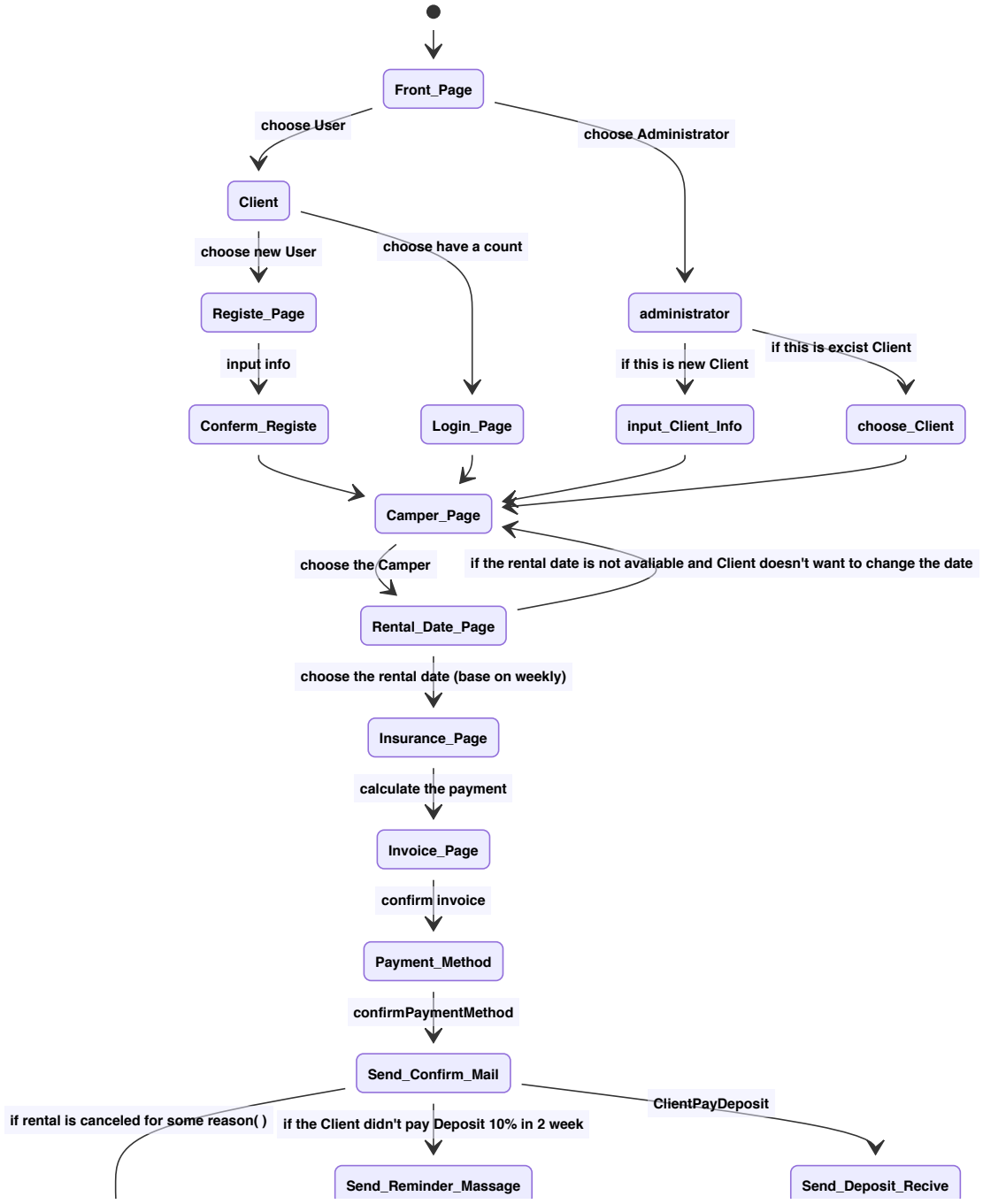
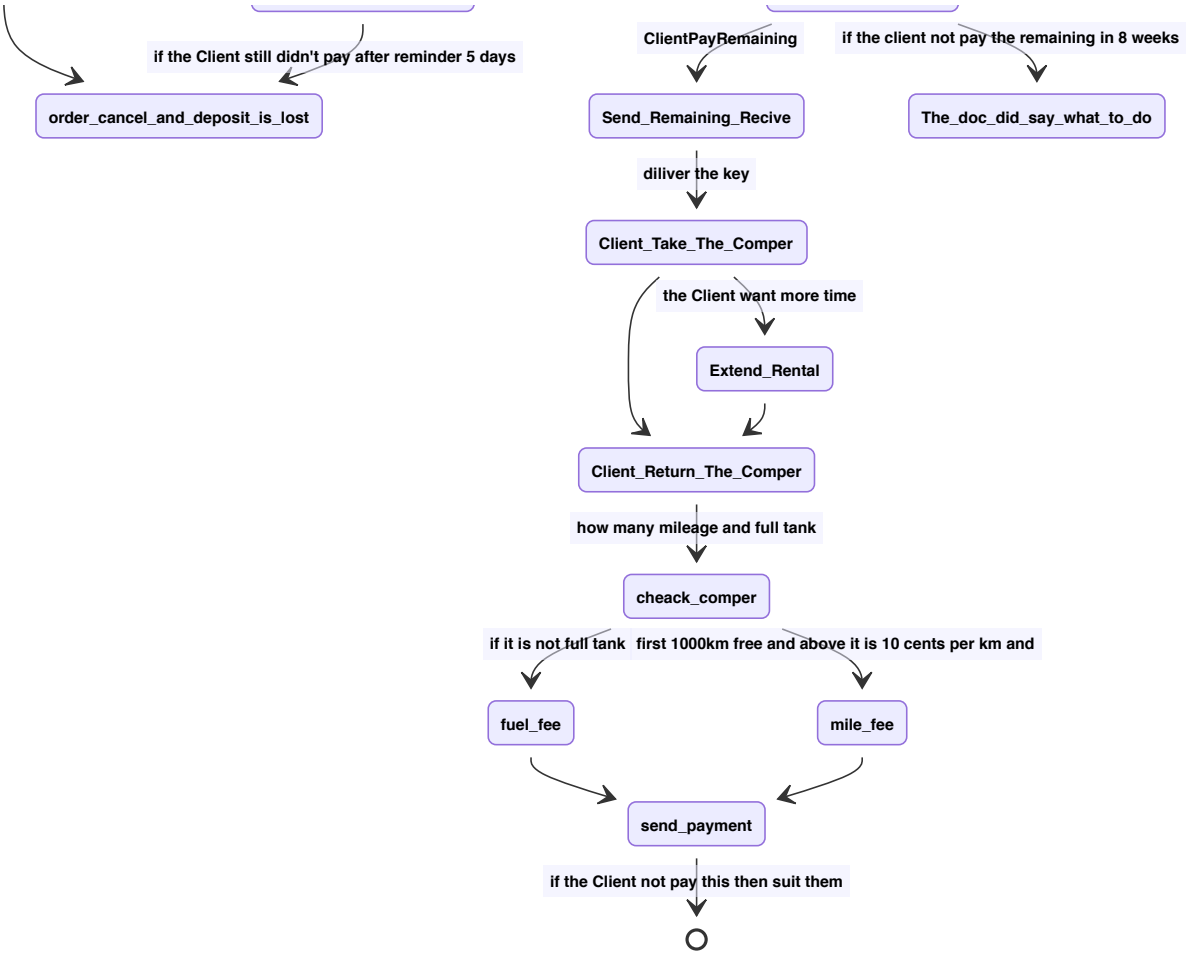
State Diagram




if the Client still didn't pay after reminder 5 days
order_cancel_and_deposit_is_lost
ClientPayRemaining
if the client not pay the remaining in 8 weeks
Send_Remaining_Recive
The_doc_did_say_what_to_do
diliver the key
Client_Take_The_Comper
the Client want more time
Extend_Rental
Client_Return_The_Comper
how many mileage and full tank
cheack_comper
if it is not full tank
first 1000km free and above it is 10 cents per km and
fuel_fee
mile_fee
send_payment
if the Client not pay this then suit them

Dao

The basic idea about ado patten is separate the database access and the control method for guarantee the data security.

So following the concept, we make four part of syntax to make sure about it.

1. Database access part :
    1. Connection class : to make the connection between JVM and Database(SQL Server) used by JDBC.
    2. CRUD method to access data.
2. Dao Interface :

    To make the comment interface to implement the comment database object, to ensure the database control method.

    Separate the database access and data operation.
3. Object Dao interface :

    Specific Object Dao interface extend the comment Dao interface separate the different database/table operation.
4. Object Dao implement : implement the day interface . Override the specific method. Access the specific data.

1. Database access part

Connection Class

```
1  public class ConnectionUtil {
2      public static Connection getConnection() throws Exception {
3          InputStream is =
   ClassLoader.getSystemClassLoader().getResourceAsStream("jdbc.propertie
   s");
4          Properties pros = new Properties();
5          pros.load(is);
6
7          String user = pros.getProperty("user");
8          String password = pros.getProperty("password");
9          String url = pros.getProperty("url");
10         String driverClass = pros.getProperty("driverClass");
11
12         Class.forName(driverClass);
```

```
13
14          Connection connection = DriverManager.getConnection(url, user,
    password);
15          System.out.println("The connection is successful : \n" +
    connection);
16          return connection;
17
18      }
19
20      public static void closeConnection(Connection connection) {
21          try {
22              if (connection != null) {
23                  connection.close();
24                  System.out.println("the connection has been closed !
    ");
25              }
26          } catch (SQLException throwables) {
27              throwables.printStackTrace();
28          }
29      }
30
31      public static void closeConAndPS(Connection connection, Statement
    preparedStatement) {
32          try {
33              if (preparedStatement != null) {
34                  preparedStatement.close();
35              }
36          } catch (SQLException throwables) {
37              throwables.printStackTrace();
38          }
39          try {
40              if (connection != null) {
41                  connection.close();
42              }
43          } catch (SQLException throwables) {
44              throwables.printStackTrace();
45          }
46      }
47
48      public static void closeConPSAndRS(Connection connection,
    Statement preparedStatement, ResultSet resultSet) {
49          try {
50              if (preparedStatement != null) {
51                  preparedStatement.close();
52              }
```

```
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        try {
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        try {
            if (resultSet != null) {
                resultSet.close();
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
```

Test method

```
@Test
    public void testGetconnection() throws Exception {
        Connection connection = ConnectionUtil.getConnection();
        ConnectionUtil.closeConnection(connection);

    }


```

CRUD Class

```
public class CRUD {

    public static void update(String sql, Object... args) {

        Connection connection = null;
        PreparedStatement preparedStatement = null;

        try {
            connection = ConnectionUtil.getConnection();
            preparedStatement = connection.prepareStatement(sql);
```

```
            for (int i = 0; i < args.length; i++) {
                preparedStatement.setObject(i + 1, args[i]);
            }
            preparedStatement.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            ConnectionUtil.closeConAndPS(connection,
preparedStatement);
        }
    }

    public static void getInstance(String sql, Object args) {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = ConnectionUtil.getConnection();
            preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setObject(1, args);

            resultSet = preparedStatement.executeQuery();

            ResultSetMetaData metaData = resultSet.getMetaData();
            int columnCount = metaData.getColumnCount();

            for (int i = 0; i < columnCount; i++) {
                System.out.print(metaData.getColumnName(i + 1) +
"\t");
            }
            System.out.println();

            if (resultSet.next()) {
                for (int i = 0; i < columnCount; i++) {
                    System.out.print(resultSet.getObject(i + 1) + "
|    ");
                }
                System.out.println();
            }


        } catch (Exception e) {
            e.printStackTrace();
        } finally {
```

```
52              ConnectionUtil.closeConPSAndRS(connection,
preparedStatement, resultSet);
53          }
54      }
55
56      public static void getTable(String sql) {
57          Connection connection = null;
58          PreparedStatement preparedStatement = null;
59          ResultSet resultSet = null;
60
61          try {
62              connection = ConnectionUtil.getConnection();
63              preparedStatement = connection.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
64              resultSet = preparedStatement.executeQuery();
65              ResultSetMetaData metaData = resultSet.getMetaData();
66
67              int columnCount = metaData.getColumnCount();
68              for (int i = 0; i < columnCount; i++) {
69                  System.out.print(metaData.getColumnName(i + 1) +
"\t");
70              }
71              System.out.println();
72
73              while (resultSet.next()) {
74                  for (int i = 0; i < columnCount; i++) {
75                      System.out.print(resultSet.getString(i + 1) + "
    |       ");
76                  }
77                  System.out.println();
78              }
79
80              resultSet.beforeFirst();
81          } catch (Exception e) {
82              e.printStackTrace();
83          } finally {
84              ConnectionUtil.closeConPSAndRS(connection,
preparedStatement, resultSet);
85          }
86
87      }
```

Test method

```
@Test
    public void testUpdate() {
        try {
            String sql = "update tbl_Bank set fld_BankName = ? where fld_BankID = ? ";
            update(sql, "DanskBank", 1);
            System.out.println("update finished");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Test
    public void testDelete() {
        try {
            String sql = " delete from tbl_Bank where fld_BankID = ? ";
            update(sql, 1);
            System.out.println("delete finished");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Test
    public void testGetInstance() {
        String sql = "select fld_BankID, fld_BankName from tbl_Bank where fld_BankID = ?";
        getInstance(sql, 1);
    }

    @Test
    public void testGetTable() {
        String sql = "select * from tbl_Bank";
        getTable(sql);
    }
```

2. The Base Dao Interface

```java
public interface Dao<T> {
    public void add(T t);

    public void update(T t);

    public void delete(int id);

    public void getInstance(int id);

    public void getAll();
}
```

3. The Client Dao interface

```java
public interface ClientDao extends Dao<Client> {
}
```

4. The Client Dao implement

```java
public class ClientDaoImpl implements ClientDao {

    @Test
    public void testAdd() {
        Client client = new Client();
        client.setClientID(3);
        client.setName("Fei Gu3");
        client.setAddress("Odense3");
        client.setEmail("feix0033@easv365.dk2");
        client.setPhoneNo(12345678);

        add(client);
    }
```

```
14
15      @Test
16      public void testUpdate() {
17          Client client = new Client();
18          client.setClientID(001);
19          client.setName("Fei Gu update ");
20          client.setAddress("Odense update");
21          client.setEmail("feix0033@easv365.dk update");
22          client.setPhoneNo(87654321);
23
24          update(client);
25
26      }
27
28      @Test
29      public void testDelete() {
30          delete(1);
31
32      }
33
34      @Test
35      public void testGetInstance() {
36          getInstance(1);
37      }
38
39      @Test
40      public void testGetAll() {
41          getAll();
42      }
43
44
45      @Override
46      public void add(Client client) {
47
48          try {
49              String sql = "insert into tbl_Client (fld_ClientID,
    fld_Name, fld_Address, fld_EmailAddress, fld_PhoneNo) values
    (?,?,?,?,?)";
50
51              int clientID = client.getClientID();
52              String name = client.getName();
53              String address = client.getAddress();
54              String email = client.getEmail();
55              int phoneNo = client.getPhoneNo();
56
```

```
            CRUD.update(sql, clientID, name, address, email,
phoneNo);
            System.out.println("Add database table Client
successful!");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void update(Client client) {
        try {
            String sql = "update tbl_Client set fld_Name = ? ,
fld_Address = ?, fld_EmailAddress = ? , fld_PhoneNo = ? where
fld_ClientID = ?";

            int clientID = client.getClientID();
            String name = client.getName();
            String address = client.getAddress();
            String email = client.getEmail();
            int phoneNo = client.getPhoneNo();

            CRUD.update(sql, name, address, email, phoneNo,
clientID);
            System.out.println("Update database table Client is
successful! ");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void delete(int id) {
        try {
            String sql = "delete from tbl_Client where fld_ClientID =
?";
            CRUD.update(sql, id);
            System.out.println("Delete database table Client is
successful!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
94
95      @Override
96      public void getInstance(int id) {
97          String sql = "select fld_ClientID, fld_Name, fld_Address,
        fld_EmailAddress, fld_PhoneNo from tbl_Client where fld_ClientID =
        ?";
98          CRUD.getInstance(sql, id);
99      }
100
101     @Override
102     public void getAll() {
103         String sql = "select * from tbl_Client";
104         CRUD.getTable(sql);
105     }
106
107 }
108
```

Singleton

The singleton patten make sure each time when we need the Object and its method. We only make “One” object.

Right here we use the patten for make sure when we get a new value from the UI input. We can just change the same one object but not create a new object.

```
public class RegisterClient {
    private static Client newClient = new Client();

    private RegisterClient() {
    }


    public static Client getNewClient() {
        return newClient;
    }

    public static void setNewClientID() {
        /* use hashCode() method to set the ClientID :
         * this is a simple and unique way to identify the customer
         * but there is a not easy way to find the specific user so
         * I should make a searching method to find the ClientID.
         */
        newClient.setClientID(newClient.hashCode());
    }

}
```