

# SOFTWARE ENGINEERING

Tithira Withanaarachchi(Batch 4)-0119

Lecturer :-  
Mahesha Thejani

## Contents

Part 1.....	3
Task 1 -Resource Allocation .....	3
Part 2.....	7
Task 1-Functional and Non-Functional Requirements.....	7
Task 2- Designing the System.....	11
Task 3 - Implementing the System.....	18
Task 4 -Testing and Maintenance .....	39
Task5- Project Management .....	63
References .....	65

## Part 1

### Task 1 -Resource Allocation

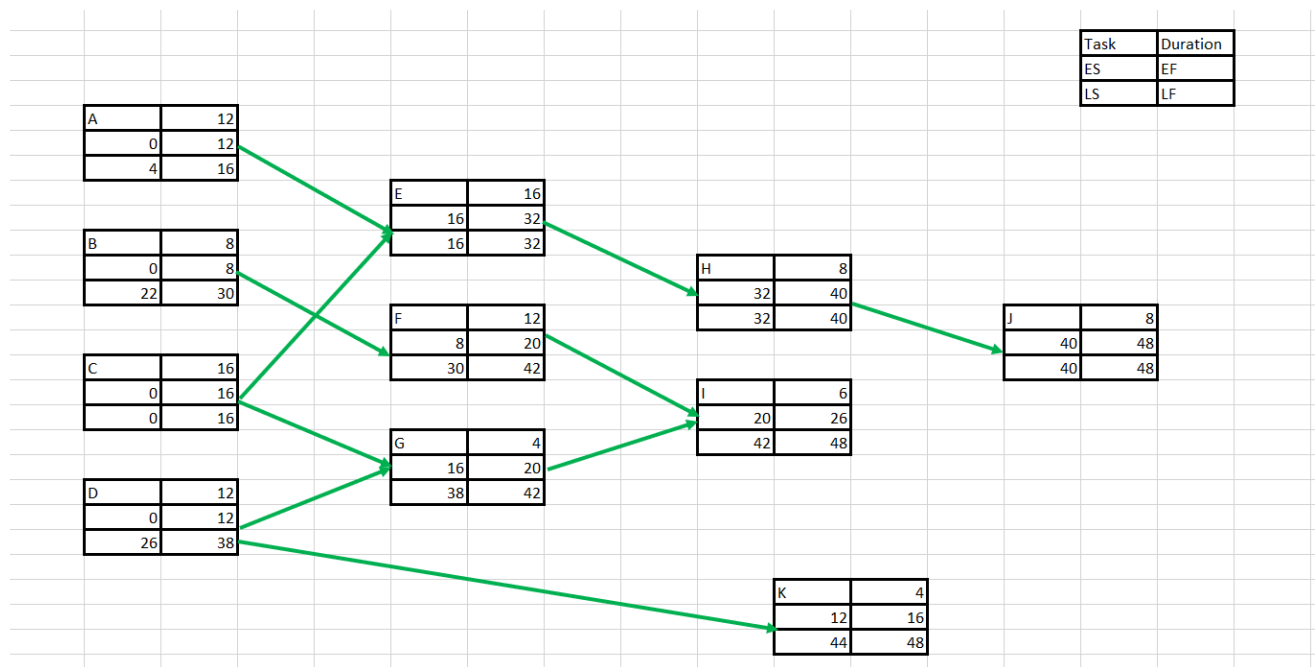


Figure 1 Network Diagram

- a. Critical Path :- C – E – H – J

Duration- 48 weeks

$$\begin{aligned}
 \text{Total Cost} &= (560+720+40+120+200+120+40+160+80+200+600)*1000 + \\
 &\quad 4000*48 \\
 &= 2840*1000+4000*48 \\
 &= 2840000+192000 \\
 &= \underline{\underline{\text{Rs.3,032,000}}}
 \end{aligned}$$

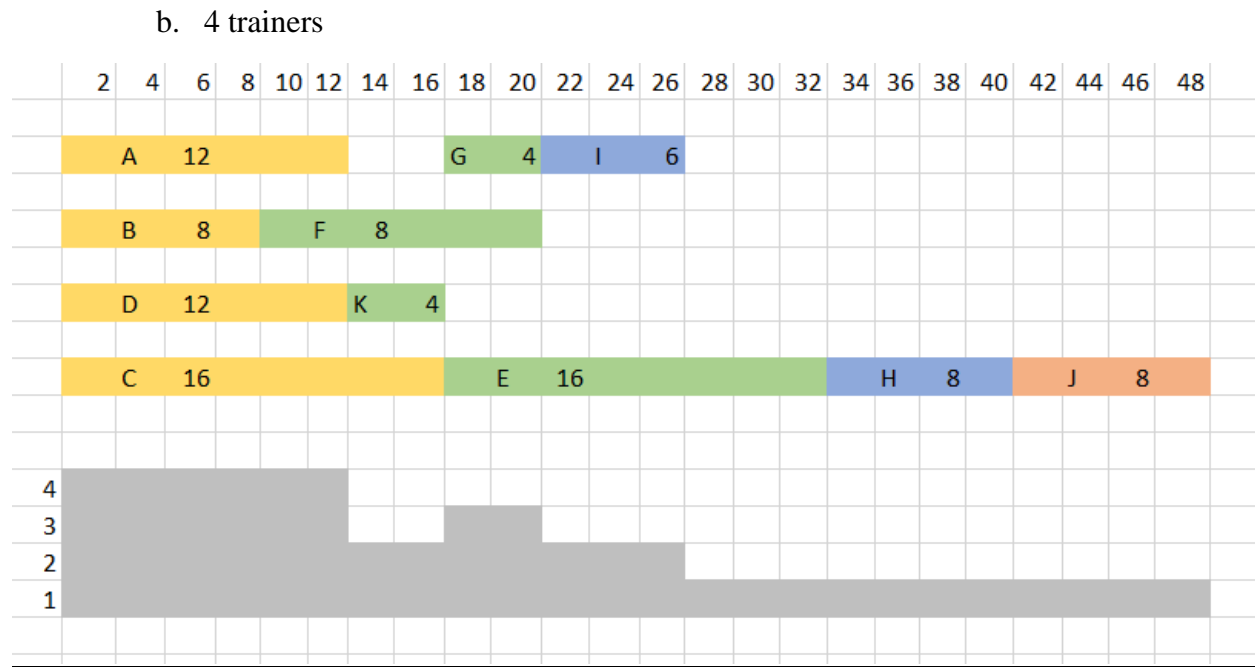


Figure 2 Gant Chart for Allocation of resources

- c.
- Yes,  
It is possible to complete the project by only using 3 trainers by changing the start and the finishing times of each task without changing the total duration for the project.

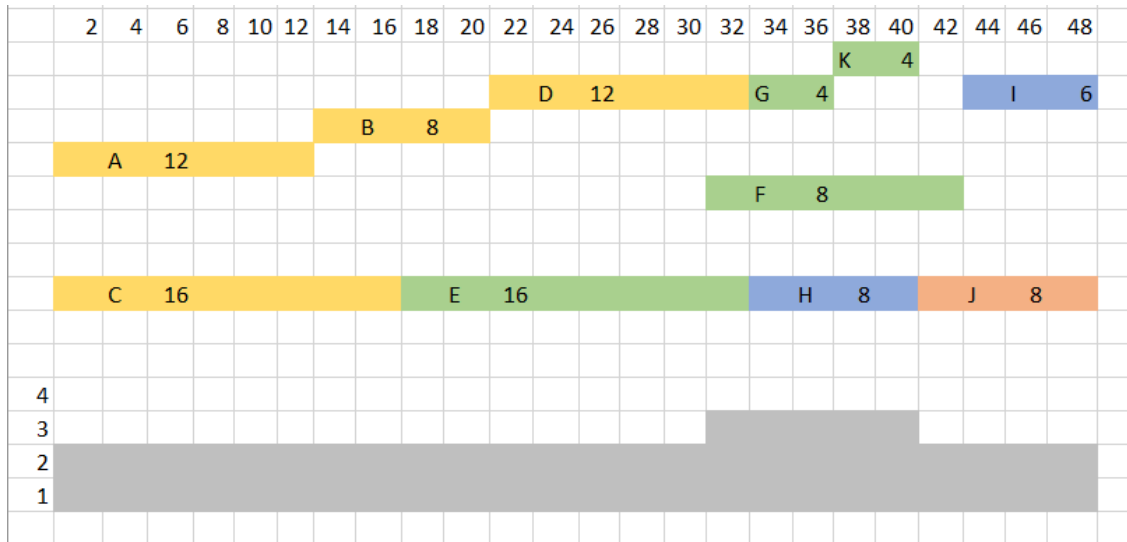


Figure 3 Gant Chart for Resource Allocation

#### Benefits –

- Less expenditure for the cost related to the trainers.
- Project can be completed with less trainers within the same time duration.

#### Drawbacks –

- The quality of the work may be reduced due to high work load for the same person.

ii. Yes,

The above tasks can be completed using only 2 trainers without increasing the length of the project more than 8 weeks.

A possible way to complete the tasks using 2 trainers without extending the project more than 8 weeks by changing the start and the finish times is shown below.

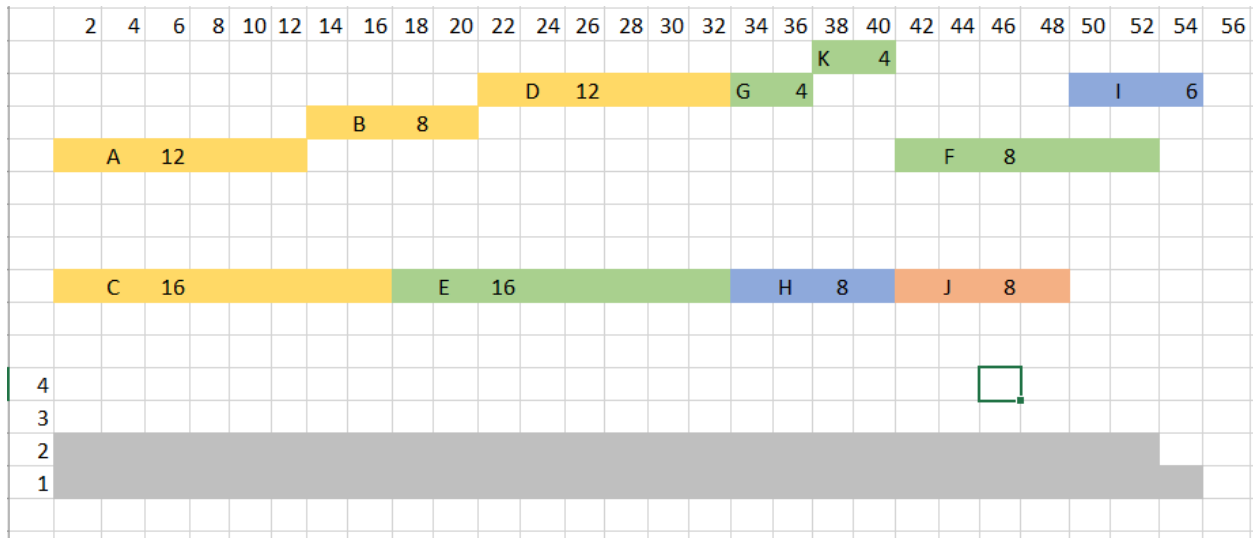


Figure 4 Gant Chart for Resource Allocation

Benefits –

- Less cost on the trainers.

Drawbacks –

- Time spent on the project will be higher.

## Part 2

### Task 1-Functional and Non-Functional Requirements

#### I. Add new Record

F1 :	Place Bookings, Add new Customer/Supervisor/Vehicle/Driver/Subsidiary.
Summary :	Bookings are placed by the customers therefore, an auto generated ID should also be recorded in order to keep track of each record. Other insertions to the System are being done by the Admin, by giving a particular ID.
Input :	Booking Details/Customer details/ Supervisor Details/ Vehicle Details/ Driver Details/ Subsidiary details.
Process :	System will enter the details to the database and record them in a database table after validating the provided inputs.
Output :	Successful Message if the data is added successfully, otherwise an error message.

#### II. Delete Records

F2:	Delete Records from the respective Database table.
Summary :	System will allow only the admin to perform the delete operation by providing the id related to the record that is needed to be deleted.
Input :	ID of the record to be deleted.
Process :	The system will first check whether a record is available with the given ID and if it's available, the

	record will be deleted after checking all the dependencies related to that record.
Output :	Successful Message will be displayed if the record is deleted successfully, otherwise it will display an error message.

### III. Search Records

F3 :	Search records from the database.
Summary :	Allows only Supervisor and the admin to search data from the database. They should be able to search data from the database table according to the required criteria.
Input :	ID of the database record/ Subsidiary Name/ Supervisor ID.
Process :	System will first check validity of the data and then check whether any data are found with the entered criteria. If data are found which satisfies the criteria, it will display them in the relevant interface.
Output :	Data will be displayed in an awt list, if the search process is successful. Otherwise, it will display an error message.



#### IV. Update Records

F4 :	Update records available in the database.( Booking/Customer/Supervisor/Vehicle/Driver/Subsidiary.)
Summary :	The system will only allow supervisor and the admins to update the records in order to perform the tasks assigned to the.
Input :	ID of the record and the data to be updated.
Process :	The system will check the validity of the updated set of values and update accordingly, if the data is valid.
Output :	Message indicating the successful update will be displayed if the update process completed without a failure. Otherwise, it will display an error message.

#### V. Generate Reports

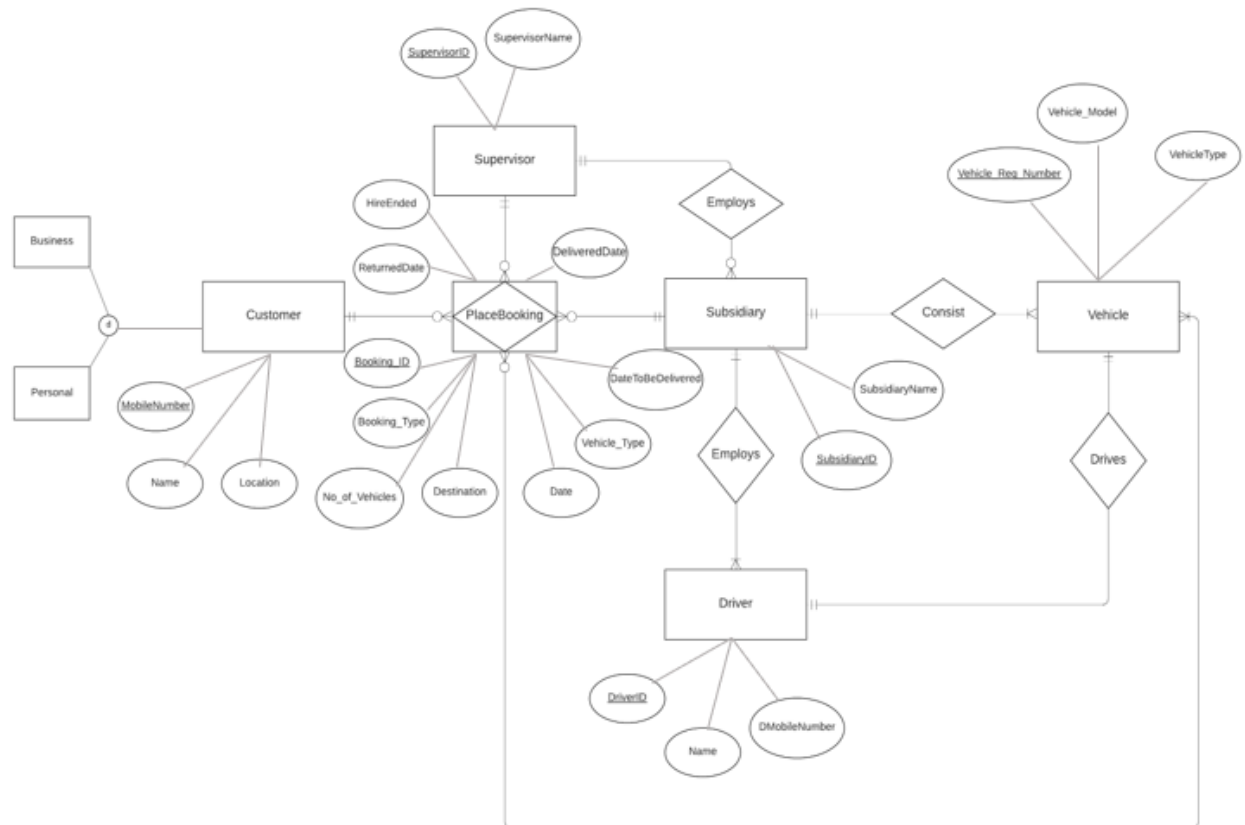
F5 :	Generate Reports.
Summary :	The system will allow admins and the supervisors to generate reports according to the available structures in the system.
Input :	–
Process :	When the user clicks on the button related to generating a particular report , the system will generate the report accordingly and display to the user.
Output :	The report will be displayed to the customer if the process is successful, otherwise it will display an error message.

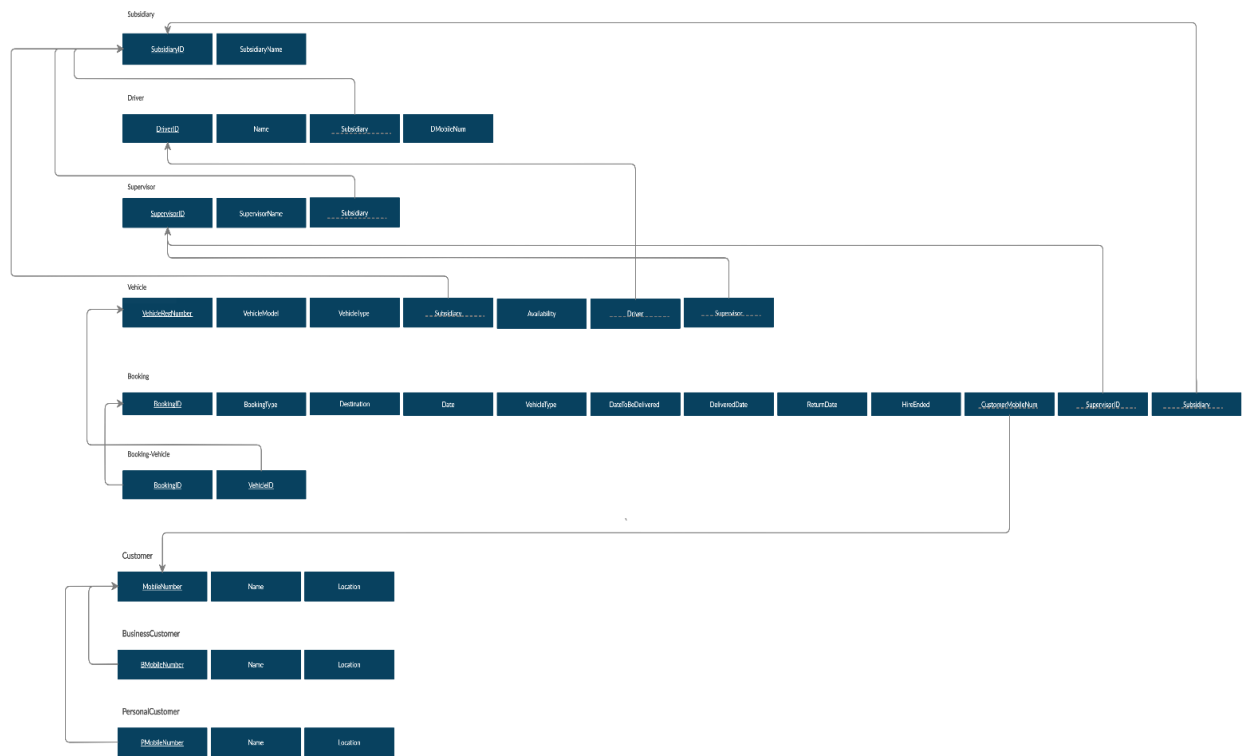
### **Non-Functional Requirements:-**

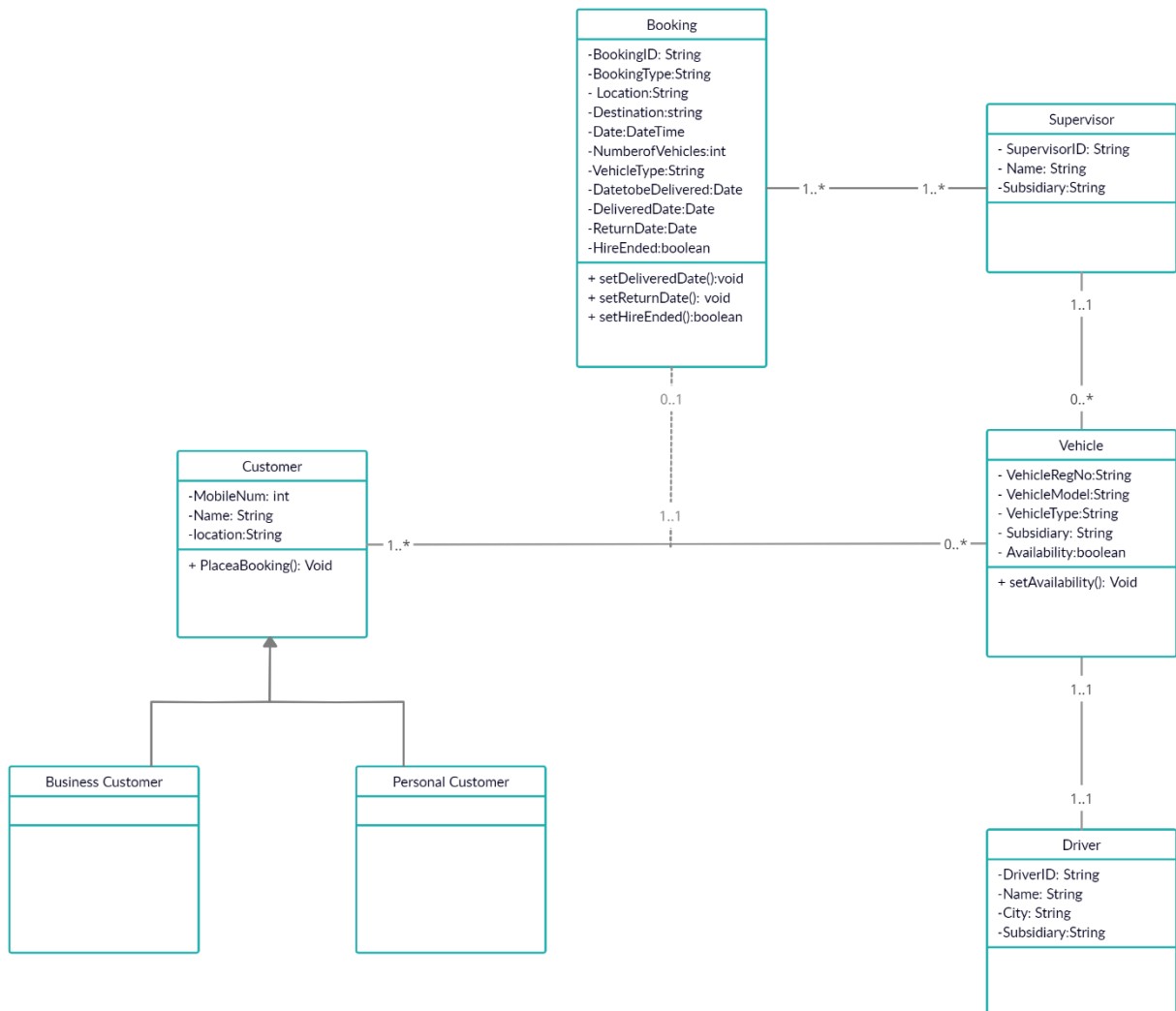
- ❖ Accuracy of time, when notifying the customer about the booking.
- ❖ Reliability of generated reports.
- ❖ Usability of the interfaces.
- ❖ Reliability of crud operations done by Customers and the Operator(Admin).

## Task 2- Designing the System

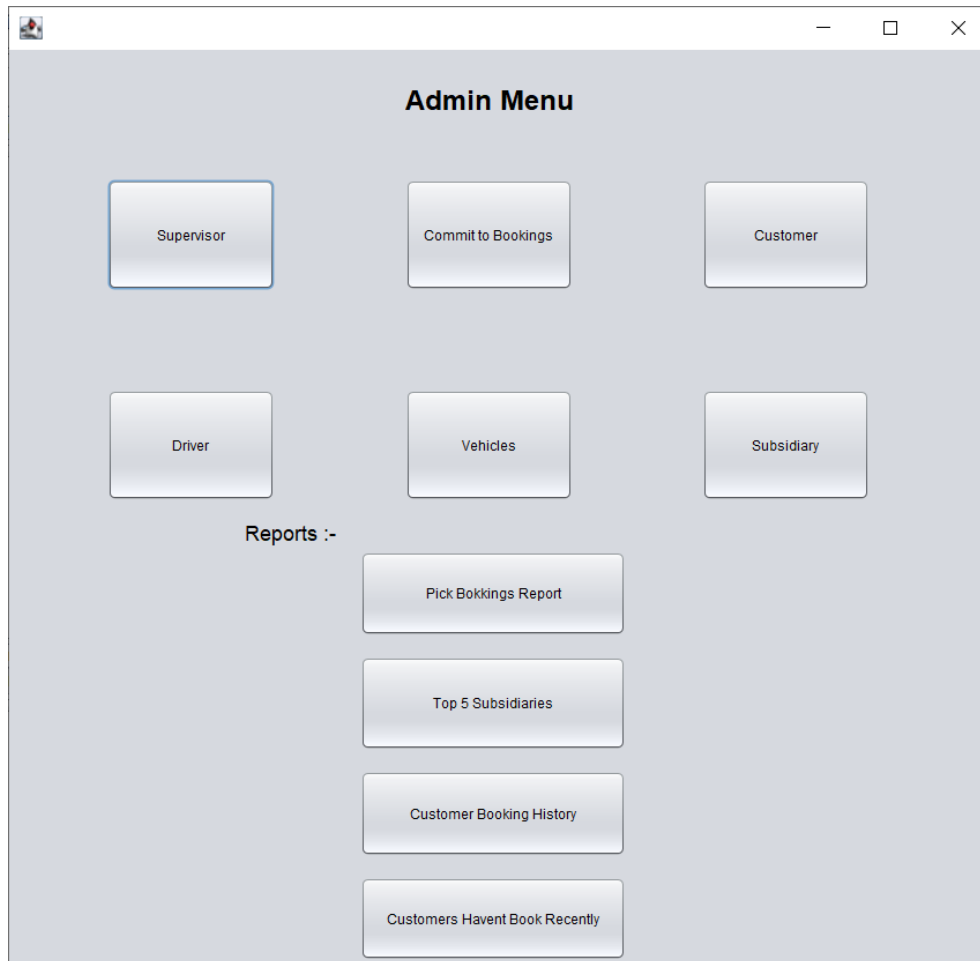
### 2.1)







## Main Menu for the Admin :-



Main interface for the Customer :-

The image shows a software window titled "Booking" with a light gray background. In the top right corner, there is a "Register" button. The main area contains several form fields: "Mobile Number" with a text input; "Booking Type" with a dropdown menu showing "Pick"; "Location" with a dropdown menu; "Destination" with a text input; "Vehicle Type" with a dropdown menu showing "Car"; "Number of Vehicles" with a spinner box set to "1"; and "Date to be Delivered" with a text input and a calendar icon. At the bottom center, there is a "Place Booking" button. The window has standard OS window controls (minimize, maximize, close) in the top right corner.

## Interface for the Supervisor

The screenshot shows a web application window titled "Update Booking(Supervisor)". The interface is organized into several sections:

- Left Section:** Contains two input fields. The first is labeled "BookingID" and the second is labeled "Vehicle Registration Number". Below the "Vehicle Registration Number" field is a button labeled "Assign Vehicle".
- Top Center Section:** Features three buttons stacked vertically: "Vehicle Delivered", "Vehicle Returned", and "Hire Ended".
- Top Right Section:** Includes a dropdown menu labeled "Select Subsidiary:" with "Colombo" selected. Below this is a button labeled "Search Available Vehicles".
- Far Right Section:** Contains an input field labeled "Enter SupervisorID:" and a button labeled "Bookings to Commit".
- Bottom Section:** Two large, empty rectangular boxes are positioned below the "Search Available Vehicles" button and the "Bookings to Commit" button, likely for displaying search results or booking details.



## 2.2)

examples of OCL related to the Class Diagram shown above.

- **Context Booking inv: NumberofVehicles>0 && NumberofVehicles<=3**
- **Context Booking inv: Date<DatetobeDelivered**
- **Context Booking inv: (DatetobeDelivered-Date) >=3**
- **Context Booking :: setHireEnded()**  
    **post:self.HireEnded=true**
- **context Booking :: setReturnDate()**  
    **pre:self.DeliveredDate!=null [1]**

OCL is a language which always evaluates a certain type, it may be a predefined OCL type or a type related to the UML diagram which the OCL is being used. And this language will not affect to any changes of the values or attributes of the classes but it will declare constraints to it [2].

Expression types used in OCL :

- Invariants.

Eg :- Context Booking inv: self.end>self.start

- Declare constraints to the class properties.
- Pre and Post conditions to operations in the class.

Eg :-pre condition

context Booking :: setReturnDate()  
pre:self.DeliveredDate!=null

Post condition

context Booking :: setHireEnded()  
Post:self.HireEnded=true [1]

## Task 3 - Implementing the System

### 3.1)

#### A) Codes Related to the Insert Button :-

The image shows a software window titled "Customer". Inside the window, there are four input fields on the left: "Mobile Number" with the value "0716891996", "Name" with the value "kamal Gunaratne", "Location" with a dropdown menu showing "Colombo", and "Type" with a dropdown menu showing "Personal". To the right of these fields is a large empty rectangular box. Above the "Mobile Number" field is a button labeled "Search by Mobile number". At the bottom of the window, there are three buttons: "Insert", "Update", and "Delete". The "Insert" button is circled in red.

### Code Implemented in the Insert Button(in the Interface for Customer Registration) :-

```
private void InsertbtnActionPerformed(java.awt.event.ActionEvent evt) {  
    //button for inserting new Customers  
    list1.removeAll();//remove all in the awt list  
    mnumcheck.setText("");  
    nmcheck.setText("");  
    loccheck.setText("");  
    statustxt.setText("");  
  
    if(mnumtxt.getText().equals("")){//Checking for empty fields  
        mnumcheck.setText("Enter the Mobile Number");  
    }else if(nmtxt.getText().equals("")){//Checking for empty fields  
        nmcheck.setText("Enter the Name");  
    }  
    else{  
        if(mnumtxt.getText().length()!=10){//Checking for Mobile Number Validity  
            mnumcheck.setText("Invalid Mobile Number");  
        }  
        else if((cdao.SearchbytelNo(mnumtxt.getText()).getmnum()==null)){//checking whether a customer has already registered.  
            Customer cob=new Customer(mnumtxt.getText(), nmtxt.getText(), loctxt.getSelectedItem().toString(), tptxt.getSelectedItem().toString());  
            boolean status=cdao.Insert(cob);  
            if(status==true){  
                statustxt.setText("Insertion Successful");  
                mnumtxt.setText("");  
                nmtxt.setText("");  
                loctxt.setSelectedItem("Colombo");  
            }else{  
                statustxt.setText("Insertion Failed");  
            }  
        }else{  
            statustxt.setText("Customer has already Registered");  
        }  
    }  
}
```

Code implemented here, first check for the empty fields and notify if one or more fields are empty. This process expects the user to enter all the fields provided in the interface in order to register a customer. Then it also checks the validity of the mobile number entered and checks whether a customer is already registered before passing the Customer object to the database. Here, the insert method implemented in the Customer Dao class is used in order to pass the new Customer(Customer object) to the Database. If the insertion is successful it notifies through a message and set the input fields of the Customer interface to default state.

### Constructor of the Customer Manager Class (Customer Dao Class) :-

```
public CustomerDao() { //Constructor of the Dao class of Customer
    try {
        Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(); //connecting to JDBC driver
    } catch (Exception ex) {
        System.out.println("Error loading Driver: " + ex.toString());
    }
}
```

This is the Constructor of the Customer Dao class which is used to connect the Front end of the System with the Database. Inside the constructor try-catch blocks are used to handle the errors that may occur related to the connection with the JDBC driver.

### Insert Method of Customer, implemented in the Customer Dao Class :-

```
public boolean Insert(Customer cust) { //method for inserting a new Customer
    Connection dbcon;
    String mnum=cust.getMnum();
    String name=cust.getNm();
    String loc=cust.getLoc();
    String type=cust.getType();
    try {
        dbcon=DriverManager.getConnection("jdbc:derby://localhost:1527/FiestaCabDB;create=true","TithiraSE","tithirase"); //Connecting to the database
        PreparedStatement ps=dbcon.prepareStatement("insert into Customer values(?,2,2,2,?)"); //query to access data
        ps.setString(1,mnum);
        ps.setString(2,name);
        ps.setString(3,loc);
        ps.setString(4,type);
        ps.executeUpdate();
        dbcon.close();
        return true;
    } catch (SQLException ex) {
        System.out.println(ex);
        return false;
    }
}
```

This method is called with a customer object (parameter ) which has to be added as a record to the Customer table. When a customer object is passed to this method, it will access each and every attribute of the object and pass them to the Customer table in the Database as a record using a query.

**B)Codes Related to the Update Button(Customer Registration Interface for Admin) :-**

The screenshot shows a web application window titled "Customer". It contains a form with the following fields and controls:

- Mobile Number:** A text input field containing "0716891996". To its right is a button labeled "Search by Mobile number".
- Name:** A text input field containing "kamal Gunaratne".
- Location:** A dropdown menu with "Colombo" selected.
- Type:** A dropdown menu with "Personal" selected.
- Buttons:** At the bottom, there are three buttons: "Insert", "Update", and "Delete". The "Update" button is circled in red.

The form is set against a light gray background with a white border. The "Update" button is highlighted with a red circle.

## Code Implemented in the Update Button(Interface for Customer Registration for Admin)

```
private void updatebtnActionPerformed(java.awt.event.ActionEvent evt) {  
    mnumcheck.setText("");  
    nmcheck.setText("");  
    loccheck.setText("");  
    statustxt.setText("");  
    list1.removeAll();  
  
    if(mnumtxt.getText().equals("")){//checking for empty fields  
        mnumcheck.setText("Enter the Mobile Number");  
    }else if(nmtxt.getText().equals("")){  
        nmcheck.setText("Enter the Name");  
    }else if(loctxt.getSelectedItem().equals("")){  
        loccheck.setText("Enter the Location");  
    }  
    else{  
        if(mnumtxt.getText().length()!=10){  
            mnumcheck.setText("Invalid Mobile Number");  
        }  
        else{  
            Customer cob=new Customer(mnumtxt.getText(), nmtxt.getText(),loctxt.getSelectedItem().toString(),tpetxt.getSelectedItem().toString());//Creating a customer object  
            boolean status=odao.update(cob);//passing customer object to the update method in the CustomerDao  
            if(status==true){  
                statustxt.setText("Customer Update Successful");  
                mnumtxt.setText("");  
                nmtxt.setText("");  
                loctxt.setSelectedItem("Colombo");  
            }else{  
                statustxt.setText("Customer Update Failed");  
            }  
        }  
    }  
}
```

Code implemented in the update button first checks for any empty fields and notify if there are empty fields. Then it checks for the validity of the Mobile number after the changes of data related to the customer. Then it makes a customer object with the updated values and pass it to the update method in the Customer Dao class with the help of a CustomerDao object.

Finally if the update is successful it will display a successful message and set the input fields to the default state otherwise if there's an error the system will display an error message.

Update Method for Customer, implemented in the Customer Dao Class :-

```
public boolean update(Customer cobj){//Mwthos in the controller class to update data in tables
    Connection dbcon;
    String mnum=cobj.getmnum();
    String name=cobj.getnm();
    String loc=cobj.getloc();
    String type=cobj.gettype();
    try{
        dbcon=DriverManager.getConnection("jdbc:derby://localhost:1527/FiestaCabDB;create=true","TithiraSE","tithirase");//database connection
        PreparedStatement ps=dbcon.prepareStatement("update Customer set Cname=?, Clocation=?, type=? where mobnumber=?");//query for updating customer info

        ps.setString(1,name);
        ps.setString(2,loc);
        ps.setString(3,type);
        ps.setString(4,mnum);
        ps.executeUpdate();
        dbcon.close();
        return true;
    }catch(SQLException ex){
        System.out.println(ex);
        return false;
    }
}
```

This is a method Boolean return, implemented inside the Customer Dao class which is used to communicate with the database for update process. This method is called by passing a customer object with the updated customer details related to an already available record in the customer table of the database. The System is connected to the Database using the Driver Manager Class. Then the attributes of the customer object passed to the method is accessed separately using the getters and they are passed to the query using the setters in order to update the record in the database table.

### C) Codes Related to the Delete Button(Customer Registration Interface):-

**Customer**

Mobile Number: 0716891996 Search by Mobile number

Name: kamal Gunaratne

Location: Colombo

Type: Personal

Insert Update **Delete**



Code Implemented in the Delete Button(Interface for Customer Registration for Admin) :-

```
private void deletebtnActionPerformed(java.awt.event.ActionEvent evt) {  
    mnumcheck.setText("");  
    nmcheck.setText("");  
    loccheck.setText("");  
    statustxt.setText("");  
    list1.removeAll();  
    if(mnumtxt.getText().equals("")){  
        statustxt.setText("Enter the mobile Number of the Customer");  
    }else{  
        Customer cob=cdao.SearchbytelNo(mnumtxt.getText()); //searching the Customer by the given ID  
        if(cob.getmnum()!=null){ //checking whether a customer is available with the entered ID  
            boolean status=cdao.delete(mnumtxt.getText());  
            if(status==true){  
                statustxt.setText("Customer "+cob.getnm()+ " has been Deleted Successfully");  
            }else{  
                statustxt.setText("Customer Deletion Failed");  
            }  
        }else{  
            statustxt.setText("Customer not found");  
        }  
    }  
}
```

First, this code checks whether the customer has entered a Mobile number in order to delete the customer. If the mobile number is provided, it will check whether, a customer is available with the given Mobile number. If a customer is not available with the provided mobile number, it will notify that a customer with the entered mobile number is not available. Otherwise, the system will call the delete method implemented in the Customer Dao class using a customer Dao object in order to delete the customer from the Customer table of the Database.

Delete method for Customer, implemented in the Customer Dao Class :-

```
public boolean delete(String mnum) {
    Connection dbcon;
    try{
        dbcon=DriverManager.getConnection("jdbc:derby://localhost:1527/FiestaCabDB:create=true","TithiraSE","tithirase");//database connection
        PreparedStatement ps=dbcon.prepareStatement("delete from Customer where mobnumber=?");//Query for deleting a customer from the customer table
        ps.setString(1,mnum);
        ps.executeUpdate();
        dbcon.close();
        return true;
    }catch(SQLException ex){
        System.out.println(ex);
        return false;
    }
}
```

This is the Delete method implemented in the customer Dao class which is used to connect the system with the database when it is needed to delete a record from the customer table of the database. This method is called by passing a string which is the mobile number of the customer that is needed to be deleted. This String is passed to a query which is used to delete data from the customer table using a setter. After the process is over, the database connection is closed. If any error occurred within the method itself, try-catch blocks are used in order to handle the error.

#### D) Codes Related to the Search Button(Customer Registration Interface)

**Customer**

Mobile Number: 0716891996

Name: kamal Gunaratne

Location: Colombo

Type: Personal

Search by Mobile number

Insert Update Delete

Code Implemented in the Search button(in the Interface for Customer Registration) :-

```
private void searchbtnActionPerformed(java.awt.event.ActionEvent evt) {  
    //button for searching a customer by the mobile number  
    try{  
        list1.removeAll();//remove all in the awt list  
        mnumcheck.setText("");  
        nmcheck.setText("");  
        loccheck.setText("");  
        statustxt.setText("");  
        if(mnumtxt.getText().equals("")){//checking for empty fields  
            mnumcheck.setText("Enter the Mobile Number");  
        }else{  
            if(mnumtxt.getText().length()!=10){  
                mnumcheck.setText("Invalid Mobile Number");  
            }else{  
  
                Customer cob= cdao.SearchbytelNo(mnumtxt.getText());//searching the customer by the entered mobile number  
                if(cob.getmnum()==null){  
                    statustxt.setText("Customer Not Found");  
                }else{  
                    //Displaying details about the customer if the customer is available with the entered mob number  
                    list1.add("Mobile Number:- " +cob.getmnum());  
                    list1.add("Customer Name:- " +cob.getnm());  
                    list1.add("Location:- " +cob.getloc());  
                    list1.add("Type:- " +cob.gettype());  
                }  
            }  
        }  
    }  
    catch(Exception ex){  
        statustxt.setText("Error: " +ex);  
    }  
}
```

In the search button, it checks whether a mobile number is entered and also checks validates it before starting the searching process. If there is any defect found in the inputs needed, it will notify accordingly.

When searching the Customer, it uses the SearchbytelNo() method implemented in the customer Dao class through an object of customer Dao. It takes the returned customer object by the search method to an object reference.

If a customer with the provided ID is null it displays as “Customer not found” otherwise it will display the customer details in the interface using an awt list.

Search Method for Customer, implemented in the Customer Dao Class (Search by Monbile Number) :-

```
public Customer SearchbytelNo(String TelNo){//Method for searching a customer by their Mobile Number
    Connection dbcon;
    Customer cust=new Customer();//Creating a default customer object
    try{
        dbcon=DriverManager.getConnection("jdbc:derby://localhost:1527/FiestaCabDB;create=true","TithiraSE","tithirase");//Database Connection
        Statement stmt=dbcon.createStatement();
        ResultSet rslt=stmt.executeQuery("select * FROM Customer WHERE Mobnumber='"+TelNo+"'");//Query to access data from the database
        while(rslt.next()){
            cust=new Customer(rslt.getString("Mobnumber"), rslt.getString("Cname"), rslt.getString("Clocation"), rslt.getString("type"));//Taking data from the database to
                                                                    a customer object*/
        }
        dbcon.close();
    }catch(SQLException ex){
        System.out.println(ex);
    }
    return cust;
}
```

This is the method implemented in the Dao class of Customer in order to communicate with the database when performing the searching process of the Customer entity. Customer mobile number is passed as a parameter when calling this method. This mobile number passed as a parameter will be set to the search query in order to search the customer with that particular Mobile number.

Finally, the search result of the Customer is returned as a Customer object by the method.

Encapsulation :-

Encapsulation is used in order to restrict the direct access to data variables and methods in a class to the outside world. This will restrict the user from directly accessing certain variables and methods through an object [3].

In java, encapsulation is achieved by using access modifiers such as private, protected. However, public methods which is commonly known as setters and getters are used to assign and get values respectively, to and from the variables in the class [4].

```
public class Customer {  
  
    private String MobileNum;//Customer Mobile Number  
    private String Name;//Customer Name  
    private String location;//Customer Location  
    private String type;//Personal or Business
```

```
public void setNum(String mnum){//setter for Customer Mobile number  
    this.MobileNum=mnum;  
}  
  
public void setName(String nm){//setter for Customer Name  
    this.Name=nm;  
}  
  
public void setloc(String loc){//setter for Customer location  
    this.location=loc;  
}  
  
public void settp(String tp){//setter for Customer Type(Personal or business)  
    this.type=tp;  
}  
  
public String getmnum(){//getter of the Customer Mobile Number  
    return this.MobileNum;  
}  
  
public String getnm(){//getter of the Customer name  
    return this.Name;  
}  
  
public String getloc(){//getter of the Customer location  
    return this.location;  
}  
  
public String gettype(){// getter of the Customer Type  
    return this.type;  
}  
}
```

Figure 5 Example for implementation using Encapsulation in the system

Polymorphysm :-

Poly = Many

morphism= forms

The term polymorphism gives the meaning “having many forms”.

If we consider an real life example, Human can be considered as one entity.,but at the same time human can also take different forms.For an exampe, from the language they speak, Country they live, occupation, age etc [5].

In the programming point of view, The human class may have the same methods speak(), Country(),age() etc, but there definition may differ from each other. Therefore, they should have different froms of the same method. This is the situation where polymorphism come in to play [5].

Similarly Polymorphysm allows us to have different ways of doing the same process.

In Java, there are 2 ways of achieving polymorphism, namely,

- 1) Compile time Polymorphism.  
This is achieved by the Method overloading.
- 2) Runtime Polymorphism.  
This is achieved by method overriding [5].

Shown below is the way that the polymorphism is implemented in the implemented system. Here, the polymorphism is achieved by Method overloading.(Compile time polymorphism)

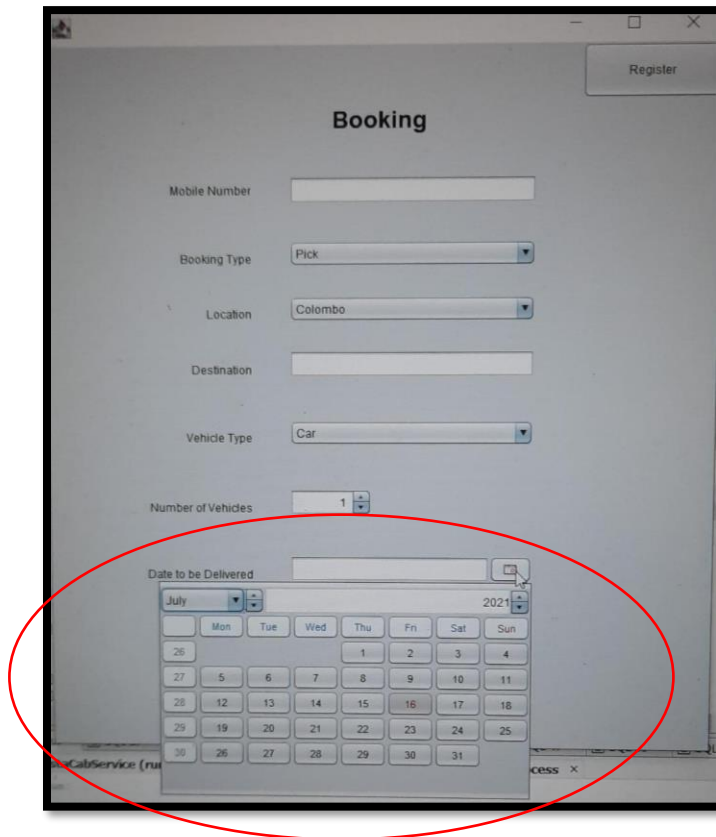
```
public Customer() { //default constructor
}

public Customer(String num) { //Constructor with one parameter
    this.MobileNum=num;
}

public Customer(String num, String name, String loc, String tp) { //Constructor with all parameters
    this.MobileNum=num;
    this.Name=name;
    this.location=loc;
    this.type=tp;
}
```

Figure 6 Example for implementation using Polymorphism in the System.

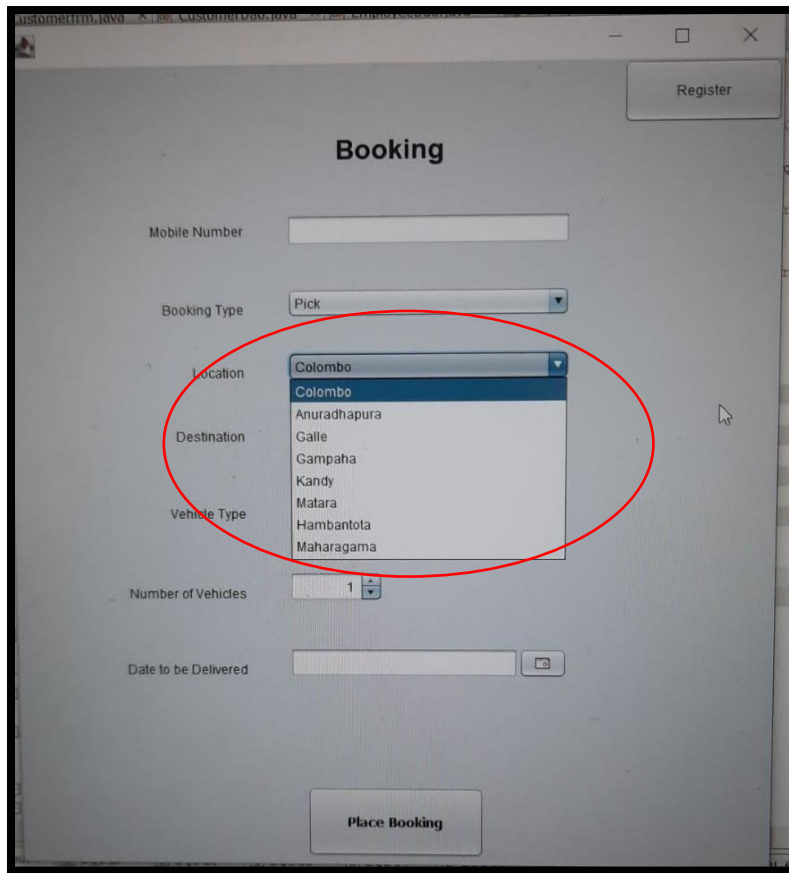
JDateChooser :-



JDateChooser is used to input dates to the System easily, just by selecting the date by clicking on the relevant date itself.



## JComboBox :-



JCombobox is a feature in of Java swing package. Combo Box is used for providing the user a dropdown menu in order to select the value as an input to the system. It has both the options to make it editable or read only.

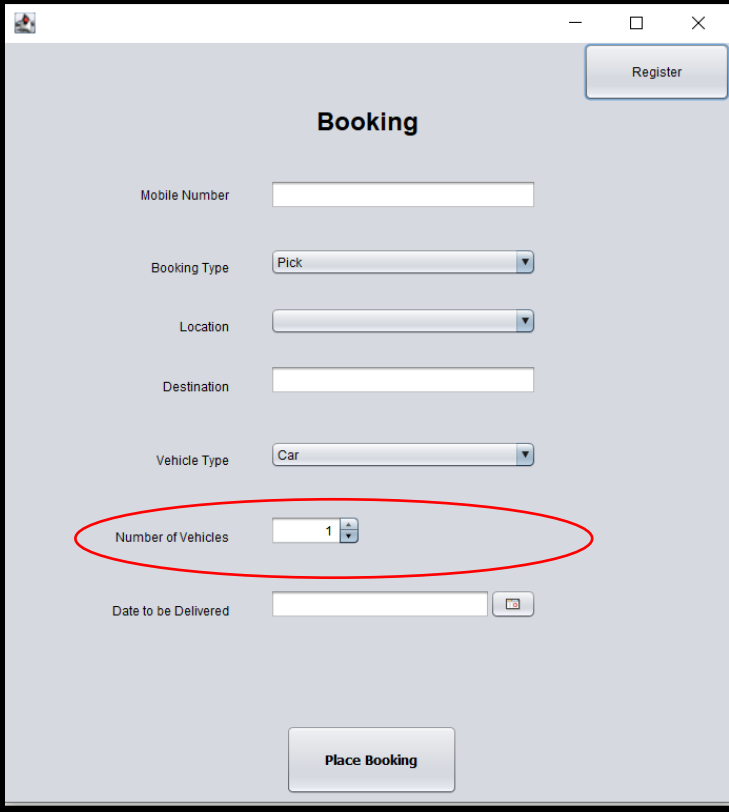
Some of the Constructors of JComboBox are,

- JComboBox()
- JComboBox(ComboBoxModel M)
- JComboBox(E[]i)
- JComboBox(Vector items) [6].

JComboBox Methods :-

- addItem(E item)
- getSelectedItem()
- removeItemAt(int i)
- setSelectedItem(Object a)

Spinner :-



JSpinner is from javax.swing package. Numbers or objects can be input through a Jspinner. It also provides the facility to type a valid data in the text space provided and input the value. By clicking the upward and downward arrows, the user can switch between the values in the input field.

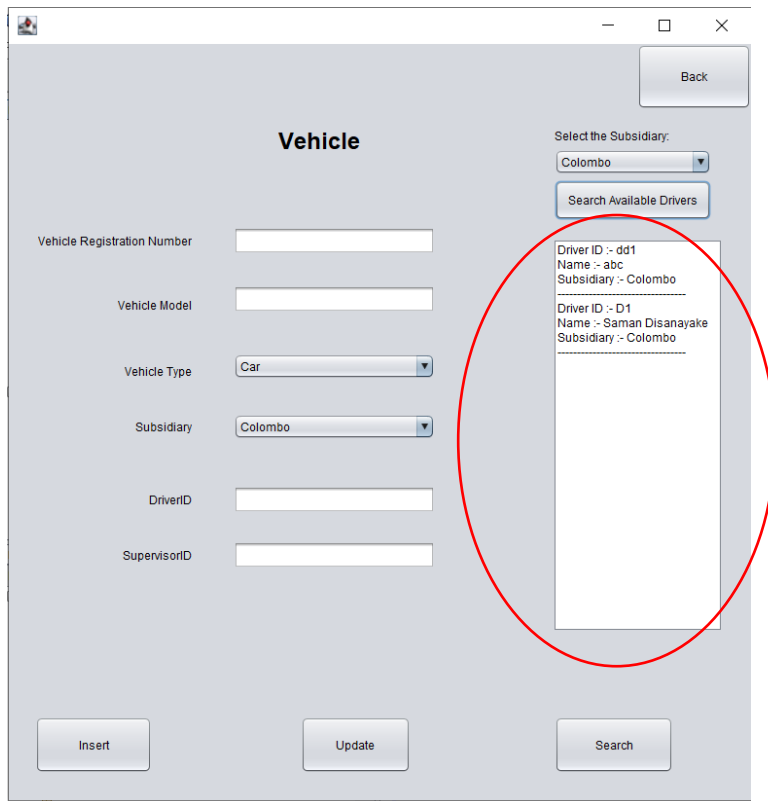
Constructors of JSpinner -

- JSpinner()
- JSpinner(SpinnerModel model)

Methods used in Jspinner –

- SpinnerListModel (List l).
- SpinnerNumberModel (int value, int max, int min, int step) [7].

List:-



List is in the java awt package. It can be used to display a list of elements to the user and it can also provide the facility to click an item in the list in order to do a particular event.

Constructors of the List Class are,

- List().
- List(int rows).
- List(int rows, boolean multipleMode).

Some of the Methods in the List Class,

- void add(String item).
- void add(String item, int index).
- int getSelectedIndex() [8].

3.2)

Exception is an unexpected event that can occur, which will interrupt the normal flow of the code.

Exception handling is the way to handle those unexpected events in order to maintain the flow of the process which is implemented by the code.

Examples of Exceptions :-

- ClassNotFoundException.
- IOException.
- SQLException.
- RemoteException.
- ArrayIndexOutOfBoundsException [9].

```
public CustomerDao() { //Constructor of the Dao class of Customer
    try{
        Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance(); //connecting to JDBC driver
    } catch (Exception ex) {
        System.out.println("Error loading Driver: " + ex.toString());
    }
}
```

Shown above is the Constructor of Customer Dao Class. Here, the try-catch blocks are used to handle the errors that may occur related to the JDBC driver.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    statustxt.setText("");
    mnumchk.setText("");
    try{
        //Creating a Date
        Date date=Calendar.getInstance().getTime();
        DateFormat dateformat=new SimpleDateFormat("yyyy-MM-dd");
        String sdate=dateformat.format(date);

        if(mnumtxt.getText().length()!=10){ //Checking the validity of the Mobile Number
            mnumchk.setText("Invalid Mobile Number");
        } else if(cdaol.SearchbytelNo(mnumtxt.getText()).getmnum()==null){ //Checking whether the Customer is Registered
            statustxt.setText("Should be Registered Before Placing a Booking");
        } else{
            if((bdao.getCustBooknotEnded(mnumtxt.getText()).getbid()==null){ //Checking whether Customer is currently on a booking which is not ended
                Booking bob=new Booking("BK"+bdao.getnewid()+"-"+sdate, bttypetxt.getSelectedItem().toString(), loctxt.getSelectedItem().toString(), Desttxt.getText(), sdate, Inte
                //Generating an ID and making a Booking Object
                bdao.insert(bob); //passing the Created Booking Object to the insert method in Booking Dao class
                statustxt.setText("Booking Placed");
            } else{
                statustxt.setText("Cannot Book again Before Ending the Current Hire");
            }
        }
    } catch (Exception ex) {
        statustxt.setText("Error" + ex);
    }
}
```

Shown above is the code implemented in the insert button of the Booking interface. Control Structures are used in the above code to maintain the accuracy of information taken from the user while using try-catch blocks to handle the unexpected errors.

```
private void searchbtnActionPerformed(java.awt.event.ActionEvent evt) {  
    //button for searching a customer by the mobile number  
    try{  
        list1.removeAll();//remove all in the awt list  
        mnumcheck.setText("");  
        nmcheck.setText("");  
        loccheck.setText("");  
        statustxt.setText("");  
        if(mnumtxt.getText().equals("")){//checking for empty fields  
            mnumcheck.setText("Enter the Mobile Number");  
        }else{  
            if(mnumtxt.getText().length()!=10){  
                mnumcheck.setText("Invalid Mobile Number");  
            }else{  
  
                Customer cob= cdao.SearchbytelNo(mnumtxt.getText());//searching the customer by the entered mobile number  
                if(cob.getmnum()==null){  
                    statustxt.setText("Customer Not Found");  
                }else{  
                    //Displaying details about the customer if the customer is available with the entered mob number  
                    list1.add("Mobile Number:- " +cob.getmnum());  
                    list1.add("Customer Name:- " +cob.getnm());  
                    list1.add("Location:- " +cob.getloc());  
                    list1.add("Type:- " +cob.gettype());  
                }  
            }  
        }  
    }  
    }catch(Exception ex){  
        statustxt.setText("Error: " +ex);  
    }  
}
```

Shown above is the code implemented in the search button of the customer interface. Here, control structures are used to minimize the inaccuracies while try-catch blocks are used to handle the unexpected errors.

```

public ArrayList<Booking> searchbysupID(String SupID){//Method for searching Bookings by Supplier ID
    Connection dbcon;
    Booking bk=new Booking();//Creating a Booking Object
    ArrayList<Booking> blist=new ArrayList<Booking>();//Creating an Array List of the type Booking
    try{
        dbcon=DriverManager.getConnection("jdbc:derby://localhost:1527/FiestaCabDB;create=true","TithiraSE","tithirase");//Database connection
        Statement stmt=dbcon.createStatement();
        ResultSet rslt=stmt.executeQuery("select * FROM Booking WHERE Supervisor='"+SupID+"'");//Query to filter Bookings according to the given ID
        while(rslt.next()){//creating Booking objects with the Searched results and passing it to the Array List
            bk=new Booking(rslt.getString("BookingID"), rslt.getString("Btype"),rslt.getString("location"), rslt.getString("Destination"), rslt.getString("Date"), rslt.getInt("
            blist.add(bk);
        }
        dbcon.close();//Closing the database connection
    }catch(SQLException ex){
        System.out.println(ex);//passing the error to the console, if an error is occurred(SQL Exceptions)
    }
    return blist;//returning the ArrayList
}
}

```

Shown above is a method in the Booking Dao class which connects with the database to search bookings by the supervisor ID.

Here, try-catch blocks are used to handle the errors that may occur due to the process of communicating with the Database, which are known as SQL Exceptions.

## Task 4 -Testing and Maintenance

### 4.1)

Software testing is process which the testers go through the system while giving attention even to very minute details. Because of this, Software testing has become very challenging, when it comes to doing it in an accurate manner. As a solution for doing it effectively, testing is done in different stages. The main stages of testing are,

- ❖ Unit testing.
- ❖ Integration Testing.
- ❖ System Testing.
- ❖ Acceptance Testing [10].

These main stages of testing can be again divided in to two secondary sections according to the time period they are done in the software development life Cycle. According to this classification,

Unit testing                      }  
Integration testing            } Verification Stage

System Testing                }  
Acceptance Testing            } Validation Stage  
  [10]

#### Unit Testing-

This is the very first stage of testing which should be done by going through the modules of the system separately and giving attention to even very minute issues. This stage of testing should be done by a tester who has a thorough knowledge about the subject [10].

#### Integration Testing –

In this stage, the testing is done to verify whether the modules of the system are working properly together. Even though the modules of the system work perfectly individually, it cannot be assured that the system will be a perfect outcome. The reason for this is, even though the modules function separately, when they are integrated, the functionalities and the performance of the integrated system will be changed due to collaboration issues within the modules [10].

#### System Testing –

This is the stage of testing, the whole system is being tested as one by combining all other integrated systems. In system testing, the functional requirements are being tested according to the requirements of the customer [10].

### Acceptance Testing-

Even though the system is fully tested by the testers, it cannot be assured that an error will not arise while the actual users of the system use it. The reason for this is , the ones who does testing will not have the knowledge about the day to day business tasks. Therefore, when the actual users are using the system, the faults of the system can still be identified [10].

As mentioned above, the different stages of software testing should be done in the FCS Cab service System in order to achieve an optimum result.

For an example, each function such as insert , delete , search etc can be considered as modules which should be tested in the unit testing stage.

An interface which contains the functionality of different modules such as insert , delete , update should be tested in the integration testing phase.

Then the Whole System after connecting the interfaces and making the Main menus , it should be tested again as a whole in the System testing phase.

After the development of the system, it should be again tested by letting the actual users to use the system.



4.2)

<b>Administration System for FCS Cab Service</b>	
Test Plan ID	01
Brief Description about the Database.	Administration System of FCS Cab Service will mainly manage the information regarding Bookings, Customers, Supervisors, Vehicles, Driver, Subsidiaries and controlling the bookings.
Introduction to the testing objectives.	<p>This document will mainly describe the plan for testing Administration System of FCS Cab Service and this document will support the following objectives as well,</p> <ul style="list-style-type: none"> <li>• Testing the features of the system.</li> <li>• Recommended Test Requirements.</li> <li>• Describing the recommended test strategies(Approach).</li> <li>• List the deliverables of the test activities.</li> </ul>
Testing Items.	Crud operations and other special functions such as Generating Reports and Controlling the Bookings.
Features to be tested.	Insert, Update, Delete, Search, Generate Repots
Testing Environment.	PC, JDBC Client Driver, NET Beans IDE 8.2
Testing Approach	Black Box Testing
Testing Tasks	Test Plan, Test Environment, Test Result(Test Case)
Schedule	17 <sup>th</sup> July 2021, 12.06pm to 2pm

4.3)

Administration System for the FCS Cab Service	
TEST CASE	
Test Unit: Insert Customer	Tester: Tithira Withanaarachchi
Test Case ID: 01	Test Type: Black box Testing
Description: Insert a new Customer to the System by the Admin.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date- 17 <sup>th</sup> July 2021
Title:- Customer Registration	Test Execution Time:- 12.40pm- 1.00pm

Step No	Test Steps	Testing Data	Expected Result	Actual Result	Status(Pass/Fail)	Notes
01	Login to the System.					
02	Access the relevant interface for adding a new Customer.	Mobile Number= '0716891996' Name= 'Kamal gunaratne' Location= 'Colombo' Type= 'Personel'	Display a Message "Insertion Successful"	Successful Message Displayed.	Pass	

**Customer**

Mobile Number

Name

Location

Type

**Customer**

Mobile Number

Name

Location

Type

Insertion Successful

Administration System for the FCS Cab Service	
TEST CASE	
Test Unit: Search Customer	Tester: Tithira Withanaarachchi
Test Case ID: 02	Test Type: Black box Testing
Description: Search an available Customer by using the Mobile Number by the Admin.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date- 17 <sup>th</sup> July 2021
Title:- Search by Mobile number	Test Execution Time:- 1.00pm- 1.20pm

Step No	Test Steps	Testing Data	Expected Result	Actual Result	Status(Pass/Fail)	Notes
01	Login to the System.					
02	Access the relevant interface for Searching a Customer.	Mobile Number= '0716891996'	Display the Searched Results in the List.	Searched Result Displayed	Pass	

Administration System for the FCS Cab Service	
TEST CASE	
Test Unit: Update	Tester: Tithira Withanaarachchi
Test Case ID: 03	Test Type: Black box Testing
Description: Update an available Customer in the System by the Admin.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date- 17 <sup>th</sup> July 2021
Title:- Customer Update by the Mobile Number.	Test Execution Time:- 12.40pm- 2.00pm

Step No	Test Steps	Testing Data	Expected Result	Actual Result	Status(Pass/Fail)	Notes
01	Login to the System.					
02	Access the relevant interface for Updating an available Customer.	Mobile Number= '0716891996' Name= 'Namal Gunasekara' Location= 'Colombo' Type= 'Personel'	Display a Message "Customer Update Successful"	Update successful Message Displayed.	Pass	

**Customer**

Mobile Number:

Name:

Location:

Type:

Mobile Number:- 0716891996  
Customer Name:- kamal Gunar  
Location:- Colombo  
Type:- Personal

**Customer**

Mobile Number:

Name:

Location:

Type:

Customer Update Successful

Administration System for the FCS Cab Service	
TEST CASE	
Delete	Tester: Tithira Withanaarachchi
Test Case ID: 04	Test Type: Black box Testing
Description: Delete a Customer from the System by the Admin.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date- 17 <sup>th</sup> July 2021
Title:- Customer Registration	Test Execution Time:- 12.40pm- 2.00pm

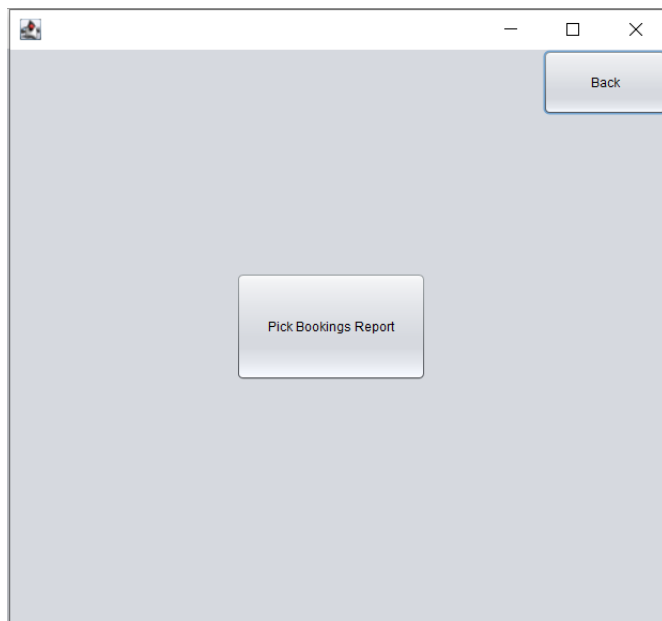
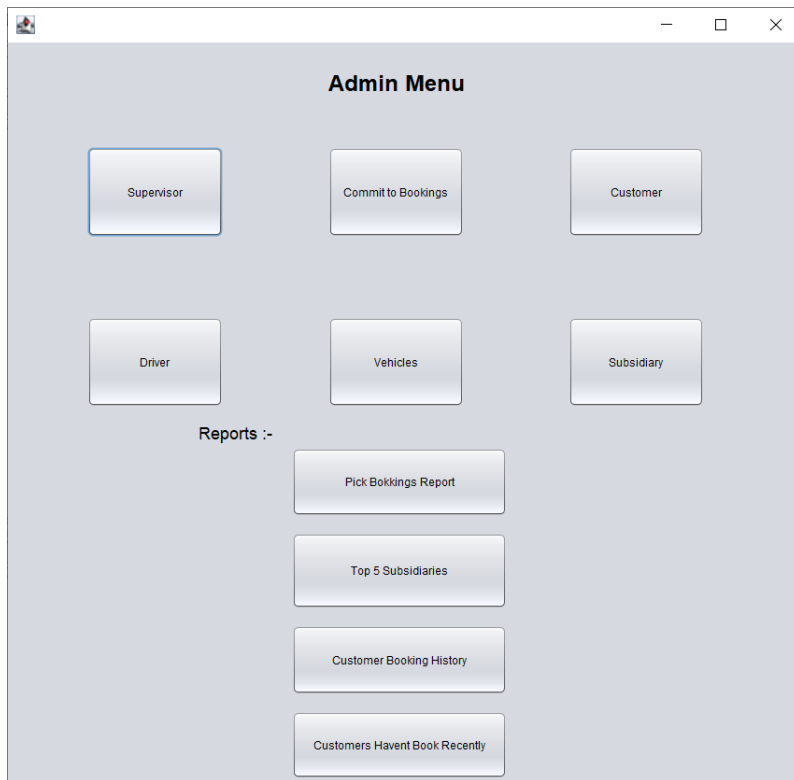
Step No	Test Steps	Testing Data	Expected Result	Actual Result	Status(Pass/Fail)	Notes
01	Login to the System.					
02	Access the relevant interface for Deleting a Customer.	Mobile Number= '0716891996'	Display a Message "Customer <Name> has been Deleted"	Successful Deletion Message Displayed	Pass	

The screenshot shows a web application window titled "Customer". It contains a search form with the following fields: "Mobile Number" (with the value "0716891996"), "Name", "Location" (a dropdown menu showing "Colombo"), and "Type" (a dropdown menu showing "Personal"). A "Search by Mobile number" button is located next to the Mobile Number field. Below the search fields, a text box displays the search results: "Mobile Number:- 0716891996", "Customer Name:- Namal Guna", "Location:- Colombo", and "Type:- Personal". At the bottom of the window, there are three buttons: "Insert", "Update", and "Delete".

Administration System for the FCS Cab Service	
TEST CASE	
Test Unit: Report Generation	Tester: Tithira Withanaarachchi
Test Case ID: 05	Test Type: Black box Testing
Description: Generate Reports According to the Requirement of the Admin.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date- 17 <sup>th</sup> July 2021
Title:- Generating Reports by the Admin.	Test Execution Time:- 12.40pm- 2.00pm

Step No	Test Steps	Testing Data	Expected Result	Actual Result	Status(Pass/Fail)	Notes
01	Login to the System.					
02	Click on the Button Relevant to the Report that is needed to be Generated.		Display the Report Generated using the Jasper Viewer.	Report Generated was displayed using the Jasper Viewer.	Pass	





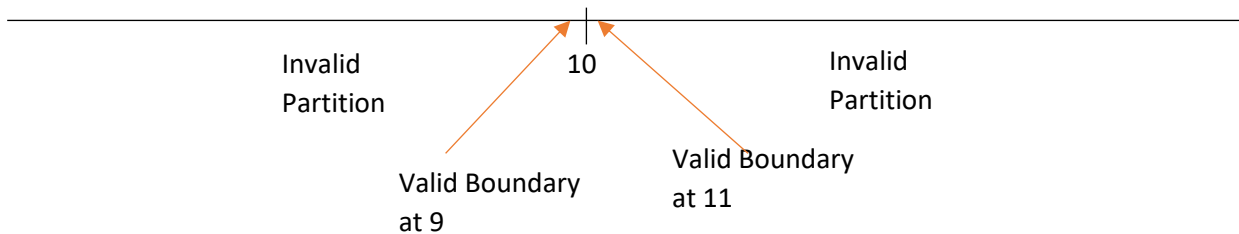
JasperViewer

Pick Booking Report

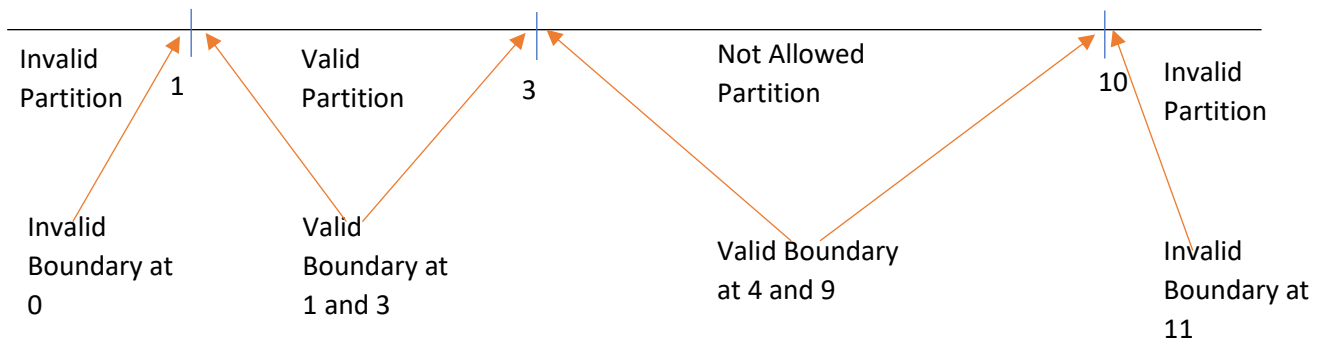
Customer Location	DESTINATION	Booking Date	No.of Vehicles	DeliveryDate	Returned Date	Customer Mobile
Colombo	Gampaha	15/07/21 00:00	1	16/07/21 00:00	null	0711231231

### Black Box Test Designing:-

When registering a new Customer ,in the System implemented , it only accepts Mobile Numbers with 10 digits. For this constraint, Black box Testing can be done as shown below.

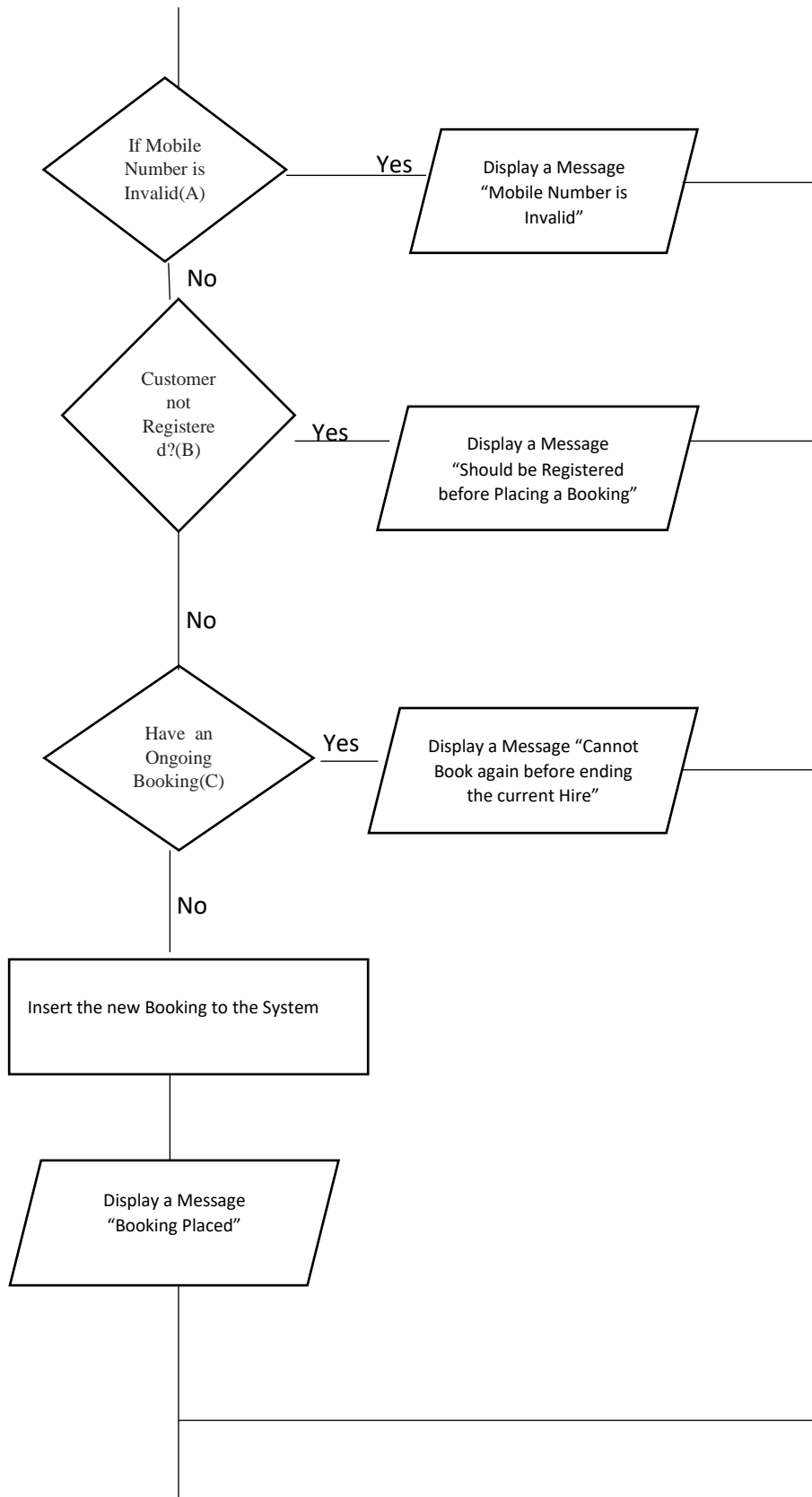


When placing a booking, the above system allows to book only 3 vehicles maximum for one Booking. Using Back box test designing for this scenario is shown below.



## White Box Testing:-

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    statustxt.setText("");  
    mnumchk.setText("");  
    try{  
        //Creating a Date  
        Date date=Calendar.getInstance().getTime();  
        DateFormat dateformat=new SimpleDateFormat("yyyy-MM-dd");  
        String sdate=dateformat.format(date);  
  
        if(mnumtxt.getText().length()!=10){//Checking the validity of the Mobile Number  
            mnumchk.setText("Invalid Mobile Number");  
        }else if(cdao1.SearchbytelNo(mnumtxt.getText()).getmnum()==null){//Checking whether the Customer is Registered  
            statustxt.setText("Should be Registered Before Placing a Booking");  
        }else {  
            if((bdao.getCustBooknotEnded(mnumtxt.getText()).getbid()==null){//Checking whether Customer is currently on a booking which is not ended  
                Booking bob=new Booking("BK"+bdao.getnewid()+"-"+sdate, btypetxt.getSelectedItem().toString(), loctxt.getSelectedItem().toString(), Desttxt.getText(), sdate, In  
                //Generating an ID and making a Booking Object  
                bdao.insert(bob); //passing the Created Booking Object to the insert method in Booking Dao class  
                statustxt.setText("Booking Placed");  
            }else{  
                statustxt.setText("Cannot Book again Before Ending the Current Hire");  
            }  
        }  
    }catch(Exception ex){  
        statustxt.setText("Error" + ex);  
    }  
}  
  
private void mnumtxtKeyPressed(java.awt.event.KeyEvent evt) {  
  
}
```



Shown above is the logic behind the source code shown in the image.

To identify the test cases , White Box test designing technique can be used as shown below.

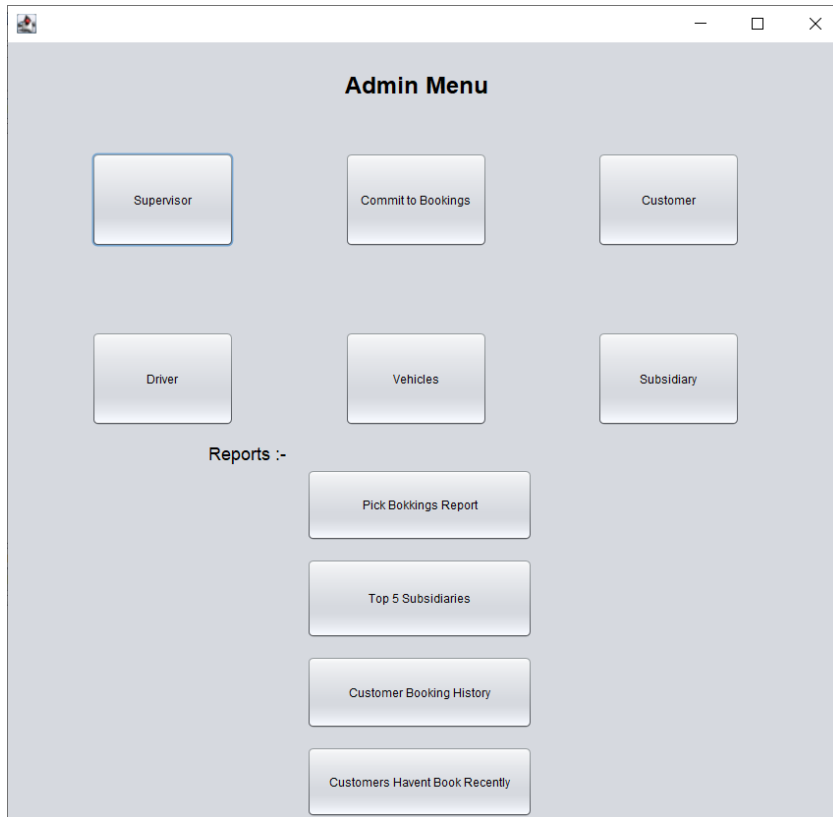
4 Test Cases are needed for the 100% statement Coverage of the above scenario:-

- Condition A true
- Condition A false B true
- Condition A false B false C true
- Condition A false B false C false

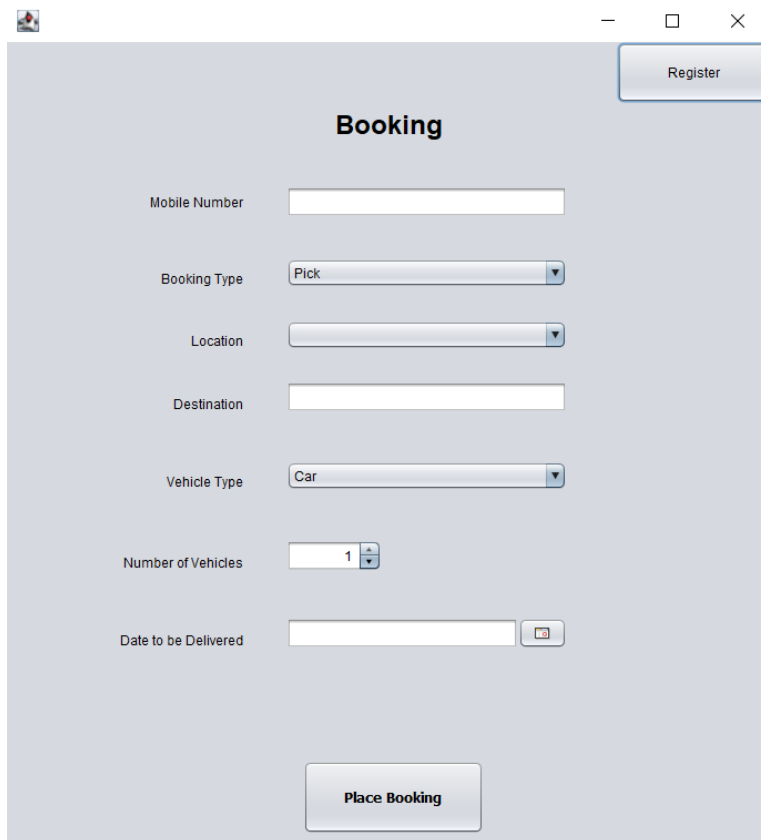
#### 4.4) User Manual :-

### Getting Started

#### Main Menu for Admin



### Interface for the Customer:-

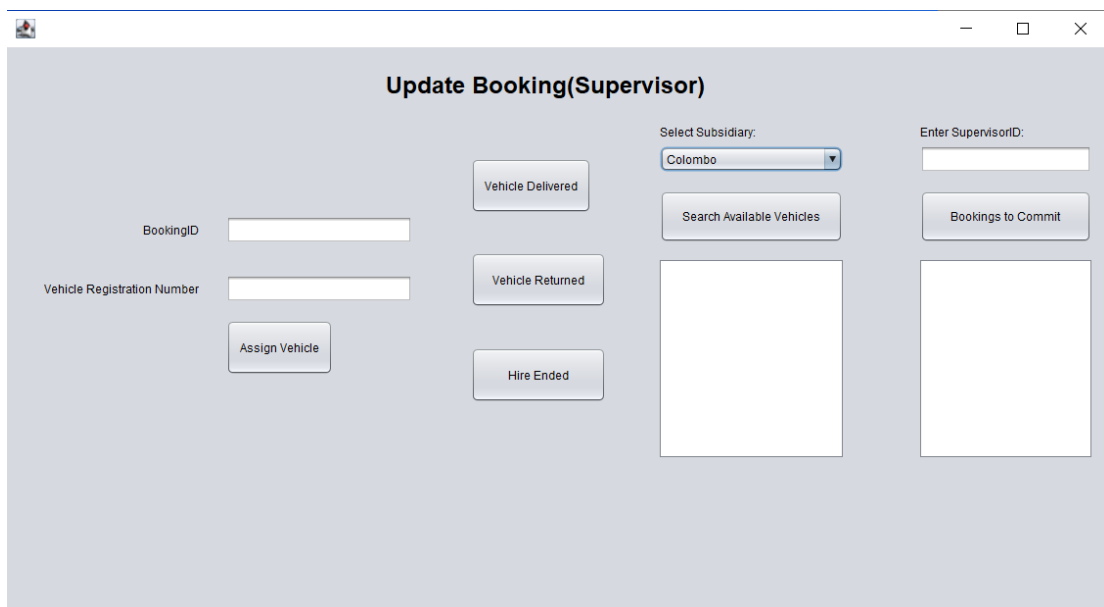


The screenshot shows a web application window titled "Booking". In the top right corner, there is a "Register" button. The main form contains the following fields and controls:

- Mobile Number:** A text input field.
- Booking Type:** A dropdown menu with "Pick" selected.
- Location:** A dropdown menu.
- Destination:** A text input field.
- Vehicle Type:** A dropdown menu with "Car" selected.
- Number of Vehicles:** A numeric input field with a spinner, currently set to "1".
- Date to be Delivered:** A text input field with a calendar icon on the right.

At the bottom center of the form is a button labeled "Place Booking".

### Main Interface for the Supervisor:-



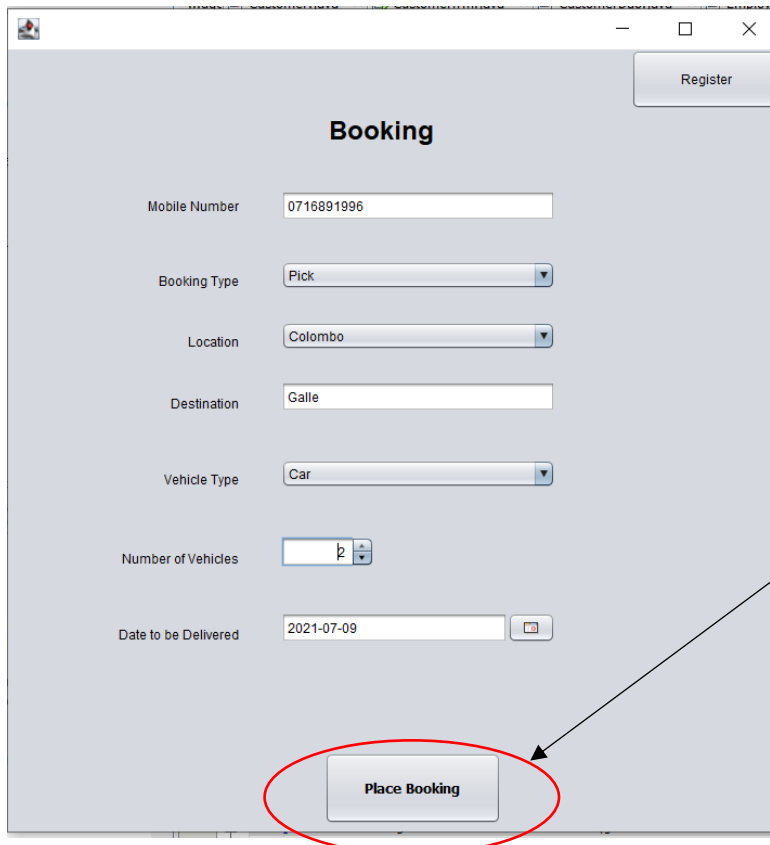
The screenshot shows a web application window titled "Update Booking(Supervisor)". The interface is divided into several sections:

- Left Section:** Contains two text input fields labeled "BookingID" and "Vehicle Registration Number". Below the "Vehicle Registration Number" field is an "Assign Vehicle" button.
- Center Section:** Contains three buttons stacked vertically: "Vehicle Delivered", "Vehicle Returned", and "Hire Ended".
- Right Section:**
  - At the top, "Select Subsidiary:" with a dropdown menu showing "Colombo".
  - Below it is a "Search Available Vehicles" button.
  - At the bottom right is a "Bookings to Commit" button.
  - Below the "Search Available Vehicles" button is a large empty rectangular box.
  - Below the "Bookings to Commit" button is another large empty rectangular box.
- Far Right Section:** Labeled "Enter SupervisorID:" with a text input field.



## Placing a Booking

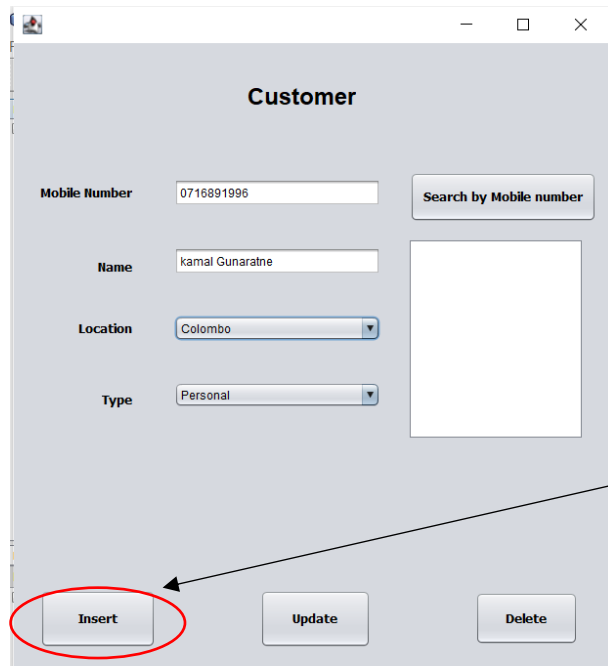
- **Mobile number:-** Mobile number has a String data type and it should have 10 characters to be a valid input.
- **Booking Type :-** Booking type provides 2 options by a drop-down menu to choose one, whether to make the booking only to pick or for both pick and drop.
- **Location :-** The area which belongs to the location that the vehicle should be delivered to, should be chosen from the list given by the dropdown menu.
- **Destination :-** Destination should be typed in the text field provided.
- **Vehicle Type:-** Vehicle Types are provided in the user interface using a dropdown menu which displays the available types of vehicles. The Required Vehicle type should be selected by the user from the dropdown menu.
- **Number of Vehicles :-** This is an integer type attribute which the user should input the values using the spinner provided. User is restricted to book only up to 3 vehicles for a Booking.
- **Date:-** A date chooser is provided to choose the date that the booking should be delivered to the customer. The user has to select the date, month and the year from the date chooser.



After Entering Valid Input data, as mentioned above, provided button should be clicked to place the booking.

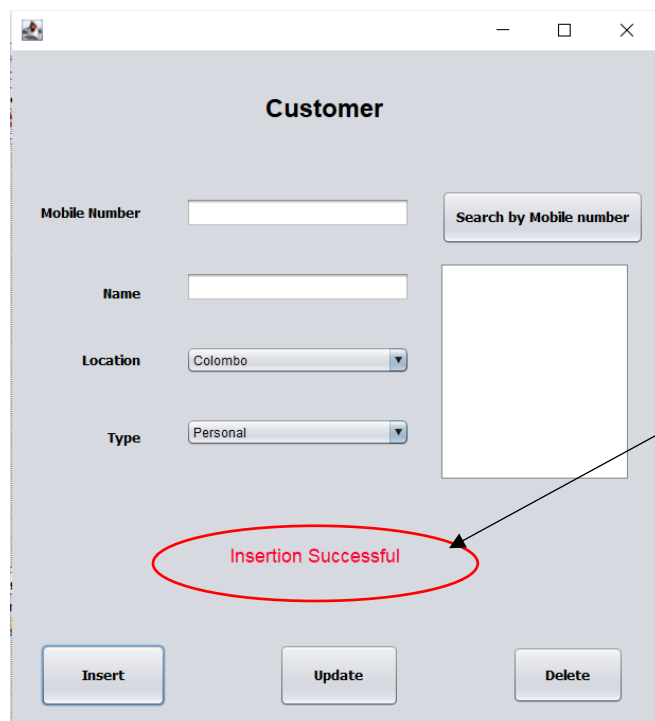
## Crud Operations:-

### Insert (Inserting a customer):-



The screenshot shows a web application window titled "Customer". It contains several input fields: "Mobile Number" with the value "0716891996", "Name" with the value "kamal Gunaratne", "Location" with a dropdown menu set to "Colombo", and "Type" with a dropdown menu set to "Personal". There is a "Search by Mobile number" button next to the Mobile Number field. At the bottom of the form, there are three buttons: "Insert", "Update", and "Delete". The "Insert" button is circled in red, and an arrow points from a text box to it.

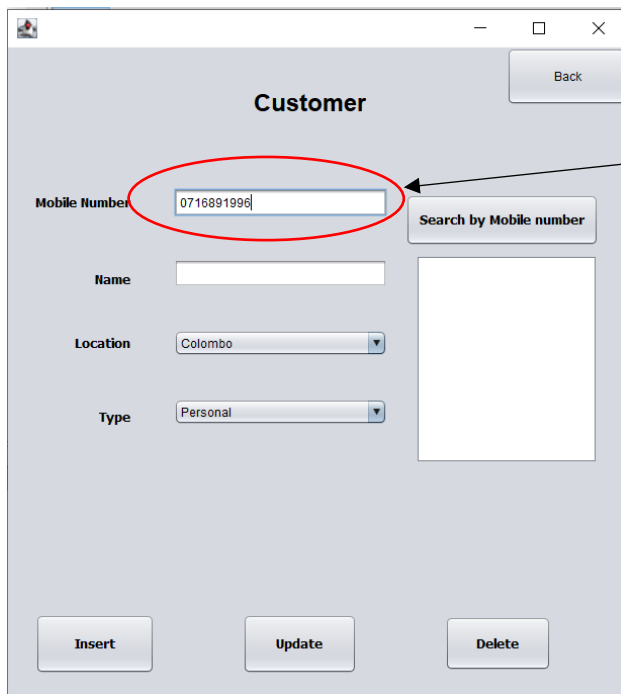
After entering valid data related to the customer, similarly as in booking form, the insert button should be clicked.



The screenshot shows the same "Customer" form as before, but now the "Insert" button is highlighted with a red oval. Below the "Insert" button, the text "Insertion Successful" is displayed in red. An arrow points from a text box to this message.

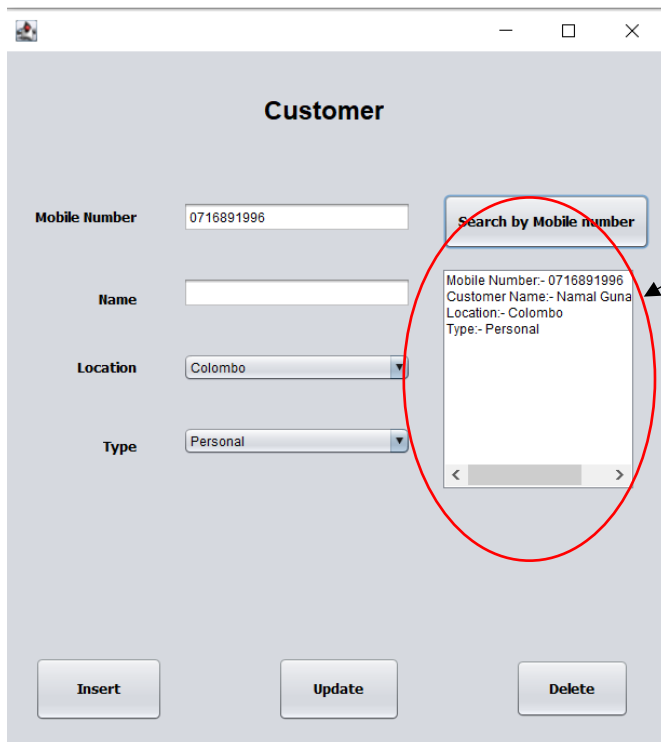
A successful message will be displayed if the data have been added to the system successfully.  
If not, an error message will be displayed here.

Search:-



The screenshot shows a window titled "Customer" with a "Back" button in the top right. The form contains the following fields: "Mobile Number" (text input with "0716891996" entered and highlighted by a red circle), "Name" (text input), "Location" (dropdown menu with "Colombo" selected), and "Type" (dropdown menu with "Personal" selected). To the right of the "Mobile Number" field is a button labeled "Search by Mobile number". At the bottom of the form are three buttons: "Insert", "Update", and "Delete".

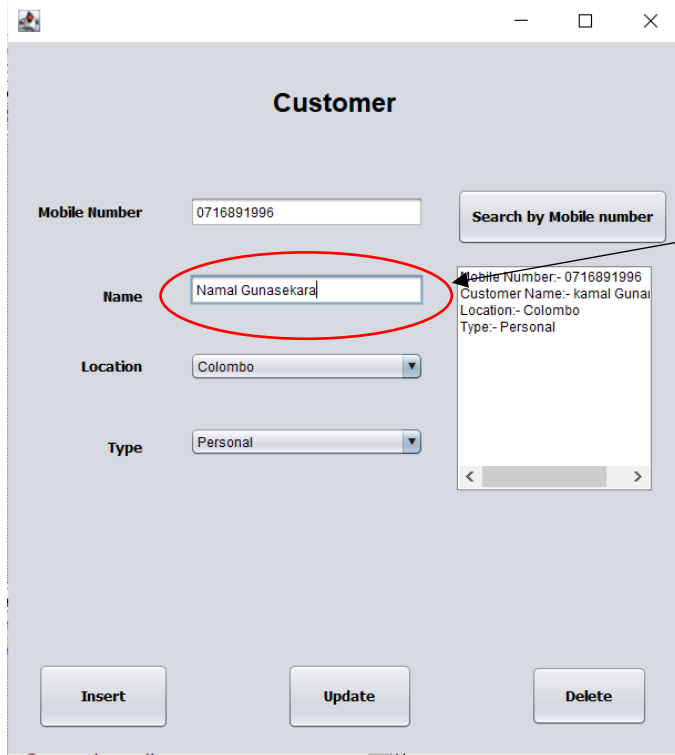
The Mobile Number of the Customer should be entered in the space provided, in order to search a customer who is already registered to the system.



The screenshot shows the same "Customer" window, but now the "Search by Mobile number" button is highlighted by a red circle. To the right of the button is a list box containing the following text: "Mobile Number:- 0716891996", "Customer Name:- Namal Guna", "Location:- Colombo", and "Type:- Personal". The list box has a scrollbar at the bottom. The other fields and buttons remain the same as in the previous screenshot.

The search results after searching a customer will be displayed through the awt List in the interface.

## Update



The screenshot shows a web application window titled "Customer". It contains a form with the following fields: "Mobile Number" (text input with "0716891996"), "Name" (text input with "Namal Gunasekara", circled in red), "Location" (dropdown menu with "Colombo"), and "Type" (dropdown menu with "Personal"). To the right of the "Name" field is a search box labeled "Search by Mobile number" with a button. Below the search box is a text area displaying the search results: "Mobile Number:- 0716891996", "Customer Name:- kamal Gunal", "Location:- Colombo", and "Type:- Personal". At the bottom of the form are three buttons: "Insert", "Update", and "Delete".

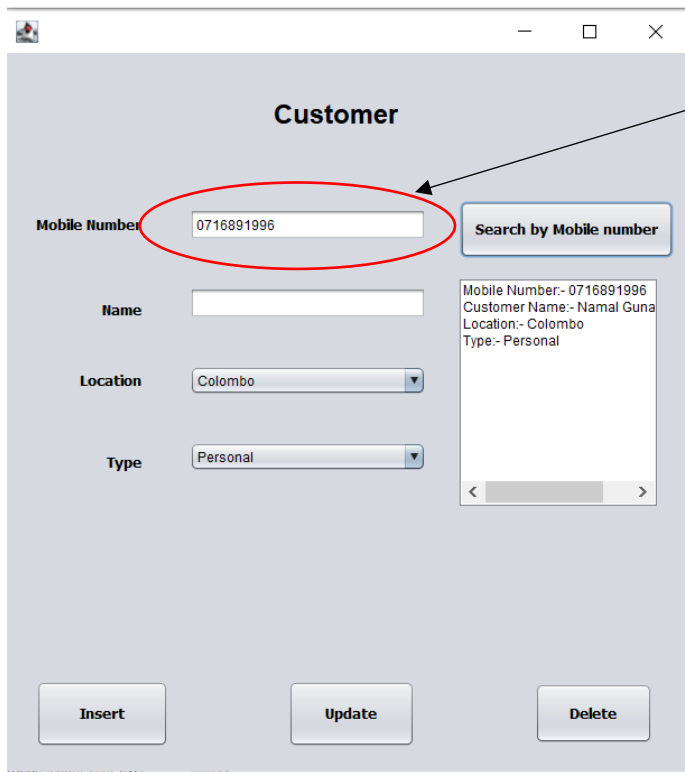
Records related to customer can be updated after searching the customer by the mobile number. The user(Admin) will be able to manipulate all the attributes except the mobile number in the update process.



The screenshot shows the same "Customer" form as above, but with the "Name" field empty. A red oval highlights the text "Customer Update Successful" at the bottom of the form, which is a message indicating a successful update. The "Update" button is highlighted with a blue border. The search results area is empty.

If the Record is updated successfully, a successful message will be displayed here.  
Otherwise, an error message will be displayed.

Delete:-



**Customer**

Mobile Number: 0716891996

Name:

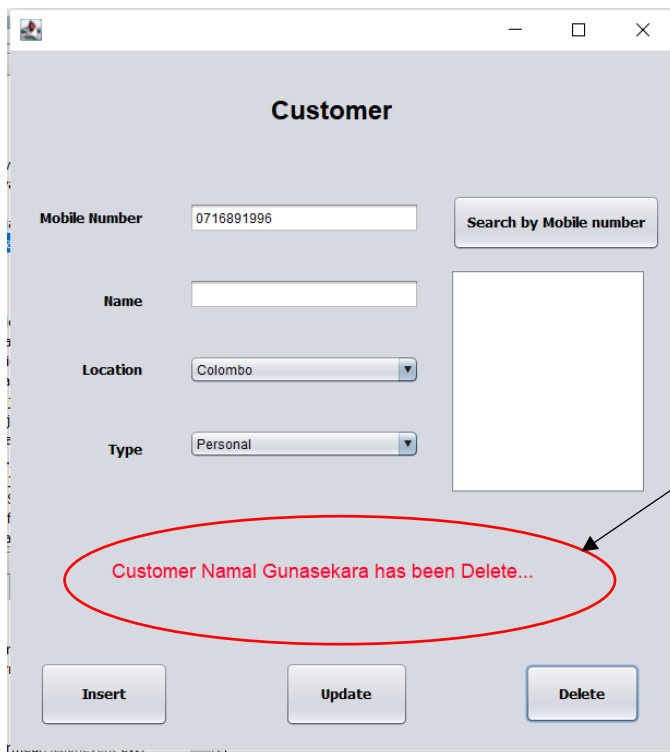
Location: Colombo

Type: Personal

Search by Mobile number

Mobile Number:- 0716891996  
Customer Name:- Namal Guna  
Location:- Colombo  
Type:- Personal

Insert Update Delete



**Customer**

Mobile Number: 0716891996

Name:

Location: Colombo

Type: Personal

Search by Mobile number

Customer Namal Gunasekara has been Delete...

Insert Update Delete

4.5)

There are 4 main software maintenance strategies. namely,

- Corrective software Maintenance.
- Adaptive software maintenance.
- Perfective Software Maintenance.
- Preventive Software Maintenance [11].

Corrective Software Maintenance :-

Corrective software maintenance is basically accepting to correct any kind of fault in the system and maintain. This addresses faults related to any part of the software such as design, logic etc. These errors are usually being detected by the actual user who uses the software. But with the corrective maintenance approach, the development team can find those error earlier and it will also affect the development companies' reputation [11].

Adaptive Software Maintenance :-

Adaptive software maintenance is suitable when the business environment and other technology related are rapidly changing. This strategy focusses on the updating technologies and their rapidly changing requirements and maintain the software [11].

Perfective Software Maintenance :-

As the real users interact with the system, they may come up with new suggestions according to their experience on the system. And also, with the time, the development team and the business which the system is implemented will also be able to identify which functionalities are rapidly using and what are the facilities which is rarely used. By those observations and comments, system can updated in a more accurate way which will make the users more convenient when using the system and also to build a system that will lead to satisfying the business goals. This strategy focusses on those criteria when maintaining the system [11].

Preventive Software Maintenance:-

This strategy focusses on maintaining the system in a way that it will long last with the changes and which will make the users easy and understandable about the system [11].

For the System implemented for the FCS Cab Service, It will be better to adapt to the corrective software maintenance at the early stage of delivery and switching to the perfective software maintenance after sometime, since in the earlier stage, there is a high possibility to have errors which are detected while using. Therefore, in the early stage those errors can be corrected by identifying them earlier as possible by following the corrective strategy and it will be more suitable to switch to the perfective strategy to provide the users an improved and convenient system.

## Task5- Project Management

### 5.1)

Software Development deals with complex processes and it often deals with critical time constraint when delivering software. For these kind of critical and complex projects, it will not even be possible to achieve the expected system, or the project outcome by the client without a proper project Management.

If a team start doing a project without a any management of the project, anyone of them will not have a proper idea about the project and how to achieve the final goal of the project. And also, the different members of the team may have different ideas about the project. This will lead to occurrence of error more often [12].

When developing the system for the FCS Cab service in the above scenario, Project Management is use in,

- Writing proposal to the Client to get the acceptance.
- Project planning and scheduling in order to manage the time and allocate the resources.
- Make the Cost Estimations.
- Controlling the Project by monitoring and reviews.
- Can write reports and do presentations in order to keep the stakeholders updated and build their trust while increasing the understandability of the group members about the project.

A- Requirement Gathering and Analysis.(2 weeks)

B- Designing the Logical Diagrams(ER Diagrams, Flow Charts, Pseudo Codes)(2 weeks)

C- Interface Designing(1 week).

D- Creating the Database (2 weeks).

E- Implementing the Database Connection and the Communication between the Interfaces and the Database(3 weeks).

F- Coding the interfaces(3 weeks).

G- Unit testing(1 week).

H- Integrated Testing(1 week).

I- System Testing(1 week).

J- Acceptance Testing(2 weeks).

Assumption :- Though the testing is done parallelly while developing, all the stages of testing is done after the implementation as well.

Tasks	Dependencies	Duration(in weeks)
A	-----	2
B	A	2
C	B	1
D	B	2
E	D	3
F	E	3
G	C, D, E, F	1
H	G	1
I	H	1
J	I	2

Table 1 Task list for the FCS Cab Service System Project

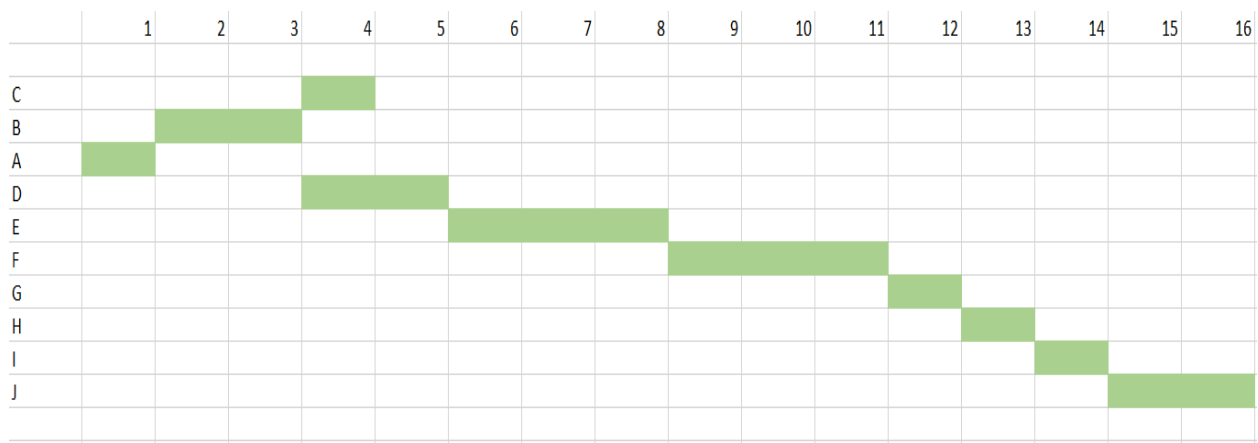


Figure 7 Gant Chart for the FCS Cab service System Project



## References

- [1] D. Demuth, "OCL," 2009. [Online]. Available: <https://st.inf.tu-dresden.de/files/general/OCLByExampleLecture.pdf>. [Accessed 17 07 2021].
- [2] "Modeling Languages," [Online]. Available: <https://modeling-languages.com/ocl-tutorial/>. [Accessed 16 07 2021].
- [3] "sumo logic," [Online]. Available: <https://www.sumologic.com/glossary/encapsulation/>. [Accessed 16 07 2021].
- [4] "JournalDev," [Online]. Available: <https://www.journaldev.com/12496/oops-concepts-java-example>. [Accessed 16 07 2021].
- [5] "Geeks for Geeks," [Online]. Available: <https://www.geeksforgeeks.org/polymorphism-in-java/>. [Accessed 16 07 2021].
- [6] "GeeksforGeeks," [Online]. Available: <https://www.geeksforgeeks.org/java-swing-jcombobox-examples/>. [Accessed 16 07 2021].
- [7] "GeeksforGeeks," [Online]. Available: <https://www.geeksforgeeks.org/java-swing-jspinner/>. [Accessed 16 07 2021].
- [8] "GeeksforGeeks," [Online]. Available: [https://www.tutorialspoint.com/awt/awt\\_list.htm](https://www.tutorialspoint.com/awt/awt_list.htm). [Accessed 16 07 2021].
- [9] "javaTpoint," [Online]. Available: <https://www.javatpoint.com/exception-handling-in-java>. [Accessed 16 07 2021].
- [10] "UTOR," [Online]. Available: <https://u-tor.com/topic/software-testing-stages-explained>. [Accessed 17 07 2021].
- [11] "CAST," [Online]. Available: <https://www.castsoftware.com/glossary/Four-Types-Of-Software-Maintenance-How-They-Help-Your-Organization-Preventive-Perfective-Adaptive-corrective>. [Accessed 17 07 2021].
- [12] "ORASES," [Online]. Available: <https://orases.com/the-importance-of-project-management-for-software-development/>. [Accessed 17 07 2021].