confirming receipt form operation. It is opened by the ‘selectedid’ string which is the username from previous forms.

```
public class ReceiptListForm extends javax.swing.JFrame {

    /**
     * Creates new form Receipt
     */
    public ReceiptListForm() {
        initComponents();
        setLocationRelativeTo(null);

        //Allow Row selection instead of individual selection on the table list.
        bookingtable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        bookingtable.setFocusable(false);

        //Load all the data to the table
        Booking.LoadBookingTable((DefaultTableModel) bookingtable.getModel());
    }
```

Figure 2.6.1.7 ReceiptListForm class

Receipt list form class is a public class that have extension to javax.swing.JFrame which opens a window panel for input/output operations like displaying room booking receipt list operation.

```
public class RoomBookingMain extends javax.swing.JFrame {

    /**
     * Creates new form RoomBooking
     */
    public RoomBookingMain() {
        initComponents();
        setLocationRelativeTo(null);
    }

    /**
     * This method is called from within the constructor to initialize the form
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

Figure 2.6.1.8 RoomBookingMain class

Room Booking Main class is a public class that have extension to javax.swing.JFrame which opens a window panel for input/output operations like displaying all functions for room booking operation.

```
public class SearchBookingListForm extends javax.swing.JFrame {

    /**
     * Creates new form ViewBooking
     */
    public SearchBookingListForm() {
        initComponents();
        setLocationRelativeTo(null);

        //Allow Row selection instead of individual selection on the table list.
        bookingtable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        bookingtable.setFocusable(false);

        //Load all the data to the table list
        Booking.LoadBookingTable((DefaultTableModel) bookingtable.getModel());
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

Figure 2.6.1.9 SearchBookingListForm class

Search booking list form class is a public class that have extension to javax.swing.JFrame which opens a window panel for input/output operations like displaying, searching, modifying, deleting, refreshing room booking operation.

2.6.2 Object

```
private void jbtnaddActionPerformed(java.awt.event.ActionEvent evt) {
    //Get input from textbox
    String name = jtxtname.getText();
    String citizenship = jtxtcitizenship.getText();
    String ic = jtxtic.getText();
    String contact = jtxtnumber.getText();
    String email = jtxtemail.getText();
    String roomtype = jtxtroomtypel.getText();
    String roomid = jtxtroomid.getText();
    String price = jtxtprice.getText();

    //Set date format
    SimpleDateFormat DateFormat = new SimpleDateFormat("yyyy/MM/dd");
    //Get start and end date entered by the users.
    String startdate = DateFormat.format(jtxtstartdate.getDate());
    String enddate = DateFormat.format(jtxtenddate.getDate());

    //Validation Details with input
    int error = Booking.validatebooking(jtxtname.getText(), jtxtic.getText(), jtxtnumber.getText(), jtxtemail.getText(), jLabel3.getText());
```

Figure 2.6.2.1 SimpleDateFormat Object

'SimpleDateFormat()' is a class where an object is created named 'DateFormat' to use the class where the attribute is equal to "yyyy/MM/dd".

```
//Load only booking start date function from all booking data
public static ArrayList<String> bookingstartdate() {
    //Set new arraylist from data in bookingdata() arraylist.
    ArrayList<String> bookingdata = bookingdata();

    //Separate every start date to another arraylist
    ArrayList<String> bookingstartdate = new ArrayList<String>();
    //Start date data is contained in every index count 7.
    int index = 7;
    //Store every index count 7 data into the arraylist.
    while (index < bookingdata.size()) {
        bookingstartdate.add(bookingdata.get(index));
        index = index + 10;
        if (bookingdata.size() <= index) {
        }
    }
    //return bookingstartdate arraylist.
    return bookingstartdate;
}
```

Figure 2.6.2.2 bookingdata Object

'bookingdata()' is a class where an object is created named 'bookingdata' with Array string list utilities to use the class functions.

```
private void jbtnloginActionPerformed(java.awt.event.ActionEvent evt) {
    //Get input from textbox
    String password = jtxtpassword.getText();
    String username = jtxtusername.getText();

    //Check if textbox input contains both admin username and admin password
    if (password.contains("admin") && (username.contains("admin")))
    {
        jtxtusername.setText(null);
        jtxtpassword.setText(null);
        JOptionPane.showMessageDialog(this, "Successfully Login.");
        systemExit();

        //If both input is correct then open to room booking main page
        RoomBookingMain Info = new RoomBookingMain();
        Info.setVisible(true);
        dispose();
    }
    else
    {
        //Display Error if input is invalid
        JOptionPane.showMessageDialog(null,"Invalid Login Details", "Login Error",JOptionPane.ERROR_MESSAGE);
        jtxtpassword.setText(null);
        jtxtusername.setText(null);
    }
}
```

Figure 2.6.2.3 Info Object for page redirection.

'RoomBookingMain()' is a class where an object is created named 'Info' to open up a new prompt to the user when 'Info.setVisible(true)' function is called.

2.6.3 Encapsulation

```
private void jbtnmodifyActionPerformed(java.awt.event.ActionEvent evt) {
    //Set date format to yyyy/mm/dd to maintain an correct order throughout entire process
    SimpleDateFormat DateFormat = new SimpleDateFormat("yyyy/MM/dd");
    try {
        //Get date and convert to String
        String startdate = DateFormat.format(jtxtstartdate.getDate());
    } catch (NullPointerException e) {
        //Date Validation
        JOptionPane.showMessageDialog(this, "Please Insert a Date.", "Message", JOptionPane.ERROR_MESSAGE);
    }

    //Validation for text input
    int error = Booking.validatebooking(jtxtname.getText(), jtxtic.getText(), jtxtnumber.getText(), jtxtemail.getText(), jLabel3.getText());

    //If there is error from the validation then output the following.
    switch (error) {
        case 1:
        JOptionPane.showMessageDialog(this, "Invalid Customer details\nPlease check on customer's name, contact number, and email.", "Message", JOptionPane.ERROR_MESSAGE);
        break;
        case 2:
        JOptionPane.showMessageDialog(this, "Invalid Identification number.", "Message", JOptionPane.ERROR_MESSAGE);
        break;
        //If there is no error then case 0 and proceed with modification function.
        case 0:
        //Get start date from input
        String startdate = DateFormat.format(jtxtstartdate.getDate());

        //Get end date from input
        String enddate = DateFormat.format(jtxtenddate.getDate());

        //Modify booking from all input
        Booking.modifybooking(jtxtname.getText(), jtxtcitizenship.getText(), jtxtic.getText(), jtxtnumber.getText(), jtxtemail.getText(), jtxtroomtypel.getText(), roomidcombobox.getSelecte

        //Once modification is successful then display Successfully Modified and prompt the Search Booking List Form to the user.
        JOptionPane.showMessageDialog(this, "Successfully Modified.");
        SearchBookingListForm brf = new SearchBookingListForm();
        brf.setLocationRelativeTo(null);
        brf.setVisible(true);
        dispose();
    }
}
```

Figure 2.6.3.1 Private class variable/attribute for modify button

The process under ‘jbtnmodifyActionPerformed’ is a sensitive data hidden from the user unless it is selected by the user. It is utilized to give better control of class attributes and methods to the user. Changing this section of the java code will not affect the entire program unless it is triggered to utilize the action event.

```
private void jbtnbackActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible(false);
    RoomBookingMain Info =new RoomBookingMain();
    Info.setVisible(true);
}
```

Figure 2.6.3.2 Private class variable/attribute for back button

The process under ‘jbtnbackActionPerformed’ is a sensitive data hidden from the user unless it is selected by the user. This java code section will not be performed if no action is done by the user. If it is performed, it will open the specified prompt page menu which is ‘RoomBookingMain()’, to the user.

2.6.4 Generalization

```
public class LoadData {
    //Load all booking data function
    public static ArrayList<String> bookingdata() {
        //Appending every data in bookingdetails.txt to an arraylist
        ArrayList<String> bookingdata = new ArrayList<String>();
        try {
            //Open new file for the bookingdetails.txt
            File bookingfile = new File("bookingdetails.txt");
            //Extract content from the bookingdetails.txt
            Scanner myReader = new Scanner(bookingfile);
            //Read every line and add into booking data arraylist
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                bookingdata.add(data);
            }
            myReader.close();
        //Call error if file is not found.
        } catch (FileNotFoundException e) {
            System.out.println("File Not found");
        }
        //return bookingdata arraylist.
        return bookingdata;
    }

    //Load only room ID function from all booking data
    public static ArrayList<String> roombookingid() {
        //Set new arraylist from data in bookingdata() arraylist.
        ArrayList<String> bookingdata = bookingdata();

        //Separate every booking room id to another arraylist.
        ArrayList<String> roombookingid = new ArrayList<String>();
        //Booking room id data is contained in every index count 6.
        int index = 6;
        //Store every index count 6 data into the arraylist.
        while (index < bookingdata.size()) {
            roombookingid.add(bookingdata.get(index));
            index = index + 10;
            if (bookingdata.size() <= index) {
            }
        }
        //return roombookingid arraylist.
        return roombookingid;
    }
```

Figure 2.6.4.1 Generalization for loading data into an array list

The arraylist string from ‘bookingdata()’ is generalized and used in another method in ‘roombookingid()’. From the data above, the variable in the array list is reformatted from one class method into another where the data will be inserted into another array list string.

```
public class SearchBookingListForm extends javax.swing.JFrame {

    /**
     * Creates new form ViewBooking
     */
    public SearchBookingListForm() {
        initComponents();
        setLocationRelativeTo(null);

        //Allow Row selection instead of individual selection on the table list.
        bookingtable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        bookingtable.setFocusable(false);

        //Load all the data to the table list
        Booking.LoadBookingTable((DefaultTableModel) bookingtable.getModel());
    }
```

Figure 2.6.4.2 Generalization for Search booking data in table list.

The class 'SearchBookingListForm' is extended with 'javax.swing.JFrame' where it will inherits the members and functions of it's superclass which is 'javax.swing.JFrame'.

2.6.5 Constructor

```
//Booked room id function
public static ArrayList<String> bookedroomid(String startdatel, String enddatel) {
    //List out arraylist from booking start and end date arraylist.
    ArrayList<String> bookingstartdate = LoadData.bookingstartdate();
    ArrayList<String> bookingenddate = LoadData.bookingenddate();

    //Set Date Format
    SimpleDateFormat DateFormat = new SimpleDateFormat("yyyy/MM/dd");

    //Set date format to string
    Date startdate = null, enddate = null;
    try {
        //Enter date input from the arraylist above.
        startdate = DateFormat.parse(startdatel);
        enddate = DateFormat.parse(enddatel);
    } catch (Exception e1) {
        JOptionPane.showMessageDialog(null,"System Error");
    }
```

Figure 2.6.5.1 Constructor used in bookedroomid() method

'bookedroomid' method has a constructor that initializes the booking start date and end date. The string array list inherits the data variables from 'LoadData.bookingstartdate()' and 'LoadData.bookingenddate()' which contains the information for both booking start and end date for every room booking made. From this data, the function for 'bookedroomid()' can determine the date data from every booked room id.

```
//Delete room booking function
public static void deletebooking(String name) {
    //Create new arraylist and insert room booking data into the arraylist
    ArrayList<String> bookingdata = LoadData.bookingdata();

    //Create new string and insert all data except the deleted data by looking through the username string
    String newdata = ""; int count = 0;
    while (count < bookingdata.size()) {
        if (bookingdata.get(count).equals(name)) {
            count = count + 9;
        } else {
            //Every data line once complete skips to next line with \n
            newdata = newdata + bookingdata.get(count) + "\n";
        }
        count++;
    }

    //Rewrite the room booking data in the bookingdetails.txt
    File bookingfile1 = new File("bookingdetails.txt");
    try {
        FileWriter bookingfile2 = new FileWriter(bookingfile1, false);
        bookingfile2.write(newdata);
        bookingfile2.close();
    } catch (IOException ioe) {
        JOptionPane.showMessageDialog(null,"File System Error");
    }
}
```

Figure 2.6.5.2 Constructor used in deletebooking() method

'deletebooking' method has a constructor that initializes the username. The username is a string data called 'name' that will be used for the process under the 'deletebooking()' method.

2.6.6 Get-Set Method

```
public class LoadData {
    //Load all booking data function
    public static ArrayList<String> bookingdata() {
        //Appending every data in bookingdetails.txt to an arraylist
        ArrayList<String> bookingdata = new ArrayList<String>();
        try {
            //Open new file for the bookingdetails.txt
            File bookingfile = new File("bookingdetails.txt");
            //Extract content from the bookingdetails.txt
            Scanner myReader = new Scanner(bookingfile);
            //Read every line and add into booking data arraylist
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                bookingdata.add(data);
            }
            myReader.close();
        //Call error if file is not found.
        } catch (FileNotFoundException e) {
            System.out.println("File Not found");
        }
        //return bookingdata arraylist.
        return bookingdata;
    }

    //Load only room ID function from all booking data
    public static ArrayList<String> roombookingid() {
        //Set new arraylist from data in bookingdata() arraylist.
        ArrayList<String> bookingdata = bookingdata();

        //Separate every booking room id to another arraylist.
        ArrayList<String> roombookingid = new ArrayList<String>();
        //Booking room id data is contained in every index count 6.
        int index = 6;
        //Store every index count 6 data into the arraylist.
        while (index < bookingdata.size()) {
            roombookingid.add(bookingdata.get(index));
            index = index + 10;
            if (bookingdata.size() <= index) {
            }
        }
        //return roombookingid arraylist.
        return roombookingid;
    }
```

Figure 2.6.6.1 Get-Set method for LoadData class

'bookingdata()' and 'roombookingid' utilizes the get-set method under the 'LoadDate' class where the method will get the input into a string array list and use the string array list to extract the requirement information that may be in different data index then once all data are extracted. It will set the data extracted into a new string array list and return the result.

```
private void jbtnaddActionPerformed(java.awt.event.ActionEvent evt) {
    //Get input from textbox
    String name = jtxtname.getText();
    String citizenship = jtxtcitizenship.getText();
    String ic = jtxtic.getText();
    String contact = jtxtnumber.getText();
    String email = jtxtemail.getText();
    String roomtype = jtxtroomtype1.getText();
    String roomid = jtxtroomid.getText();
    String price = jtxtprice.getText();

    //Set date format
    SimpleDateFormat DateFormat = new SimpleDateFormat("yyyy/MM/dd");
    //Get start and end date entered by the users.
    String startdate = DateFormat.format(jtxtstartdate.getDate());
    String enddate = DateFormat.format(jtxtenddate.getDate());

    //Validation Details with input
    int error = Booking.validatebooking(jtxtname.getText(), jtxtic.getText(), jtxtnumber.getText(), jtxtemail.getText(), jLabel3.getText());

    //If there is error from validation
    switch (error) {
        case 1:
            JOptionPane.showMessageDialog(this, "Invalid Customer details\nPlease check on customer's name, contact number, and email.", "Message", JOptionPane.ERROR_MESSAGE);
            break;
        case 2:
            JOptionPane.showMessageDialog(this, "Invalid Identification number.", "Message", JOptionPane.ERROR_MESSAGE);
            break;
        case 0:
            //If no error then proceed to this section.
            try
            {
                //Write input from textbox into the booking details file.
                File f = new File("bookingdetails.txt");
                FileWriter fw = new FileWriter(f,true);
                fw.write("\n"+name+"\n"+citizenship+"\n"+ic+"\n"+contact+"\n"+email+"\n"+roomtype+"\n"+roomid+"\n"+startdate+"\n"+enddate+"\n"+price);
                fw.close();

                //Once all data are insert then set all data into null (empty values).
                jtxtname.setText(null);
                jtxtic.setText(null);
                jtxtnumber.setText(null);
                jtxtemail.setText(null);
                jtxtcitizenship.setText(null);
                jtxtroomtype1.setText(null);
                jtxtroomid.setText(null);
                jtxtprice.setText(null);
                JOptionPane.showMessageDialog(null,"Successfully Added.");
```

Figure 2.6.6.2 Get-Set Method for adding room booking data.

This is another example for Get-set method in java where it is used to get text and set text in the textbox configure in the java swing panel. ‘getText()’ method is used to extract the information entered into the text box like ‘jtxtname’ and the data is converted into a string data. ‘setText()’ is another method to manually configure the input of the textbox where it can insert any value specified by the system.

2.6.7 Normal Method

```
public class Booking {
    //Validate booking function
    public static int validatebooking(String name, String idnum, String connum, String email, String idtype) {
        int error = 0;
        //error 1 = input error
        //error 2 = IC number invalid
        //error 0 = no error

        //Data validation for name, contact number, email
        if (name.length() < 2 || name.length() > 50 || name.matches("[+-]?\\d*(\\.\\d+)?") == true
                || connum.length() < 10 || connum.length() > 11 || connum.matches("[+-]?\\d*(\\.\\d+)?") == false
                || email.contains("@") == false || email.contains(".com") == false) {
                    //Display error 1
                    error = 1;
        }

        //Data validation for IC / Passport
        if (idtype.equals("IC / Passport :")) {
            if (idnum.length() != 12 || idnum.matches("[+-]?\\d*(\\.\\d+)?") == false) {
                //Display error 2
                error = 2;
            }
        }
        //If no eeror Display error 0
        return error;
    }
```

Figure 2.6.7.1 Normal Method (Validate booking)

This method is under Booking class where it will not run unless it is called. Booking.validatebooking() will be called with string name, contact number, IC/passport number, email in add booking form and modifying booking form where there will be text input textbox from users.

```
//Delete room booking function
public static void deletebooking(String name) {
    //Create new arraylist and insert room booking data into the arraylist
    ArrayList<String> bookingdata = LoadData.bookingdata();

    //Create new string and insert all data except the deleted data by looking through the username string
    String newdata = ""; int count = 0;
    while (count < bookingdata.size()) {
        if (bookingdata.get(count).equals(name)) {
            count = count + 9;
        } else {
            //Every data line once complete skips to next line with \n
            newdata = newdata + bookingdata.get(count) + "\n";
        }
        count++;
    }

    //Rewrite the room booking data in the bookingdetails.txt
    File bookingfile1 = new File("bookingdetails.txt");
    try {
        FileWriter bookingfile2 = new FileWriter(bookingfile1, false);
        bookingfile2.write(newdata);
        bookingfile2.close();
    } catch (IOException ioe) {
        JOptionPane.showMessageDialog(null,"File System Error");
    }
}
```

Figure 2.6.7.2 Normal Method (Delete booking)

This method is under Booking class where it will not run unless it is called. Booking.deletebooking() will be called with string name in modifying booking form and search booking form where there will be a data will be selected for deletion.

2.6.8 Exceptional Handling

```
//Booked room id function
public static ArrayList<String> bookedroomid(String startdate1, String enddate1) {
    //List out arraylist from booking start and end date arraylist.
    ArrayList<String> bookingstartdate = LoadData.bookingstartdate();
    ArrayList<String> bookingenddate = LoadData.bookingenddate();

    //Set Date Format
    SimpleDateFormat DateFormat = new SimpleDateFormat("yyyy/MM/dd");

    //Set date format to string
    Date startdate = null, enddate = null;
    try {
        //Enter date input from the arraylist above.
        startdate = DateFormat.parse(startdate1);
        enddate = DateFormat.parse(enddate1);
    } catch (Exception e1) {
        JOptionPane.showMessageDialog(null,"System Error");
    }
```

Figure 2.6.8.1 Exception error

This exception is an event that will occur with try and catch method. If there is an error in the progress for try section where it there is an disruption for the normal flow of the program instructions then it will prompt user an error with 'JOptionPane' which is an extra window panel that shows details of the error like "System Erorr".

```
//Insert data to tempdata from bookingdetails.txt
File bookingfile = new File("bookingdetails.txt");
try {
    Scanner myReader = new Scanner(bookingfile);
    for (int row = 0; row != bookings; row++) {
        for (int i = 0; i != 12; i++) {
            while (myReader.hasNextLine() && i != 0) {
                String data = myReader.nextLine();
                tempdata = tempdata + data + "\n";
                break;
            }
        }
    }
} catch (FileNotFoundException e) {
    System.out.println("File not found.");
}
```

Figure 2.6.8.2 FileNotFoundException error

This exception is used when there is attempted failure when opening the file with specified path name. This exception will be prompt to the user by File Input Stream and File Output Stream and Random-Access File constructors where the specified file path does not exist.

```
//Rewrite the room booking data in bookingdetails.txt
File bookingfile1 = new File("bookingdetails.txt");
try {
    FileWriter bookingfile2 = new FileWriter(bookingfile1, false);
    bookingfile2.write(newdata);
    bookingfile2.close();
} catch (IOException ioe) {
    JOptionPane.showMessageDialog(null,"File System Error");
}
```

Figure 2.6.8.3 IOException error

This exception is an event that will occur when there is input and output operation failure. This exception can be handles by try-catch-finally-block to allows developers to understand the problem with the java code and provides the correct error where actions that can be made to solve the problem which for example in Figure 2.6.8.1 it will state that the error is caused by I/O operations where the data stream is corrupted, or some error occur from writing/reading new data into the ‘bookingdetails.txt’ file.

2.6.9 File Concept

```
private void jbtnaddActionPerformed(java.awt.event.ActionEvent evt) {
    //Get input from textbox
    String name = jtxtname.getText();
    String citizenship = jtxtcitizenship.getText();
    String ic = jtxtic.getText();
    String contact = jtxtnumber.getText();
    String email = jtxtemail.getText();
    String roomtype = jtxtroomtype1.getText();
    String roomid = jtxtroomid.getText();
    String price = jtxtprice.getText();

    //Set date format
    SimpleDateFormat DateFormat = new SimpleDateFormat("yyyy/MM/dd");
    //Get start and end date entered by the users.
    String startdate = DateFormat.format(jtxtstartdate.getDate());
    String enddate = DateFormat.format(jtxtenddate.getDate());

    //Validation Details with input
    int error = Booking.validatebooking(jtxtname.getText(), jtxtic.getText(), jtxtnumber.getText(), jtxtemail.getText(), jLabel3.getText());

    //If there is error from validation
    switch (error) {
        case 1:
            JOptionPane.showMessageDialog(this, "Invalid Customer details\nPlease check on customer's name, contact number, and email.", "Message", JOptionPane.ERROR_MESSAGE);
            break;
        case 2:
            JOptionPane.showMessageDialog(this, "Invalid Identification number.", "Message", JOptionPane.ERROR_MESSAGE);
            break;
        case 0:
            //If no error then proceed to this section.
            try
            {
                //Write input from textbox into the booking details file.
                File f = new File("bookingdetails.txt");
                FileWriter fw = new FileWriter(f,true);
                fw.write("\n"+name+"\n"+citizenship+"\n"+ic+"\n"+contact+"\n"+email+"\n"+roomtype+"\n"+roomid+"\n"+startdate+"\n"+enddate+"\n"+price);
                fw.close();

                //Once all data are insert then set all data into null (empty values).
                jtxtname.setText(null);
                jtxtic.setText(null);
                jtxtnumber.setText(null);
                jtxtemail.setText(null);
                jtxtcitizenship.setText(null);
                jtxtroomtype1.setText(null);
                jtxtroomid.setText(null);
                jtxtprice.setText(null);
                JOptionPane.showMessageDialog(null,"Successfully Added.");
```

Figure 2.6.9.1 Adding data into file.

Firstly, string variables are created to get data from text files with ‘getText()’ method. Once the data are inserted into the string variables, it will undergo data validation and prompt error to user if there is any. Once there is no error, it will create new file variable to open the ‘bookingdetails.txt’ file where room booking data are stored. Another variable is created to add data using File writer class. ‘write()’ method is used to store the string variables into the specified file with specified order. Once all data is inserted, it will then close the specified file with file writer class. All textboxes will then be set as null, and a prompt will be displayed to user that the data is successfully added.

```
//Adding item in the RoomID combo box
//Check if textbox  input = Jungle View
if(jtxtroomtype1.getText().equals("Jungle View"))
{
try
{
    //Read all data from Jungle Room file like all room ID.
    File jungleroomfile = new File("jungleroom.txt");
    Scanner myReader = new Scanner(jungleroomfile);
    while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        roomidcombobox.addItem(data);
    }
}
catch (FileNotFoundException e)
{
System.out.println("File not found.");
}
```

Figure 2.6.9.2 Read data from a file.

Scanner class is used to read the 'jungleroom.txt' file. While loop is used to repeat reading the data from every line in the specified file and add the read data into the combo box specified. Once there is no more next line then the while loop will stop. If there is an error, then it will catch the file not found exception to the user.

```
public class LoadData {
    //Load all booking data function
    public static ArrayList<String> bookingdata() {
        //Appending every data in bookingdetails.txt to an arraylist
        ArrayList<String> bookingdata = new ArrayList<String>();
        try {
            //Open new file for the bookingdetails.txt
            File bookingfile = new File("bookingdetails.txt");
            //Extract content from the bookingdetails.txt
            Scanner myReader = new Scanner(bookingfile);
            //Read every line and add into booking data arraylist
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                bookingdata.add(data);
            }
            myReader.close();
        //Call error if file is not found.
        } catch (FileNotFoundException e) {
            System.out.println("File Not found");
        }
        //return bookingdata arraylist.
        return bookingdata;
    }
```

Figure 2.6.9.3 Load data from a file into an array list.

'LoadData' class has the method of 'bookingdata' where it will load all data from the 'bookingdetails.txt' file. First it will open a new variable to open the specified file. Extract the content with Scanner class. Use while loop to read every line and add into a string array list. Once there is no more next line then the loop is stop and the file will be close with 'close()' method from Scanner class and return the array list. If there an error with the file, then it will prompt user error.

```
//Load only booking start date function from all booking data
public static ArrayList<String> bookingstartdate() {
    //Set new arraylist from data in bookingdata() arraylist.
    ArrayList<String> bookingdata = bookingdata();

    //Separate every start date to another arraylist
    ArrayList<String> bookingstartdate = new ArrayList<String>();
    //Start date data is contained in every index count 7.
    int index = 7;
    //Store every index count 7 data into the arraylist.
    while (index < bookingdata.size()) {
        bookingstartdate.add(bookingdata.get(index));
        index = index + 10;
        if (bookingdata.size() <= index) {
        }
    }
    //return bookingstartdate arraylist.
    return bookingstartdate;
}
```

Figure 2.6.9.4 Load specific data from an array list into a new array list.

'bookingstartdate' method is used to load only the booking start date from the room booking detail string array list. The array list is collected from 'bookingdata()'. Where a new string array list will be made to collect the specific data from the 'bookingdata()' array list. The booking start date is located as index 7 on the array list data so a while loop is used to store every index count 7 data into the new string array list. The loop will stop once all data is collected and it will return all room booking start date value in the new array list.

```
//Delete room booking function
public static void deletebooking(String name) {
    //Create new arraylist and insert room booking data into the arraylist
    ArrayList<String> bookingdata = LoadData.bookingdata();

    //Create new string and insert all data except the deleted data by looking through the username string
    String newdata = ""; int count = 0;
    while (count < bookingdata.size()) {
        if (bookingdata.get(count).equals(name)) {
            count = count + 9;
        } else {
            //Every data line once complete skips to next line with \n
            newdata = newdata + bookingdata.get(count) + "\n";
        }
        count++;
    }

    //Rewrite the room booking data in the bookingdetails.txt
    File bookingfile1 = new File("bookingdetails.txt");
    try {
        FileWriter bookingfile2 = new FileWriter(bookingfile1, false);
        bookingfile2.write(newdata);
        bookingfile2.close();
    } catch (IOException ioe) {
        JOptionPane.showMessageDialog(null,"File System Error");
    }
}
```

Figure 2.6.9.5 Delete and rewrite room booking data from a file.

The method will create a new string array list and insert all data that equals to the username index 0 in the 'bookingdata()' string array list and rewrite the data with FileWriter class into the specified file path. It will prompt the user if there is an input output exception error.

```
//Modify room booking function
public static void modifybooking(String name, String citizen, String ic, String contact, String email, String roomtype, String roomid, String startdate, String enddate, String price) {

    //Create new string array to store data from room booking textbox input
    ArrayList<String> modifydata = new ArrayList<String>();
    modifydata.add(name);
    modifydata.add(citizen);
    modifydata.add(ic);
    modifydata.add(contact);
    modifydata.add(email);
    modifydata.add(roomtype);
    modifydata.add(roomid);
    modifydata.add(startdate);
    modifydata.add(enddate);
    modifydata.add(price);

    //Insert all room booking data into an arraylist
    ArrayList<String> bookingdata = LoadData.bookingdata();

    //Create new string
    String newdata = "";
    int count = 0;
    //Insert new data by looking through the username index 0.
    while (count < bookingdata.size()) {
        if (bookingdata.get(count).equals(modifydata.get(0))) {
            for (int i = 0; i != 10; i++) {
                //Every data line once complete skips to next line with \n
                newdata = newdata + modifydata.get(i) + "\n";
                if (i < 9) {
                    count++;
                }
            }
        } else {
            newdata = newdata + bookingdata.get(count) + "\n";
        }
        count++;
    }

    //Rewrite the room booking data in bookingdetails.txt
    File bookingfile1 = new File("bookingdetails.txt");
    try {
        FileWriter bookingfile2 = new FileWriter(bookingfile1, false);
        bookingfile2.write(newdata);
        bookingfile2.close();
    } catch (IOException ioe) {
        JOptionPane.showMessageDialog(null,"File System Error");
    }
}
```

Figure 2.6.9.6 Modify booking.

Firstly, create new string variable to collects all attributes collected from the method. Insert all room booking data into a new string array list. Insert new data by looking through the username index 0. While loop is used to loop the process to get every data line by line for 10 times since all 10 data equals to 1 room booking data. Once all data is collected, add the data from the new array list to a string. Rewrite the room booking data with FileWriter class. Close the file once completed and prompt error if there is an input output operation error.

3.0 Sample Output Screens

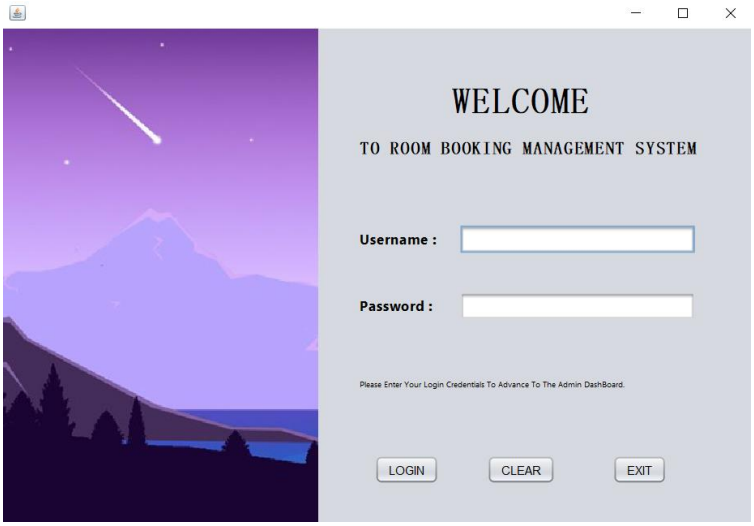

Figure 3.1.1 Admin Login Page

Login button will check login credentials, Clear button will clear the textbox, Exit button will exit java application.

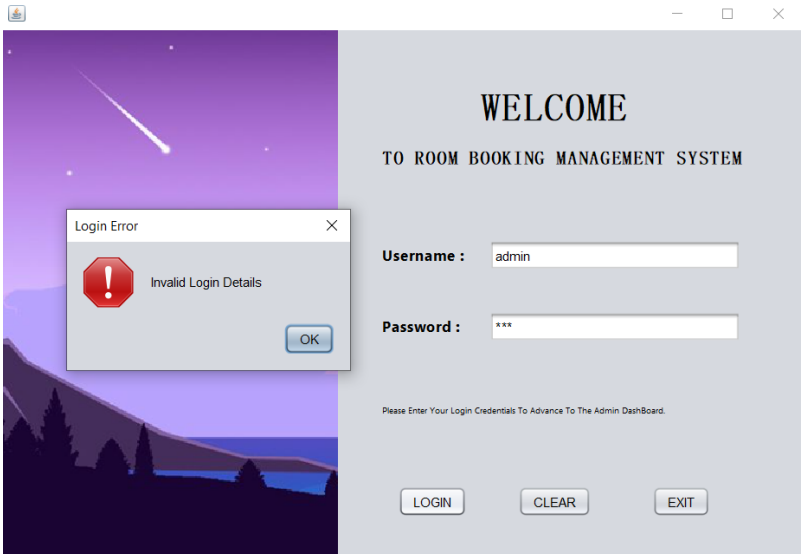

Figure 3.1.2 Admin Login Page (Invalid Login Details)

Login credentials (username-admin, password-admin). Prompt displayed when select login button.

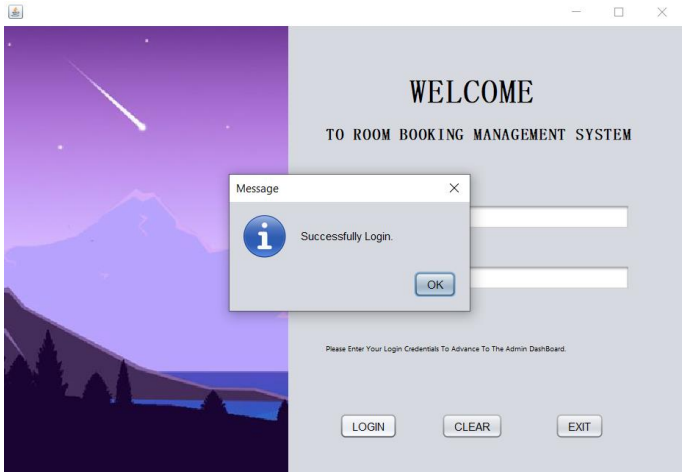

Figure 3.1.3 Admin Login Page (Successfully Login)

Select OK to go to admin dashboard.

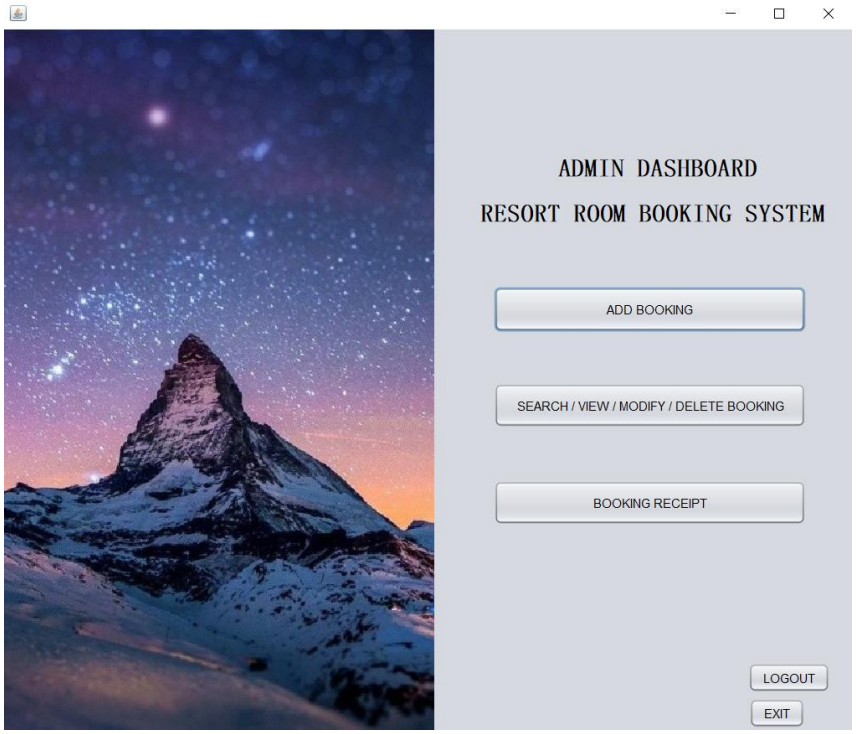

Figure 3.2 Admin Dashboard

Add booking button will direct user to add room booking panel, Search / view/ modify / delete booking button will direct user to room booking table panel, booking receipt button will direct user to room booking table list panel.

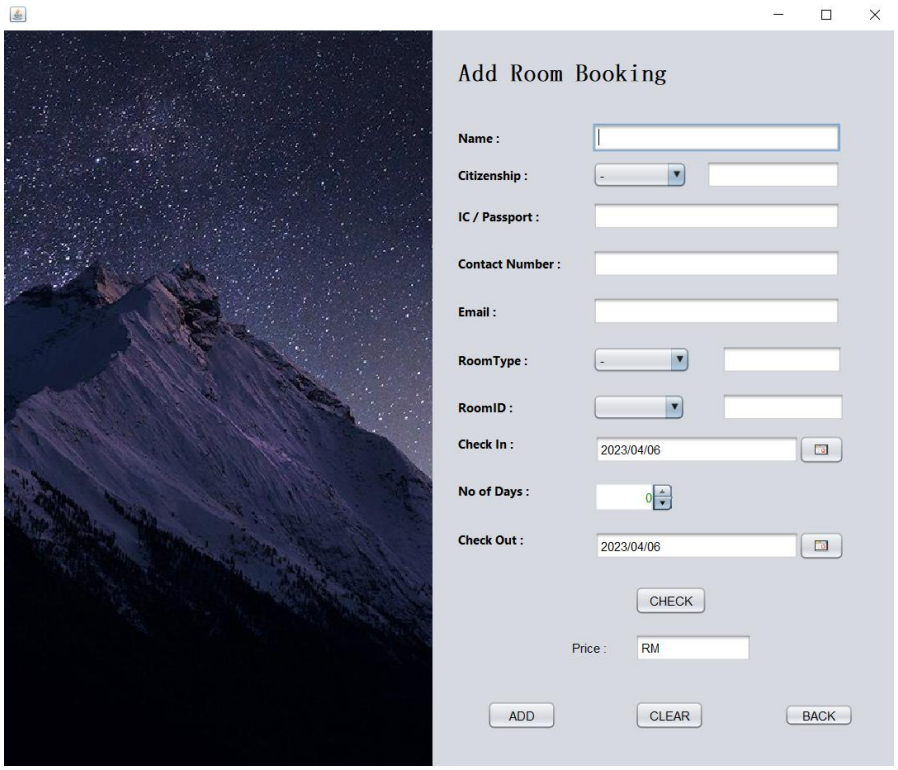

Figure 3.3.1 Add Room booking dashboard

Textbox to insert the characters, combo box to choose the available selection. Insert all details. Select check to get available room ID from the date entered and display the total price. Select Add button to add room booking information. Select Clear to clear all textbox. Select Back to return to admin dashboard.

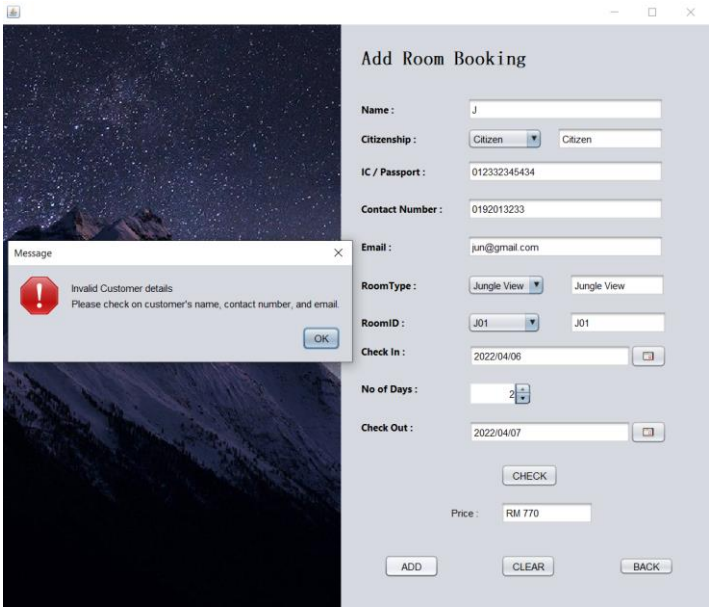

Figure 3.3.2 Add Room booking dashboard (Invalid customer details)

Error prompt to user when there is an error with the textbox input for customer's name, contact number, email.

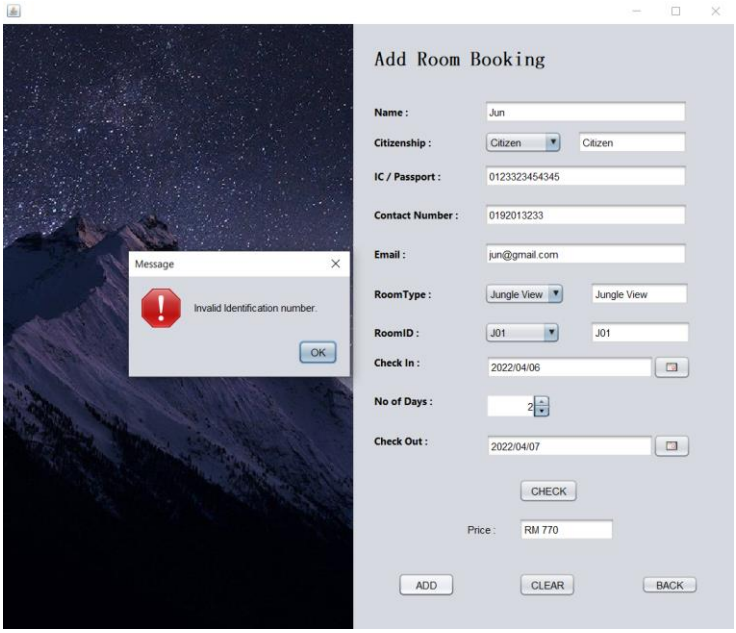

Figure 3.3.3 Add Room booking dashboard (Invalid customer details)

Error prompt to user when there is an error with the textbox input for IC / Passport number.

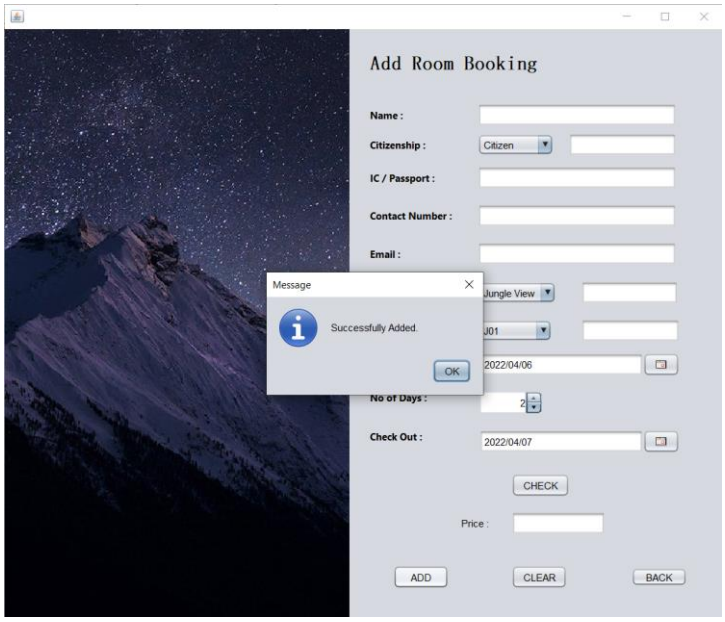

Figure 3.3.4 Add Room booking dashboard (Successfully added)

Prompt for user when all data validated correctly and inserted into the database.

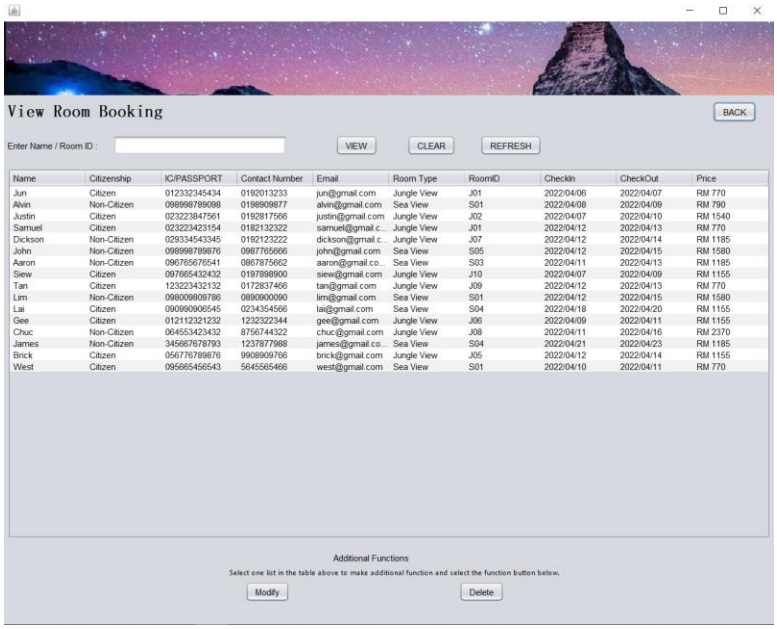

Figure 3.4.1 View Booking Dashboard

Room booking data displayed in table. Functions available like modify, delete, search, clear, refresh, and back.

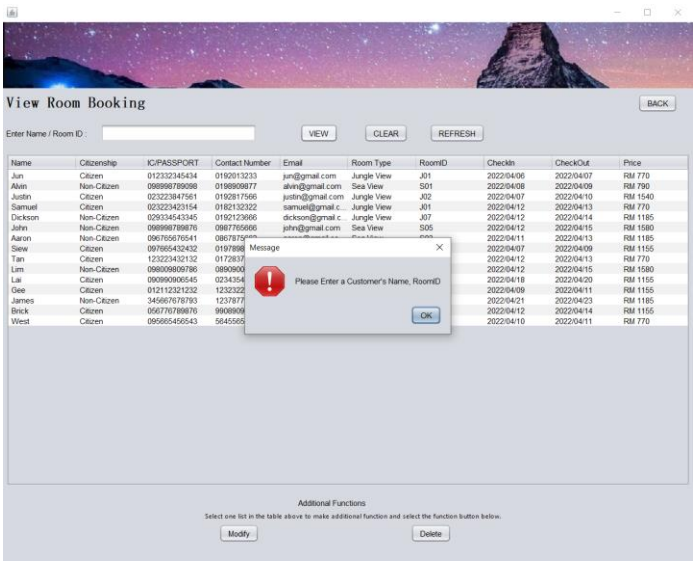

Figure 3.4.2 View Booking Dashboard (Search error)

Error prompt when View button is selected but empty input in textbox.

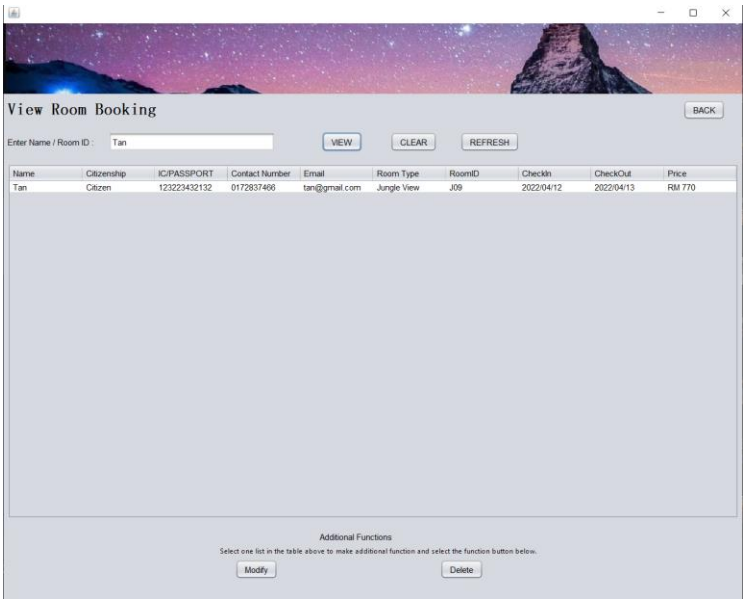

Figure 3.4.3 View Booking Dashboard (Search Function)

Room booking data displayed in table. Functions available like modify, delete, search, clear, refresh, and back.

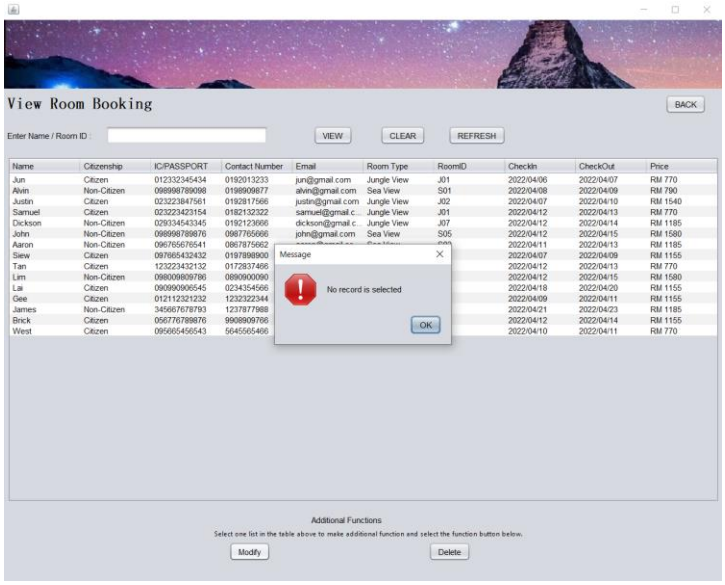

Figure 3.4.4 View Booking Dashboard (No record selected for modify and delete function)

Error prompt when there is no selected row for modify and delete button.

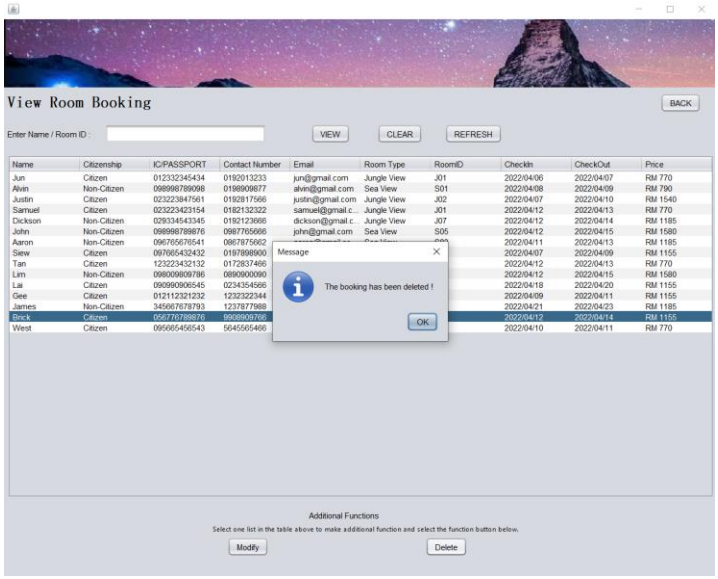

Figure 3.4.5 View Booking Dashboard (Delete function)

A prompt will successfully display deletion for the row selected.

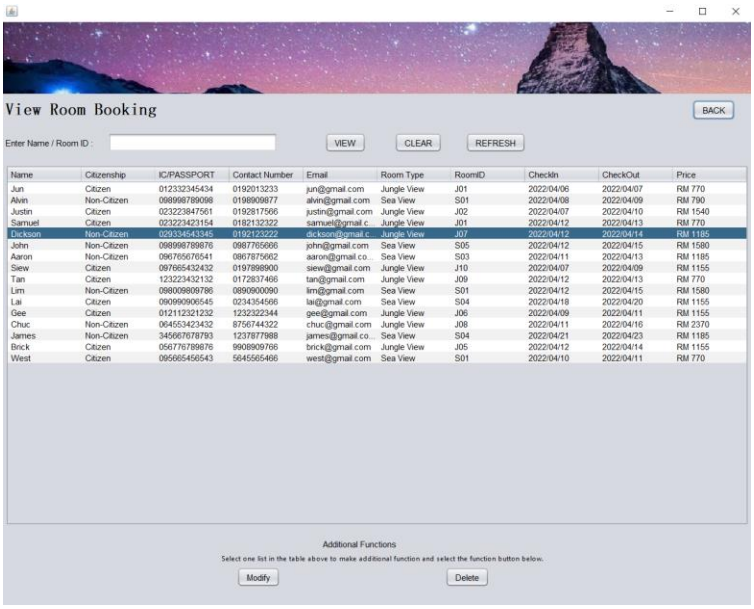

Figure 3.4.6 View Booking Dashboard (Select function)

Selected room booking detail row in the table.

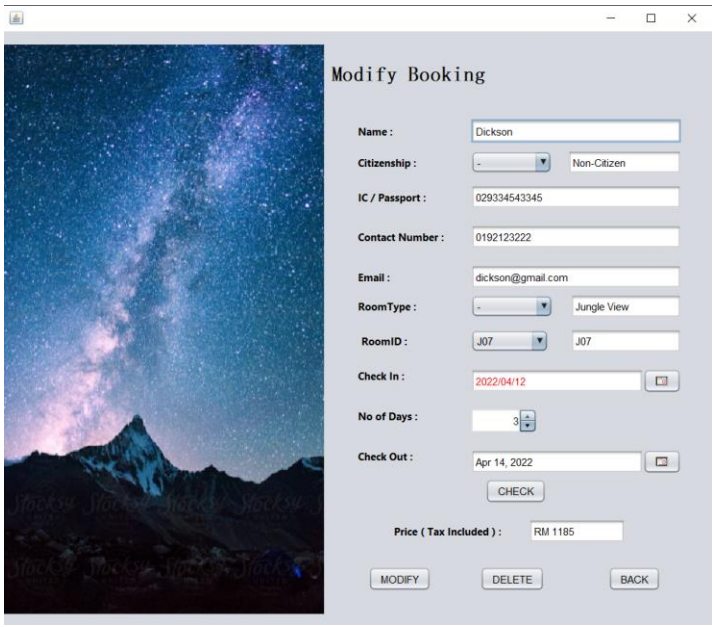

Figure 3.4.7.1 View Booking Dashboard (Modify function)

A prompt will display once user selected modify button with the selected data row in Figure 3.4.6. All information will be displayed.

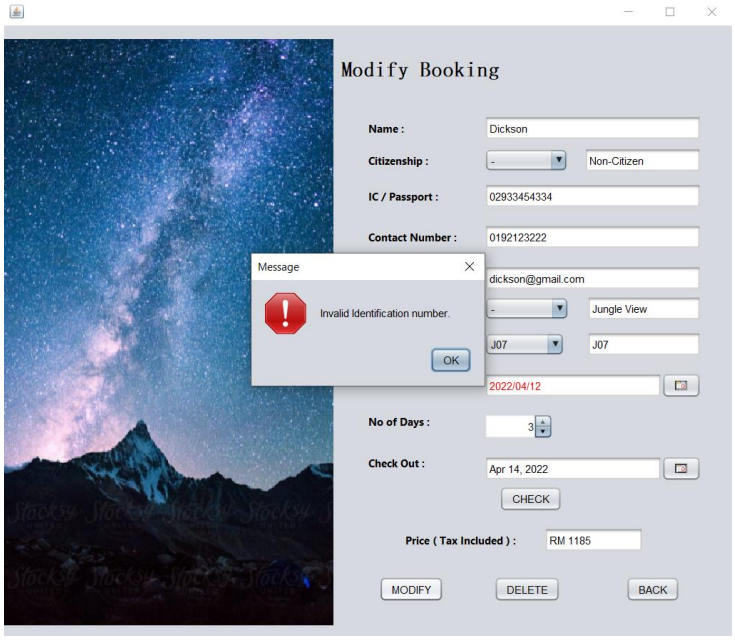

Figure 3.4.7.2 View Booking Dashboard (Modify function – Data Validation Error)

A prompt will display when user select Modify button, but the data inserted is incorrect.

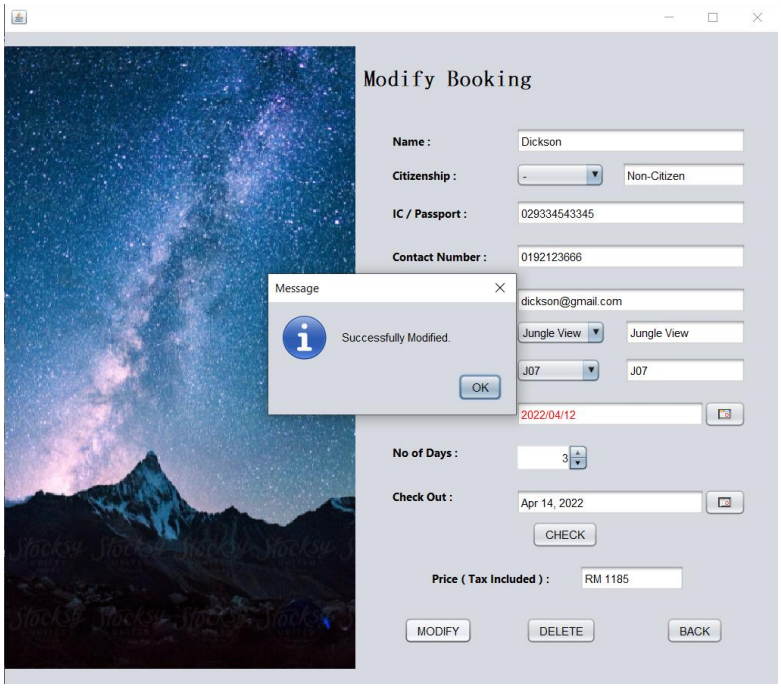

Figure 3.4.7.3 View Booking Dashboard (Modify function – Successfully Modified)

A prompt will successfully display modification for the customer room booking data.

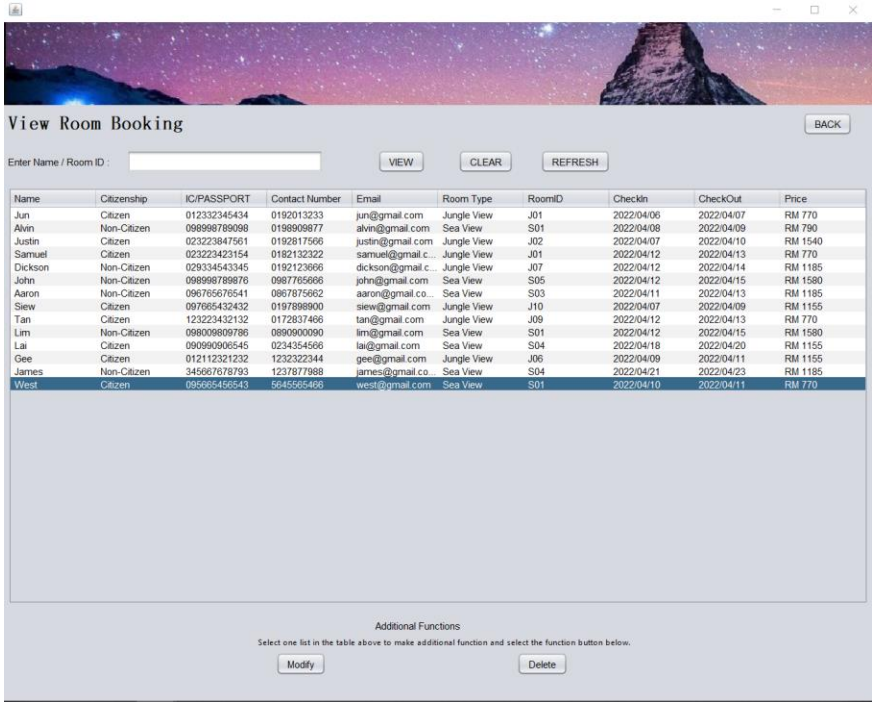

Figure 3.4.8 View Booking Dashboard (Updated Table)

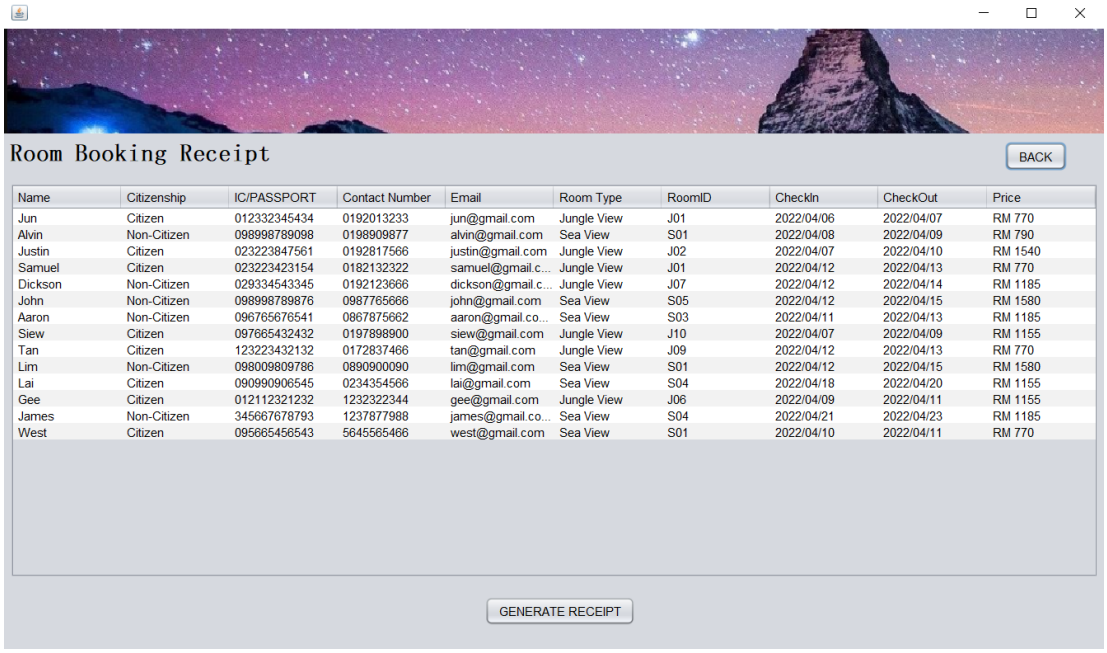

Figure 3.5.1 View All Receipt Dashboard.

Select Generate receipt button to direct to receipt page.

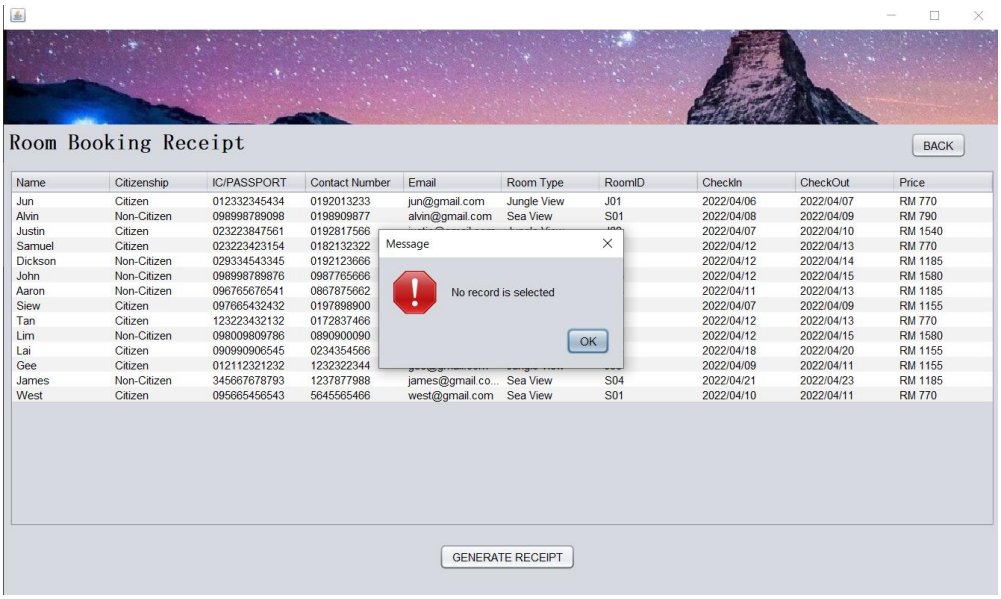

Figure 3.5.2 View All Receipt Dashboard (Error).

No data row selected to generate receipt for the room booking.

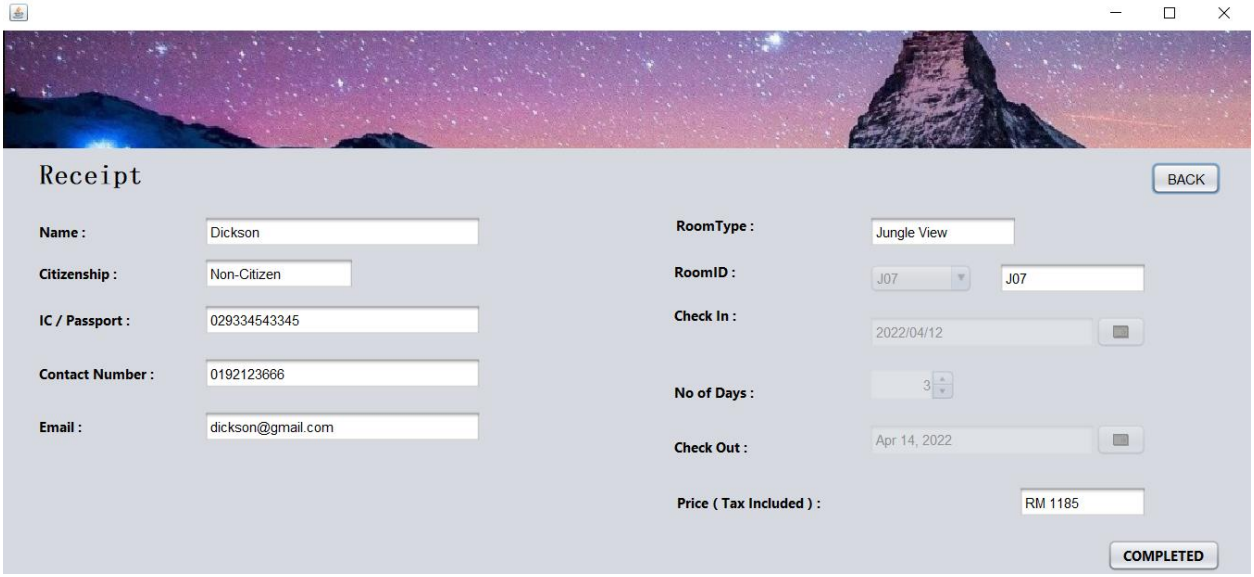

Figure 3.5.3 View All Receipt Dashboard (Receipt Page).

All data will be displayed accordingly. Select COMPLETED to direct to admin dashboard. Select BACK to direct to View Receipt dashboard in Figure 3.5.1.

4.0 Additional Features

4.1 ArrayList

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.ArrayList;

public class LoadData {
    //Load all booking data function
    public static ArrayList<String> bookingdata() {
        //Appending every data in bookingdetails.txt to an arraylist
        ArrayList<String> bookingdata = new ArrayList<String>();
        try {
            //Open new file for the bookingdetails.txt
            File bookingfile = new File("bookingdetails.txt");
            //Extract content from the bookingdetails.txt
            Scanner myReader = new Scanner(bookingfile);
            //Read every line and add into booking data arraylist
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                bookingdata.add(data);
            }
            myReader.close();
        //Call error if file is not found.
        } catch (FileNotFoundException e) {
            System.out.println("File Not found");
        }
        //return bookingdata arraylist.
        return bookingdata;
    }
```

Figure 4.1.1 Array List for LoadData class

Import ArrayList class into the LoadDate class. This feature is used because this array is resizable and does not have a fixed size for the data input.

4.2 Jtable

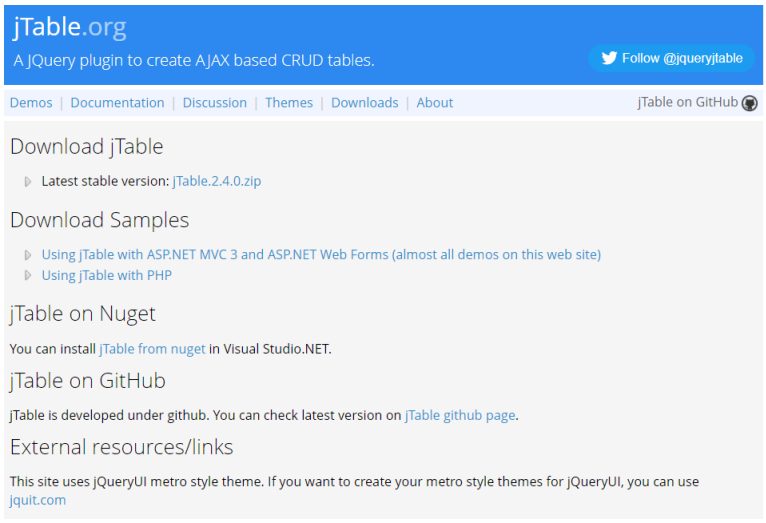

Figure 4.2.1 JTable download (jtable, 2022)

Install Jtable at https://jtable.org/Home/Downloads. Extract the zip folder into the Java application.

```
package JavaAssignment;

import javax.swing.JOptionPane;
import javax.swing.ListSelectionModel;
import javax.swing.RowFilter;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;
/**
 *
 * @author homew
 */
public class SearchBookingListForm extends javax.swing.JFrame {

    /**
     * Creates new form ViewBooking
     */
    public SearchBookingListForm() {
        initComponents();
        setLocationRelativeTo(null);

        //Allow Row selection instead of individual selection on the table list.
        bookingtable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        bookingtable.setFocusable(false);

        //Load all the data to the table list
        Booking.LoadBookingTable((DefaultTableModel) bookingtable.getModel());
    }
```

Figure 4.2.2 Import Jtable into Java application project.

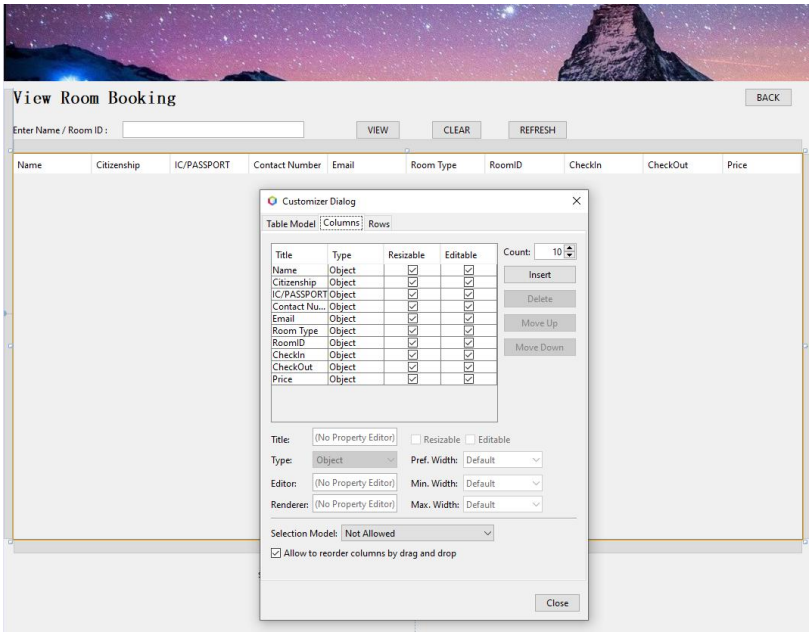

Figure 4.2.3 Customize Jtable in properties.

Add column names, types, resizable, editable properties. Once completed, select close.

4.3 Calendar

```
package JavaAssignment;

import java.io.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Scanner;
import javax.swing.JOptionPane;
/**
 *
 * @author homew
 */
public class AddBookingForm extends javax.swing.JFrame {

    /**
     * Creates new form AddBooking
     */
    public AddBookingForm() {
        initComponents();
        setLocationRelativeTo(null);

        //Set date format to yyyy/mm/dd to maintain an correct order throughout entire process
        jtxtstartdate.setDateFormatString("yyyy/MM/dd");
        jtxtenddate.setDateFormatString("yyyy/MM/dd");

        //Set date range for users
        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.DATE, 3);
        //Create date variable to get time.
        Date mindate = cal.getTime();
        cal.add(Calendar.YEAR, 1);
        Date maxdate = cal.getTime();
        //Set minimum and maximum date range
        jtxtstartdate.setDate(mindate);
        jtxtstartdate.setSelectableDateRange(mindate, maxdate);
        jtxtenddate.setDate(mindate);
        jtxtenddate.setSelectableDateRange(mindate, maxdate);
    }
```

Figure 4.3.1 Import Java utilities calendar and date.

Set calendar limit with minimum and maximum date for display. Set format of the Calendar to the general calendar format for viewing and data configuration.

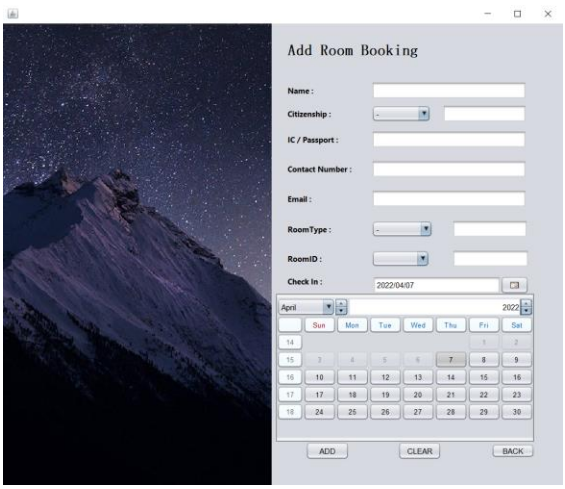

Figure 4.3.2 Calendar View

Sample of Calendar view in java Application after configuration from Figure 4.3.1.

4.4 Hide Password

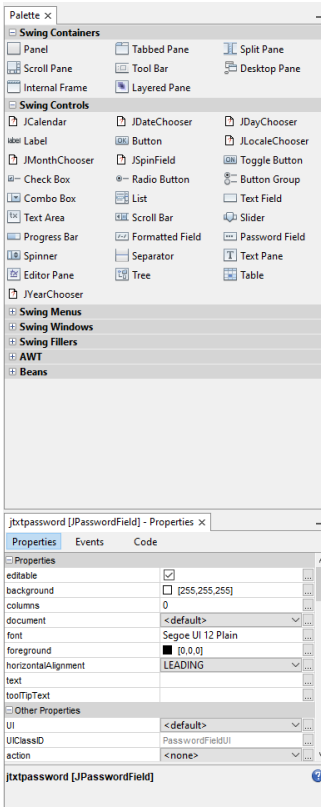

Figure 4.4.1 Hide Password

Utilize Password Filed action box under the Swing Control to hide password string to third party.

5.0 Assumptions

1. I assume user can login by providing correct login username and password.
2. I assume user can select all button in the system.
3. I assume user can add room booking details.
4. I assume all data entered in room adding and modification undergo the booking validation.
5. I assume user search and view username and room ID in search form.
6. I assume user can modify all room booking data in modification form.
7. I assume user can delete all data in search form and modification form.
8. I assume there will be no repeating room ID booked under the same date.
9. I assume the system can handle huge data reading and writing.
10. I assume user can exit the system by exit button.

6.0 References

jtable. (2022). *A JQuery plugin to create AJAX based CRUD tables.* Retrieved from jtable: https://jtable.org/Home/Downloads