The auto camper rental software

Computer Science- 2nd Semester training project

Hand in: May 7th, 2021

Supervisors: Karsten Skov, Tommy Haugaard, Frank Østergaard Hansen

Authors: Andrej Simionenko, Raheela Tasneem, Fei Gu

Contents,

Table of Figures

1 Inception Phase

1.1 Start-up phase

The start-up phase begins with preparations of working environments for project and team. For the period of the project, it was early on decided to use zoom- break out rooms as well as discord server in order to hold daily meetings and communicate. There, the progress of the project as well as the next steps were discussed and shared.
For collaboration between the team members, a Github repository was used in order to share and work together on java code and database, while draw.io program was used in order to create the diagrams. Additionally, for collection of information and report writing online google word document was utilized.

1.2 Vision

The team is designing and creating an administration and booking system for Wagner Families auto camper rental business. The system is created for rental of a fleet of auto campers, and it should be time-efficient and easily manageable for the Wagner family, modernizing the way they do business.

1.3 Domain Model

The domain model is a structured visual representation of interconnected concepts or real-world objects that incorporates vocabulary, key concepts, behavior and relationships of all its entities. In figure 1, the domain model of auto camper rental software is shown and is used in order to showcase the purpose of the software.

During the process of building a custom-made system, Domain model comes in especially useful in few different aspects. From the perspective of developers, the domain model helps to clarify and understand the real-world problems, that the customer is facing and offers an opportunity to discuss the requirements for the system. On the other hand, from the perspective of the customer, the domain model allows a more user friendly and not very technical picture of the upcoming system, as one of the draws of domain model is its visual simplicity. It must be stated that the conceptual classes in the domain model are not software objects, but more so representations of real-world conceptual classes.

For this project, a domain model using a UML class diagram notation was used. Straight lines connecting different objects showcase association. Thus client can make a reservation of a camper, which then is contained in the booking system, which creates a payment for the client. The client has attributes such as name, address, driver license id, which are considered very important for program.

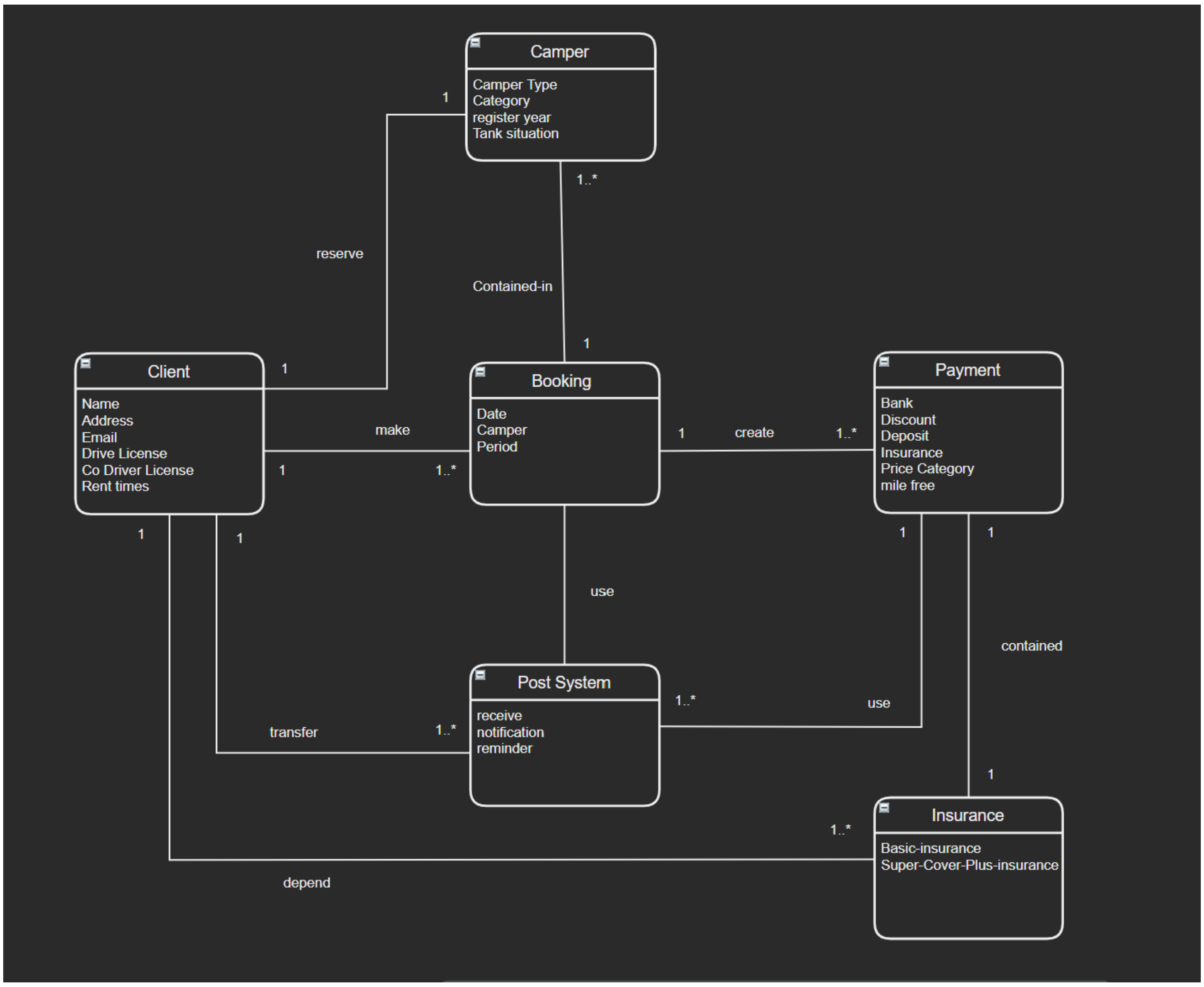


Figure 1 Domain Model

1.4 Data dictionary

The data dictionary is an agreement between the customers and developers over the concrete meaning of specific words used in the system. The dictionary is used to gain consistency and avoid confusion when specific words are used. Figure 2 shows a data dictionary for the auto camper system, where specific words are explained in further detail and their meaning in the system is set.

Name	Description	Alias	Format
Client	Customer who is looking to rent camper	**Citizen/ client**	
Administrator	Persons who manage & works at Camper renting company		
Bank	Bank is responsible to hold an account as to receive payment from the client		
Deposit	Client must pay 10% payment no later than 2 weeks after reservation. If the rental is cancelled for some reason – this deposit is lost.		

Discount	client who come back year after year accordingly how many times they rent again, they get discount 5% or more		

Figure 2 Data Dictionary

2 Elaboration Phase

2.1 Use Cases

Use cases are written descriptions or event steps defining and documenting the interactions between actors and the system to achieve a goal. They are commonly used to represent a goal of a system and describe, clarify a single task of the process. Use cases can contain one or multiple scenarios, describing the order of events.
The actors in the use cases can be humans, other users of the system, or the system itself. Description is used to explain a certain scenario, sequence of actions and interactions between the actors and the system.
For the project two main types of use cases will be used: brief use cases and fully dressed use cases. The different types of use cases are used depending on how much detail is needed for certain scenarios.

2.1.1 Brief use cases

Brief use cases are used to see a short summary of all the requirements for the system. These use cases contain a short, one line description of the scenario between the actor and the system.
Moreover, brief use case table is used to prioritize functionality of the program. The most important functions being at the top of the table, while not so important functionality near the bottom. This allows developers to focus on key functionality when creating the program. Figure 3 showcases brief use case table for the project. The use cases in the table are identified by a use case number in the first column, followed by the actor of specific use case in the second column, then in a third column use case description, the version in fourth column and finally a time estimate in the fifth. The use case number indicates the priority of the use case problem, thus the lower the use case number the more crucial it is. The actor column showcases who is the actor of the specific use case, while the description column

gives a short summary. All this information is especially important for developers and the customers who are ordering a custom-made system. For developers it gives an estimate workload per task and allows an estimation of total time for the project. The total time estimate allows developers to calculate the approximate price for the project and inform customers.

Use case number	Actor	Use case description
UC1	Admin	Create camper category
UC2	Admin	Create Contact
UC3	Client	Register user
UC4	Admin	Create Price Category according to facilities
UC5	Admin	Create Price Category in High & low seasons
UC6	client	Contact Camper Owner
UC 7	System	Create Confirmation email/invoice
UC8	Client	Confirm the email
UC 9	System	Create Rental payment
UC 10	System	Create Deposit

UC 11	System	Create Reservation
UC 12	System	Create Reminder
UC 13	System	Create insurance
UC 14	System	Create Discount
UC 15	Client	See Guidelines
UC 16	Client	Delete Review
UC 17	Admin	Delete Review
UC18	Client	Edit Review
UC19	Client	Give rating on Other Users(stars)
UC 20	Client	Accept terms of service

Figure 3 Use Case Table

2.1.2 Fully dressed use cases

The fully dressed use cases allow to dwell deeper into the use case and inspect it completely. The use cases include all the details and steps needed for the action described to succeed. Two use cases were chosen to dwell deeper and fully explain the process that a new customer must go through in order to register in the system when completing an order. These use cases are use case 3 (Figure 4: Fully Dressed Use Case 3) and use case 8 (Figure 5: Fully Dressed Use Case 8), which can be seen bellow.
To start off the registration process a new client must press the registration button when completing an order, making it the trigger event. The precondition is client selecting all the things that need to be booked. When the button is pressed, a new page is opened, where the user is required to enter a username and password, which are checked if they are valid and available. Afterwards, the clients' full details must be entered, such as the name of the client, address, phone number, email and valid drivers license number. Once all the steps are done the system will send an email that must be confirmed by the client, making it the post condition. This is where the use case 8 – confirm email starts, making its precondition completing the registration form. In order to complete the registration, the client must open the received email, and press the confirmation button. Once the button is pressed, client gets informed that the registration is complete and the new account can be used, thus the post condition for this use case is completion of registration.

Use Case #3	Register user
Actors	Client
Trigger	Search for registration
Precondition	The client has already searched for the camper and is ready to register
Postcondition	Confirmation of registration email is sent to the user
Normal flow	• The client enters username and password. • The system will check if the username and password are valid and available. • The client enters his info: name, address, phone No., email, driving license ID/No. • The system will send confirmation email.
Version	1.0
Date	5-5-2021
Author	Group: Fei, Raheela, Andrej

Figure 4 Fully Dressed Use Case 3

Use Case #8	**Confirm Email**
Actors	Client
Trigger	Search for confirmation to register
Precondition	Complete the registration form
Postcondition	Client’s account is created
Normal flow	· The client opens the received email. · The client presses the confirmation button. · The client gets the message that registration is complete.
Version	1.0
Date	5-5-2021
Author	Group: Fei, Raheela, Andrej

Figure 5 Fully Dressed Use Case 8

2.2 System Sequence Diagrams (SSD's)

System sequence diagrams are very good way to showcase visually how a certain use case works. It shows all the events that external actors generate, their order and possible inner system events for a particular use case. Two fully dressed use cases were chosen to visually showcase, use case 3 and use case 8.
In use case 3, the client presses the registration button, and the system opens the registration page. There the client enters username and password, and all of their private details. The system checks if the entered username and password are valid and available, and if so, sends a confirmation email to the client.

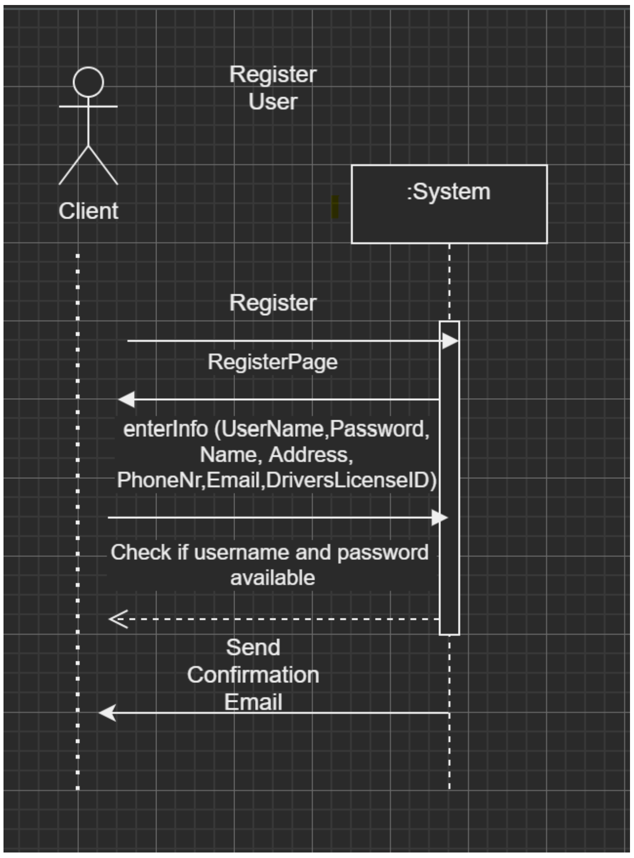

Register User
Client
:System
Register
RegisterPage
enterInfo (UserName,Password, Name, Address, PhoneNr,Email,DriversLicenseID)
Check if username and password available
Send Confirmation Email

Figure 6 Use Case 3 SSD

In use case 8, the client receives a mail to the email that was used to register. The email contains a button that must be pressed to complete the registration. After the successful press of a button by client, a message saying that registration was successful appears, and the system fully creates clients' account.

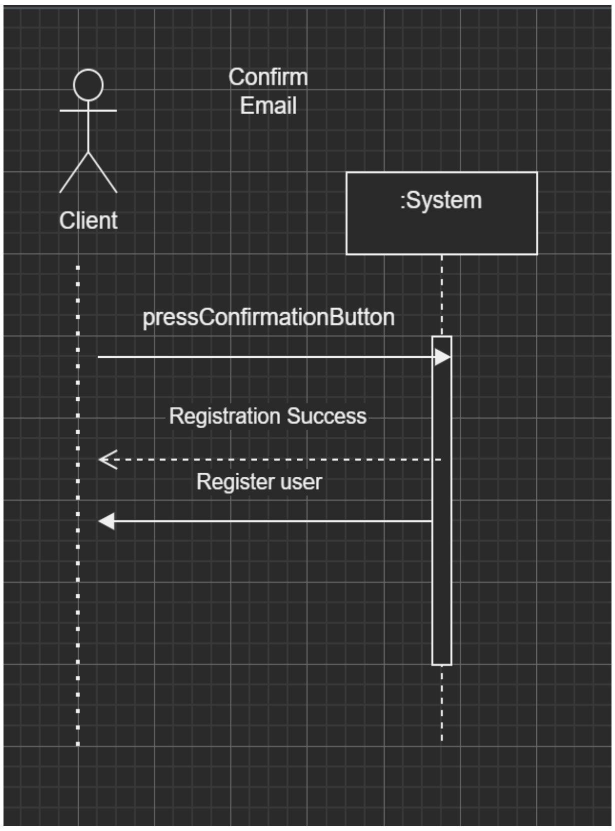


Figure 7 Use Case 8 SSD

2.3 ER Diagram

The Entity relation diagram is used to display the relationship of entity sets stored in database, in simpler terms the diagram helps to explain the logical structure of the database. The principle of relation database design is that the organization is based on logical relations between the data tables, the basic principle of relation databases is to make them simple and avoid duplication. By doing so the system has to handle a lot less data and the working speed is increased. For this reason, the databases are dynamic and have a CRUD functionality, else the databases would run a risk of being outdated and broken due to broken relationships and data trash.

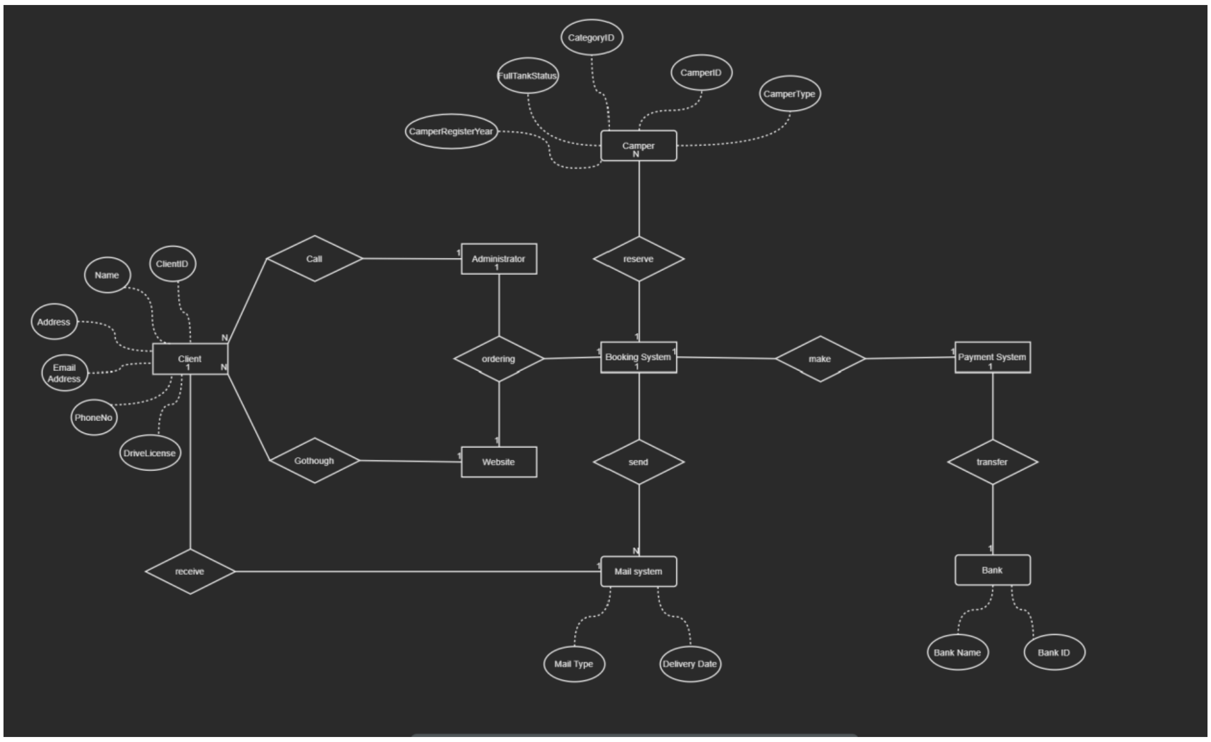


Figure 8 ER Diagram

2.4 UML Diagram

2.4.1 UI Class Diagram

This is the part of User Interface class diagram, it shows the main logic and structure behind the design of UI class and the connection with the controller class. It showcases the classes, their attributes, methods, and associations. Following the relationship we can also identify about how the method invokes the next level page to reach different purpose. At the end of the relation chain the UI controller will require the database JDBC to read and write the data.

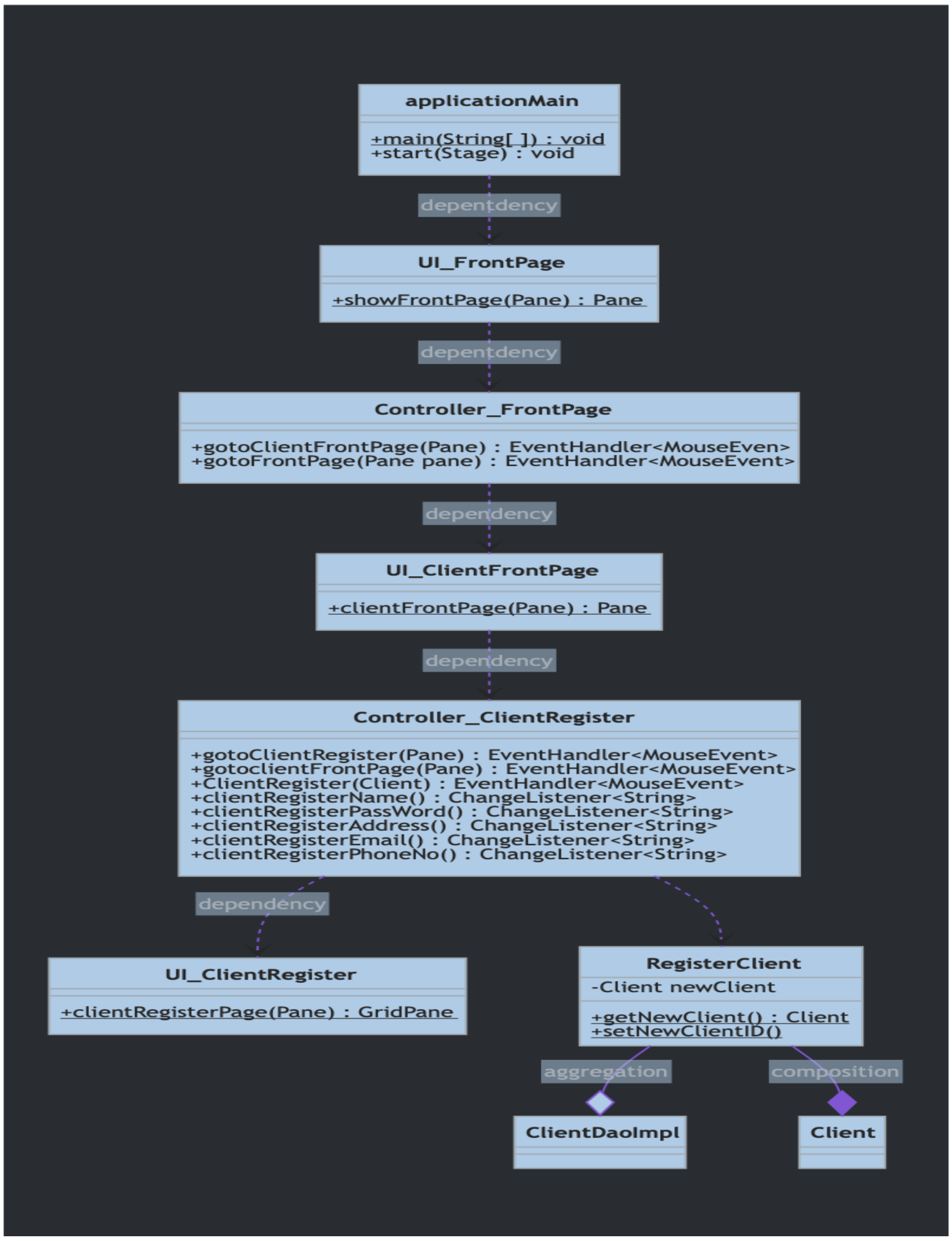

applicationMain
+main(String[ ]) : void
+start(Stage) : void
depentdency
UI_FrontPage
+showFrontPage(Pane) : Pane
depentdency
Controller_FrontPage
+gotoClientFrontPage(Pane) : EventHandler<MouseEven>
+gotoFrontPage(Pane pane) : EventHandler<MouseEvent>
dependency
UI_ClientFrontPage
+clientFrontPage(Pane) : Pane
dependency
Controller_ClientRegister
+gotoClientRegister(Pane) : EventHandler<MouseEvent>
+gotoclientFrontPage(Pane) : EventHandler<MouseEvent>
+ClientRegister(Client) : EventHandler<MouseEvent>
+clientRegisterName() : ChangeListener<String>
+clientRegisterPassWord() : ChangeListener<String>
+clientRegisterAddress() : ChangeListener<String>
+clientRegisterEmail() : ChangeListener<String>
+clientRegisterPhoneNo() : ChangeListener<String>
dependency
UI_ClientRegister
+clientRegisterPage(Pane) : GridPane
RegisterClient
-Client newClient
+getNewClient() : Client
+setNewClientID()
aggregation
composition
ClientDaoImpl
Client

Figure 9 User Interface Class Diagram

2.4.2 Database-User

The database class diagram shows the structure of the database. The connectionUtil includes the connection to database and a method to close the connection with database. The CRUD class is containing the create update delete and select methods, which are key to keeping database functional and clean. Dao implement class use DAO patten to separate the query and the prepared statements to make the data safe.

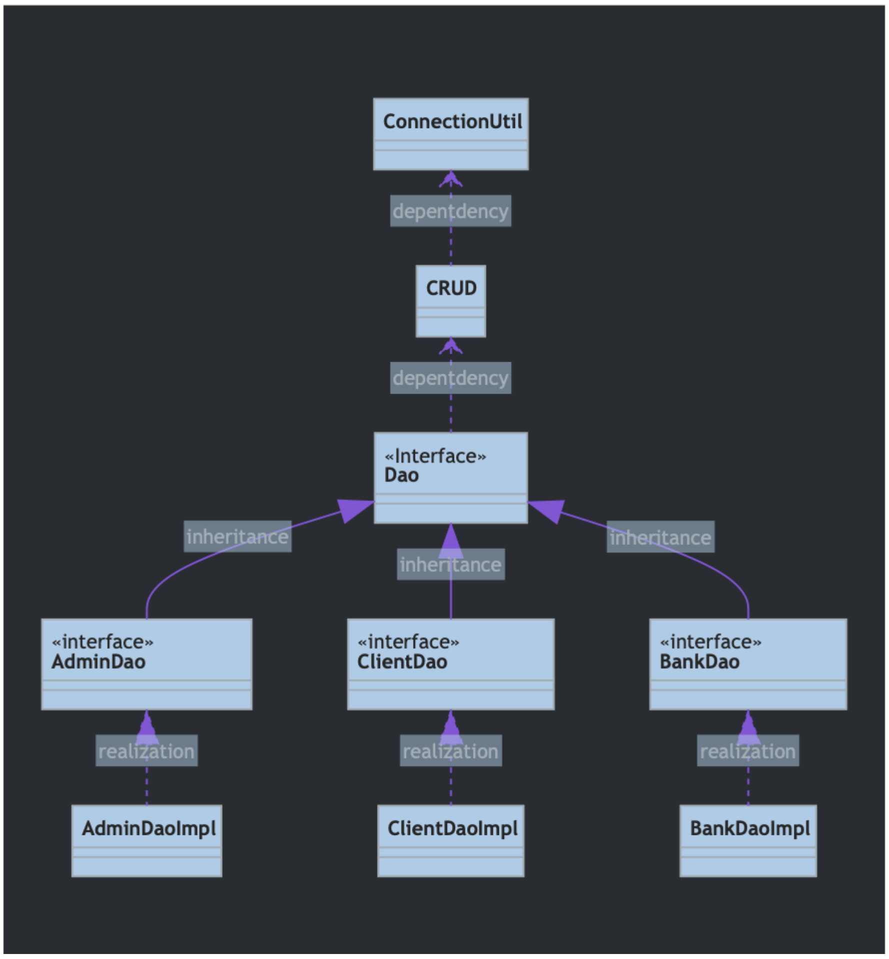


Figure 10 Database-User class diagram

2.4.3 Domain

The domain part is finishing on the UML design diagram. This class diagram shows the part of relationship between different packages and classes. The remaining of package class diagrams can be found bellow.

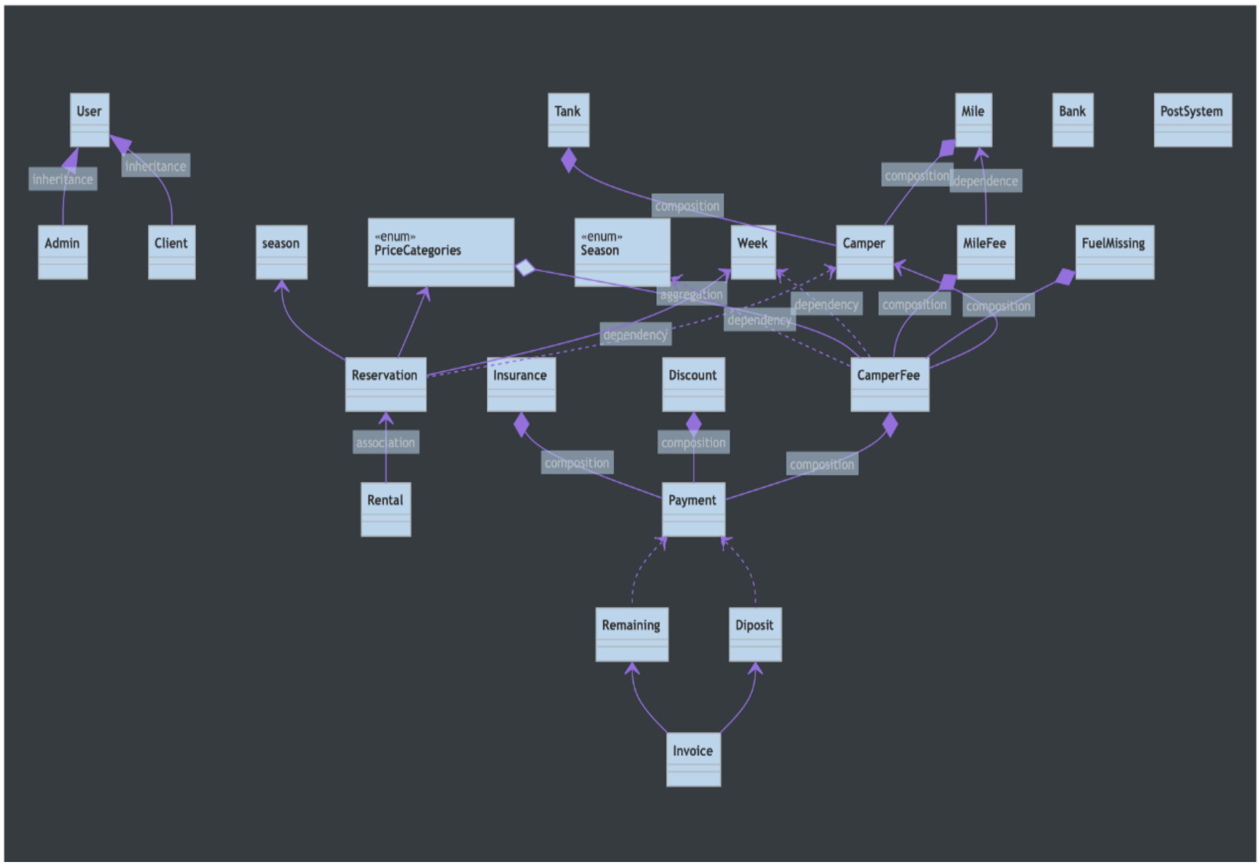


Figure 11 Domain Class diagram

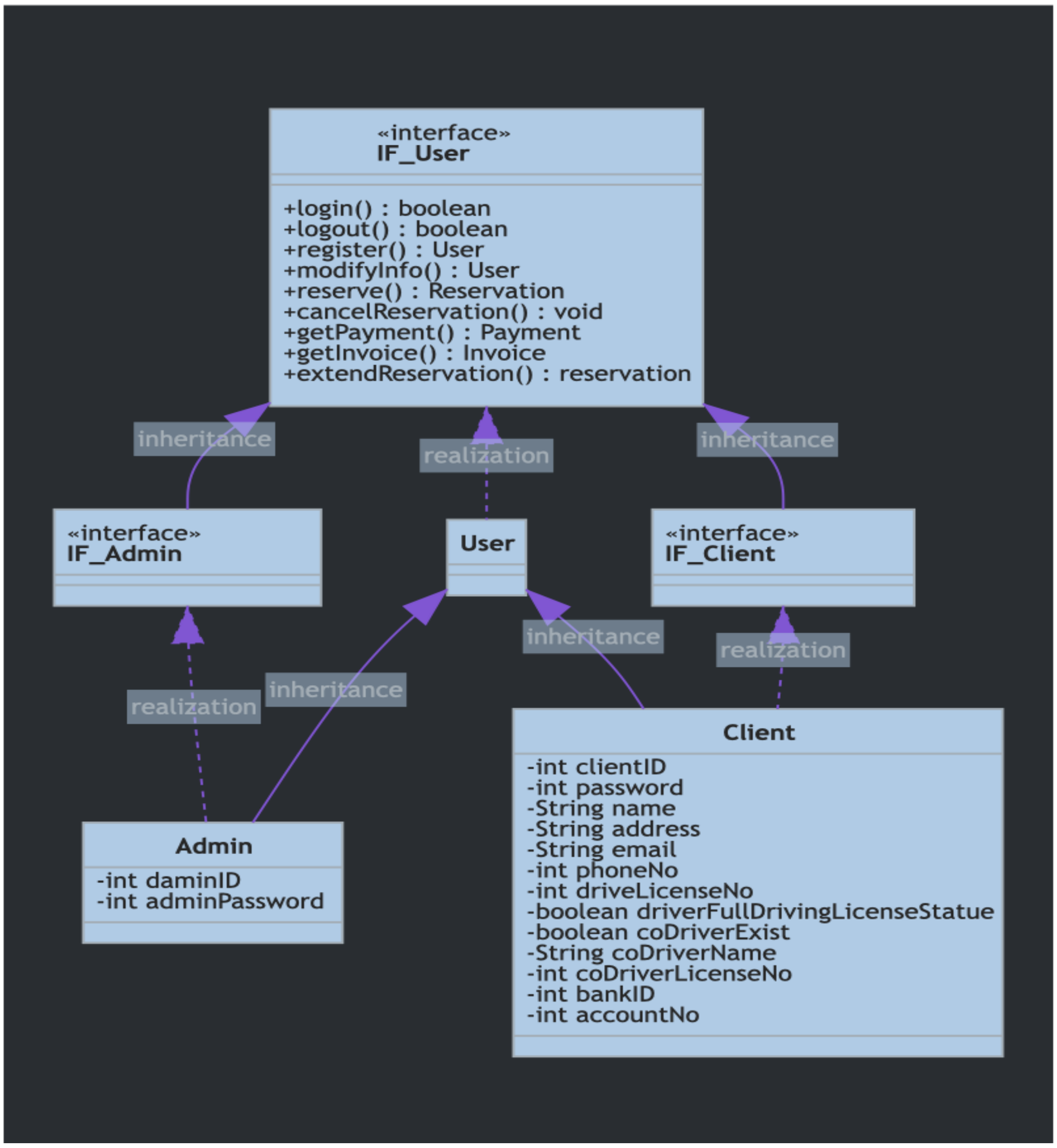


Figure 12 Domain User Class Diagram

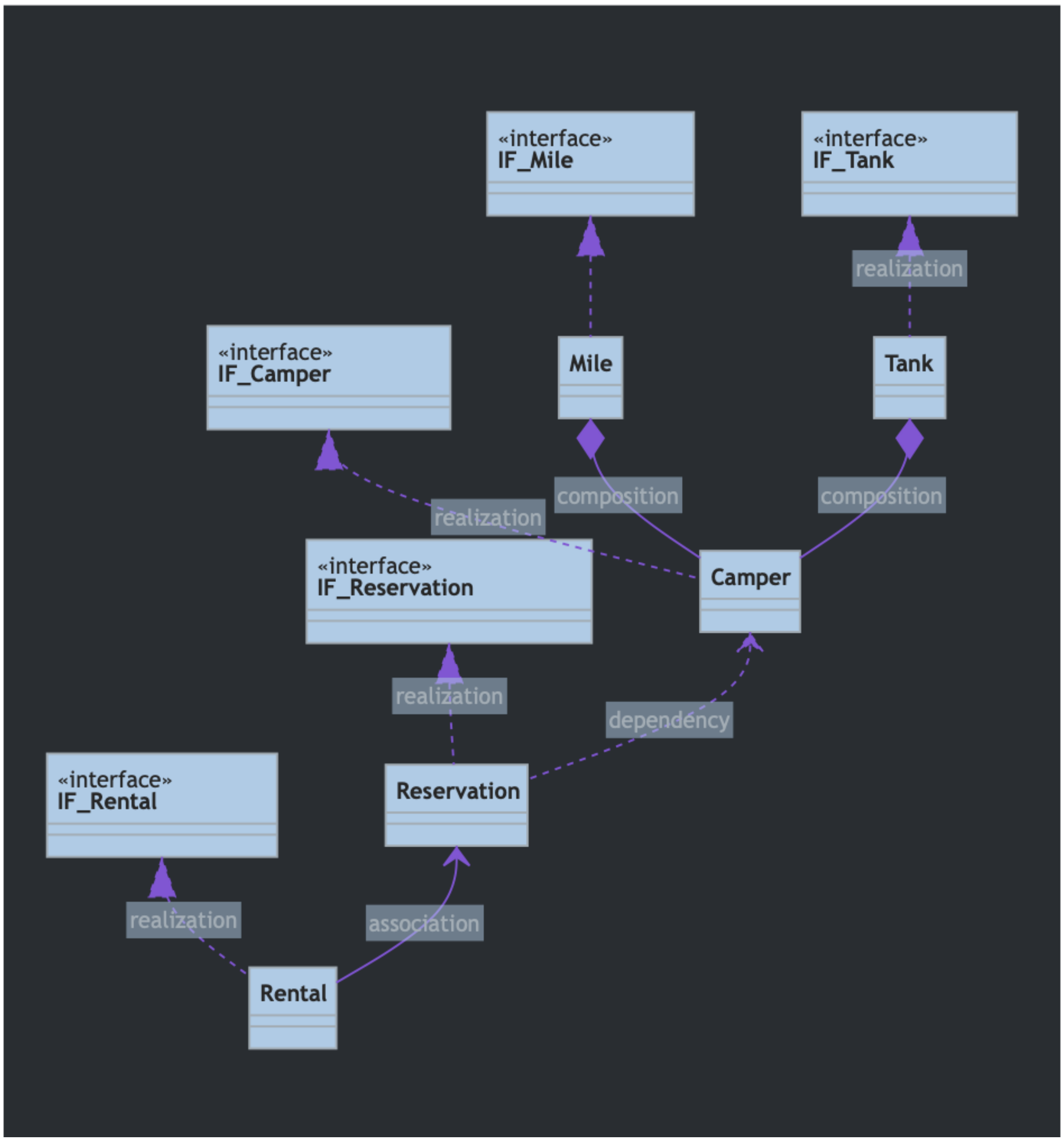


Figure 13 Domain Camper Class Diagram

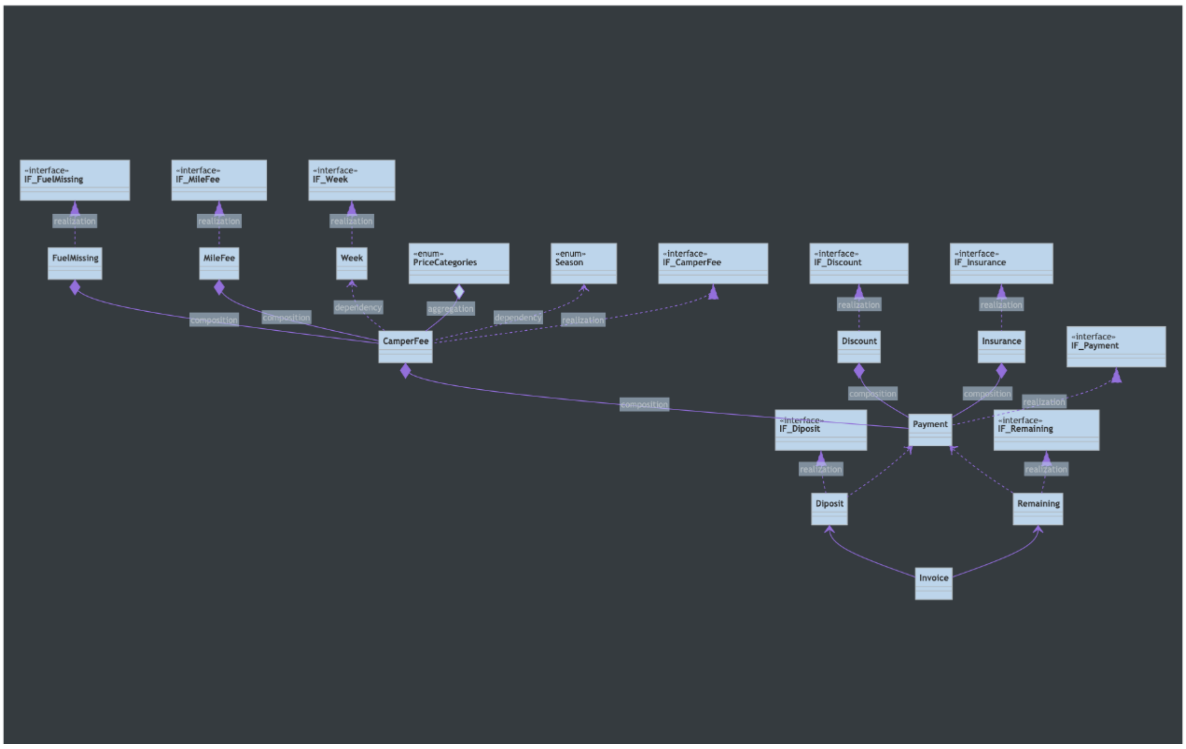


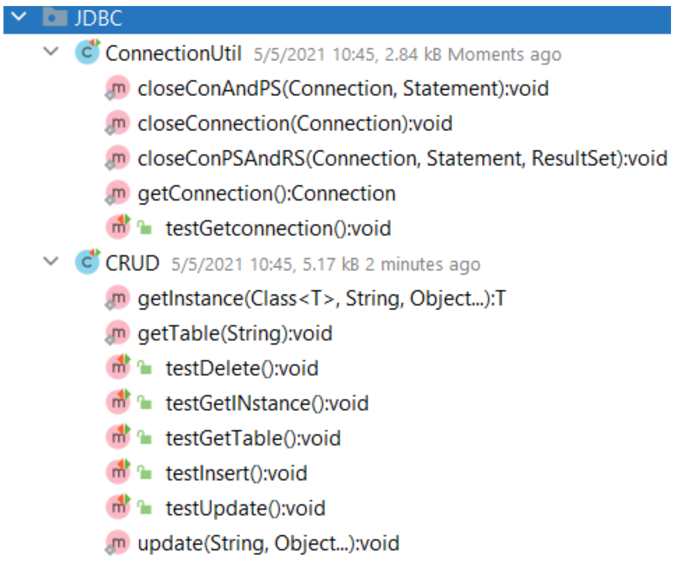
Figure 14 Domain Payment Class Diagram

3 Construction phase

3.1 Development and implementation

3.1.1 JDBC

JDBC (Java Database Connectivity) is the Java API that manages connecting to a database, issuing queries and commands, and handling result sets obtained from the database.



```
JDBC
  ConnectionUtil 5/5/2021 10:45, 2.84 kB Moments ago
    closeConAndPS(Connection, Statement):void
    closeConnection(Connection):void
    closeConPSAndRS(Connection, Statement, ResultSet):void
    getConnection():Connection
    testGetconnection():void
  CRUD 5/5/2021 10:45, 5.17 kB 2 minutes ago
    getInstance(Class<T>, String, Object...):T
    getTable(String):void
    testDelete():void
    testGetINstance():void
    testGetTable():void
    testInsert():void
    testUpdate():void
    update(String, Object...):void
```

We started first to have connection with Database

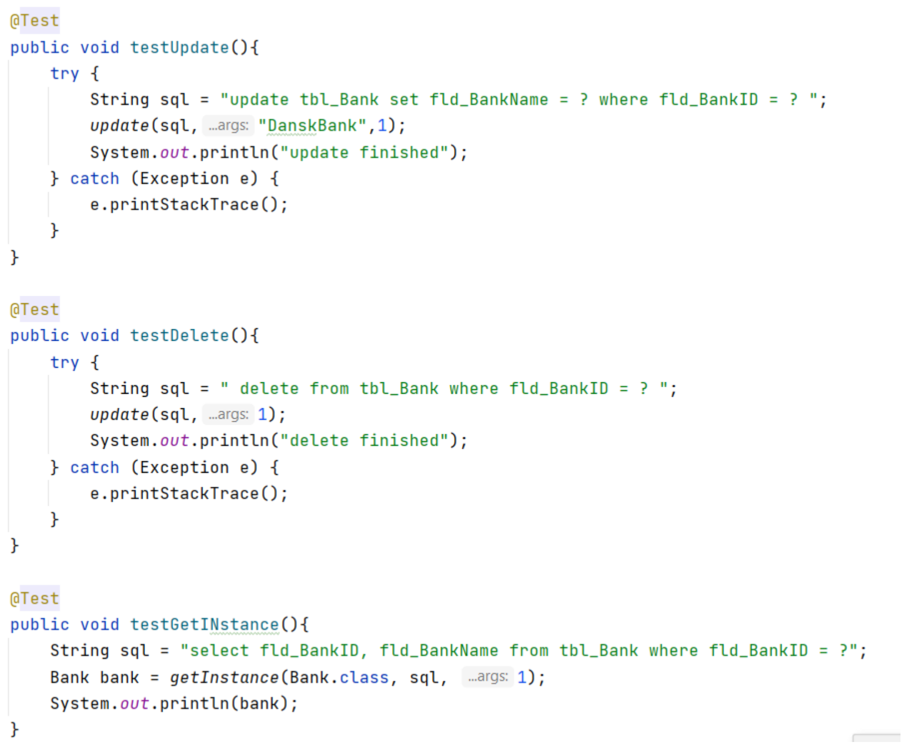
```java
public class ConnectionUtil {
    @Test
    public void testGetconnection() throws Exception {
        Connection connection = ConnectionUtil.getConnection();
        ConnectionUtil.closeConnection(connection);

    }
```

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

    Connection connection = null;
    PreparedStatement preparedStatement = null;

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

CRUD is an acronym for CREATE, READ, UPDATE and DELETE which are basic functions of persistent storage. CRUD operations can use forms or an interface view to retrieve and return data from a database.

We include few examples like to test insert ,delete and update values in the table.

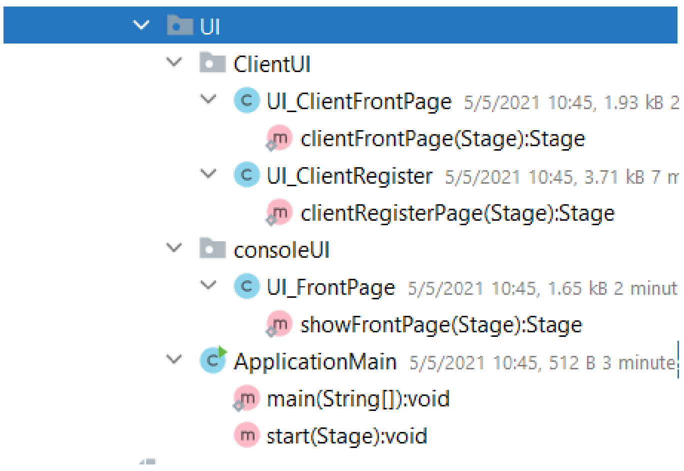
```
public class CRUD {

    @Test
    public void testInsert(){
        try {
            String sql = "INSERT INTO tbl_Bank(fld_BankID,fld_BankName) values (?,?) ";
            update(sql, ...args: 1,"Nordea");
            update(sql, ...args: 2,"DanskBank");
            update(sql, ...args: 3,"SydDenmarkBank");
            System.out.println("Insert finished");
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```



```
@Test
public void testUpdate(){
    try {
        String sql = "update tbl_Bank set fld_BankName = ? where fld_BankID = ? ";
        update(sql, ...args: "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, ...args: 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 = ?";
    Bank bank = getInstance(Bank.class, sql,  ...args: 1);
    System.out.println(bank);
}
```

3.1.2 User interface

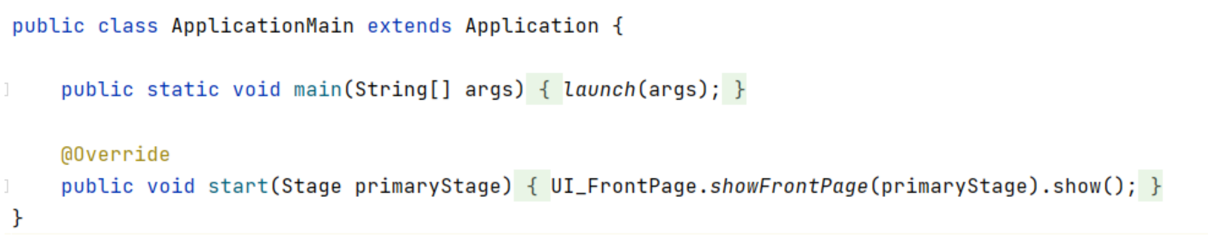
Graphical User Interface is to use visual experience builder for Java applications, through which the user can interact with an application.In our case, we have made a Client UI front page and register page.



```
UI
  ClientUI
    UI_ClientFrontPage  5/5/2021 10:45, 1.93 kB 2
      clientFrontPage(Stage):Stage
    UI_ClientRegister  5/5/2021 10:45, 3.71 kB 7 n
      clientRegisterPage(Stage):Stage
  consoleUI
    UI_FrontPage  5/5/2021 10:45, 1.65 kB 2 minut
      showFrontPage(Stage):Stage
  ApplicationMain  5/5/2021 10:45, 512 B 3 minute
    main(String[]):void
    start(Stage):void
```

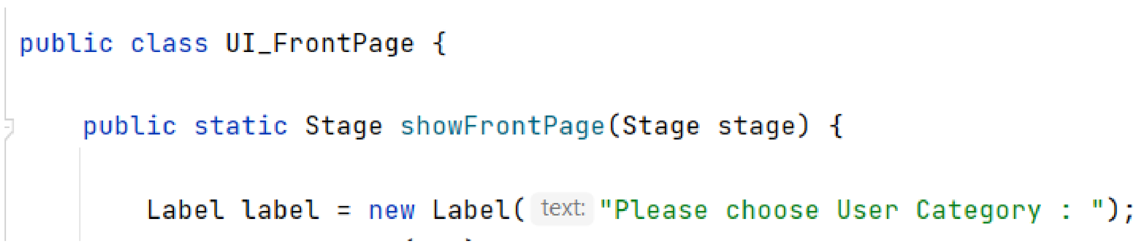
User interface main:

Main UI class will launch the App

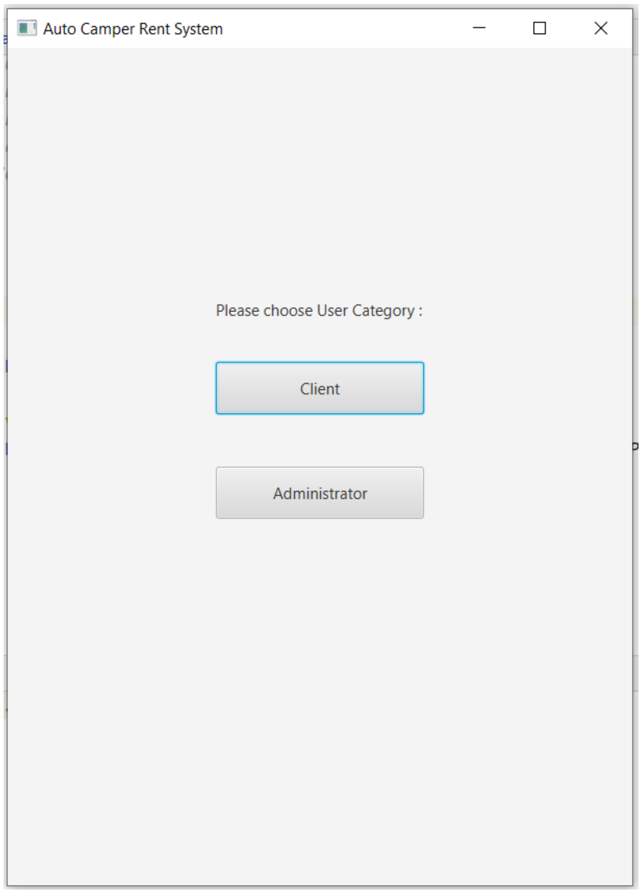


```
public class ApplicationMain extends Application {

    public static void main(String[] args) { launch(args); }

    @Override
    public void start(Stage primaryStage) { UI_FrontPage.showFrontPage(primaryStage).show(); }
}
```

UI front Page will let you see the first look of the app that gives you option to select log in as a client or as an administrator.



```
public class UI_FrontPage {

    public static Stage showFrontPage(Stage stage) {

        Label label = new Label( text: "Please choose User Category : ");
```

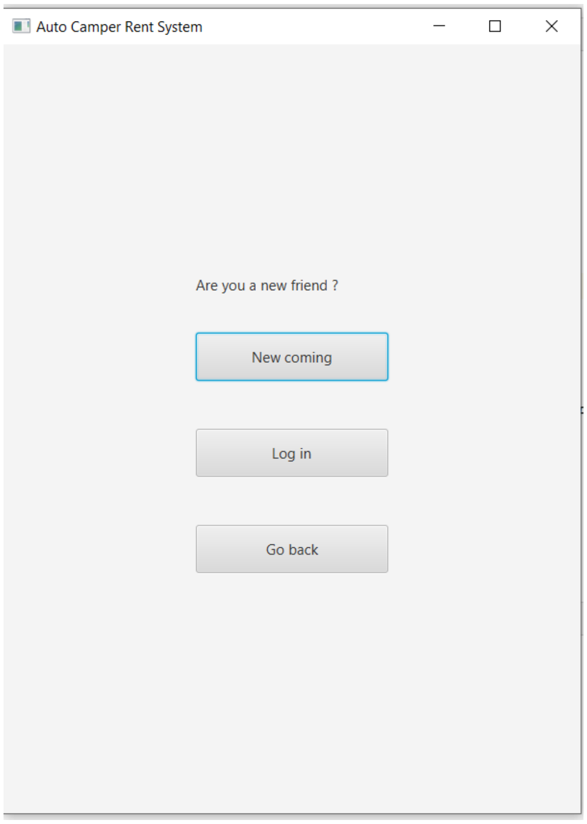


```
client.addEventHandler(MouseEvent.MOUSE_CLICKED, new Controller_FrontPage().gotoClientFrontPage(stage));
```

After selecting to log in as a client, another page opens and asks if the user is a new client or an existing user, which can proceed to log in

```
public class UI_ClientFrontPage {
    public static void clientFrontPage(Stage stage){
```

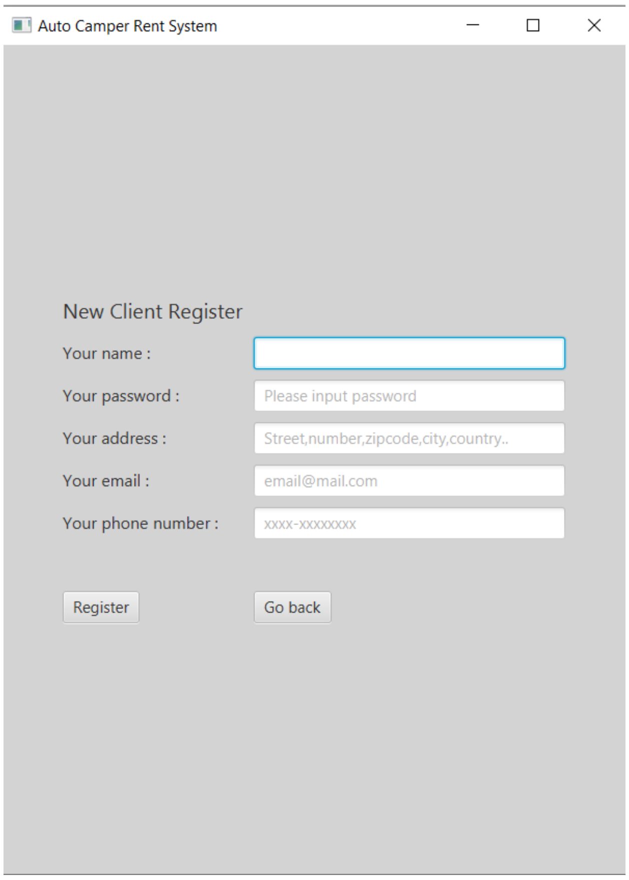
```
userRegister.addEventHandler(MouseEvent.MOUSE_CLICKED, new Controller_ClientRegister().clientRegister(stage));
```



The next step takes you to register as a new client if 'new coming' is selected and ask to fill out personal information.

```
public class UI_ClientRegister {

    public static Stage clientRegisterPage(Stage stage){

        Label l_registerPageTitle = new Label( text: "New Client Register");
        l_registerPageTitle.setStyle("-fx-font-size: 20");

        Label l_name = new Label( text: "Your name : ");
        l_name.setFont(Font.font(16));
```

```
public class Controller_ClientRegister {
    Client newRegisterClient;

    public EventHandler<MouseEvent> clientRegister(Stage stage){
        EventHandler<MouseEvent> mouseEventEventHandler = event -> UI_ClientRegister.clientRegisterPage(stage);
        newRegisterClient = new Client();

        return mouseEventEventHandler;
    }

    public ChangeListener<String> clientRegisterText(){
        ChangeListener<String> changeListener = (observable, oldValue, newValue) -> {
            System.out.println(newValue);
        };
        return changeListener;
    }
}
```


Auto Camper Rent System

New Client Register

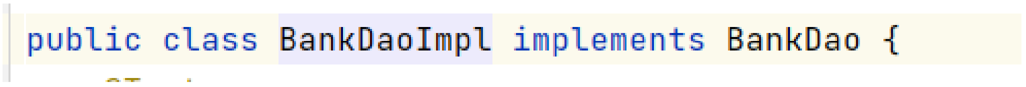
Your name :	
Your password :	Please input password
Your address :	Street,number,zipcode,city,country..
Your email :	email@mail.com
Your phone number :	xxxx-xxxxxxxx

3.1.3 DAO

The Data Access Object (DAO) pattern is a structural pattern. It gives the functionality to hide from the application all the complexities involved in performing CRUD operations in the underlying storage mechanism. This permits both layers to evolve separately without knowing anything about each other. It can be seen how it can keep the domain model completely decoupled from the persistence layer.

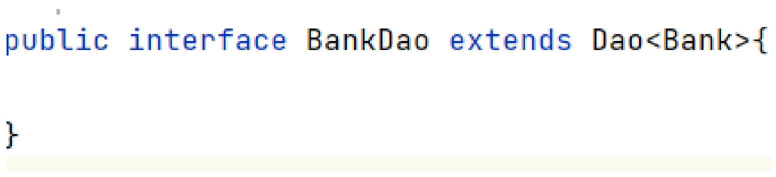
```
public interface Dao<T> {
    3 implementations
    public  void add(T t);
    3 implementations
    public  void update(T t);
    3 implementations
    public  void delete(int id);
    3 implementations
    public  T getInstance(int id);
    3 implementations
    public  void getAll();
}
```

We create a *Bank* object acting as Value Object.*Dao* is Data Access Object Interface.*BankDaoImpl* is concrete class



```
public class BankDaoImpl implements BankDao {
```

This class implementing Data Access Object Interface. *BankDao*, this class, will use *Dao.*



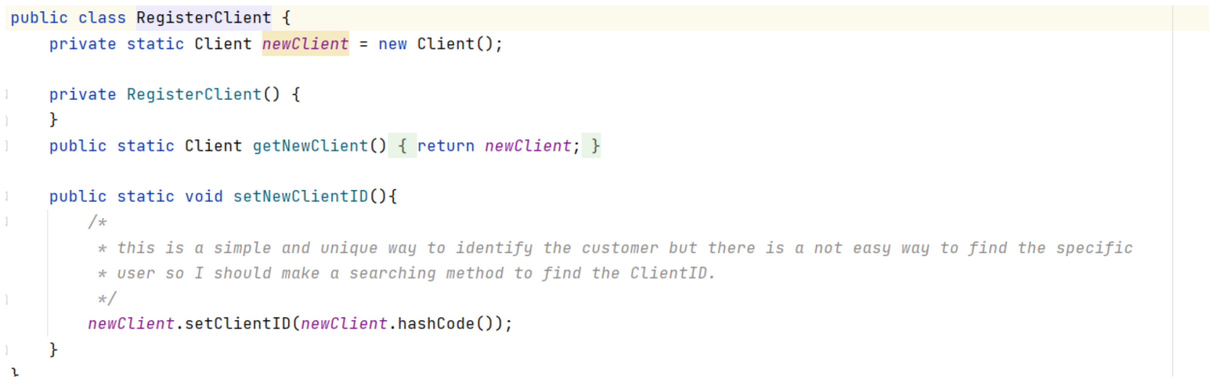
```
public interface BankDao extends Dao<Bank>{


}
```

3.1.4 Singleton

Singleton means One and only. In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time.
After first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created.

For singleton class we use getInstance() method which is static and returns instance. We may also use the class name as method name while defining this method. We instantiate inside class and it is one time use in JDBC.

.



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

    private RegisterClient() {
    }
    public static Client getNewClient() { return newClient; }

    public static void setNewClientID(){
        /*
         * 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());
    }
}
```

3.2 Github link

The Github link to repository that has been used throughout the project:
https://github.com/Fei-D20/The_Autocamper_rental_business