OOP Assignment 4

CAU Dealer

Team #8

유용민 **(Leader)**

설지환

안지완

이의제

이주형

채승운

최동욱

Contents

00 List of team members

Name	Id	Notes
유용민	20194094	Leader
설지환	20192958	
안지완	20190449	
이의제	20190197	Speaker
이주형	20214016	
채승운	20194435	
최동욱	20192971	

01 Introduction

Our team's project is developing a "Used car trading program: CauDealer". As the project 4 should be developing a software system that's related to OOP concepts, we chose the topic which is suitable to show the basic principle of OOP; data abstraction and encapsulation. In this report, we will explain about the brief concept, functionality, implementation of our project with UML modeling and how we applied object-oriented concepts

02 Concept of the project

As I mentioned at the introduction, our topic is "Used car trading program". Before implementation, we classified it to five situations.

1. **Login authentication**
2. **View user information as a customer and seller**
3. **Search the registered car for sale as a customer and seller**
4. **Sell the car (register on the system) as a seller**
5. **Buy the car as a customer**

To specify these situations, we made interacting objects like cars, users, and the android components. As there's many types of cars in the real world, we tried to get an abstraction of the car not just as a vehicle, with other attributes which matters on transaction. So, when sellers register their used car for sale in the system, they must fill in all the attributes of the car. Based on this, users can search used cars filtering by their attributes. Customer can choose what they want and sellers can get information of the overall market price. And this action causes change in the system.

And we chose Android OS as the UI of this program. Based on this basic information, we will expand it to Android environment.

Let's briefly take a look about the model class we made.

1) User

```java
public class User{
    private String userName;
    private boolean isSeller;

    protected ArrayList<Car> Cars;

    public User(String name, boolean isSeller);
    public boolean addCar(Car newCar);
    public boolean buyCar(Car targetCar, User seller);
    public static ArrayList<Car> getCarList();
    public boolean getIsSeller();
    public String getName();
    public void setCarList(ArrayList<Car> myList);
    public void setIsSeller(boolean isSeller);
    public void setName(String name);
    public void showCarList();
}
```

2) Car

```java
public class Car{

    private enum Fuel;
    private enum Type;
    private String name;
    private String manufacture;
    private String number;
    private String color;
    private String carImage;
    private int price;
    private int capacity;
    private int distanceDriven;
    private int year;
    private Fuel fuel;
    private Type type;
    private boolean isAccident;
    private boolean isTuned;
    private final ArrayList<Accident> accidents = new ArrayList<>();
    private final ArrayList<Tune> tunes = new ArrayList<>();

    public Car(String name, String manufacture, String number, String color,
String type, int price, int capacity, int distanceDriven, int year, String
fuel, boolean isAccident, boolean isTuned);
    public void printCarInfo();

    // getters
    // setters
}
```

3) Accident

```java
public class Accident{
   private String date;
   private String content;

   public Accident(String date, String content);
   public String getDate();
   public String getContent();
   public void setDate(String date);
   public void setContent(String content);
}
```

4) Tune

```java
public class Tune{
   private String date;
   private String content;

   public Tune(String date, String content);
   public String getDate();
   public String getContent();
   public void setDate(String date);
   public void setContent(String content);
}
```

All these classes are in same "Model" package.

- User
- Car
- Accident: Class containing accident histories and their information
- Tune: Class containing tune histories and their information

And there's android classes, which contains many informations. We'll explain it at 04 implementations.

03 Functionality

As I said, we divided our program's situations to five cases. I'll explain these three cases with the functionalities each situations require. Let's follow the process that customer buy the car.

1) Login authentication

In the first screen, we have to fill in ID and PW to the system to let them identify us. If we gave the information to the system, it checks from the DB, and go to Main screen. Then system has to load the information of the user, and it's cars.

2) View user information

Based on the 1) functionality, system contains user's information. At the main screen, if click the info button, it turns to user information page. And we can retrieve the data in a regular form.

3) Sell the car (register on the system) as a seller

In the first situation, there's nothing for sale on the system. In order for a customer to purchase a car, seller must register the car on the system. From the seller's point of view, to register it on the system, seller need interface that seller can fill in the information of it, and based on those data, system should store the information in its database.

4) Search the registered car for sale as a customer and seller

If few sellers upload their cars on the system, customers and other sellers now can search the updated system information. If there's no filtering system, every user would have to see the whole data, even though they don't want. So, we need interface that user can fill in the information which can be a single value or the ranged value. Based on these values, system has to load the data from the database so as the data to be screened on user's viewport. And if the user selects one, he could see the whole attributes the object has.

5) Buy the car as a customer

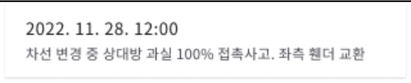
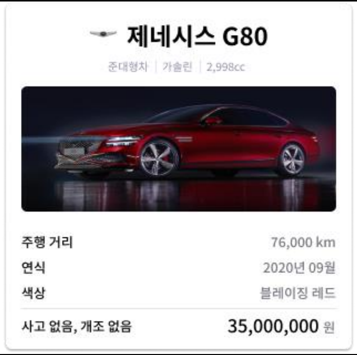
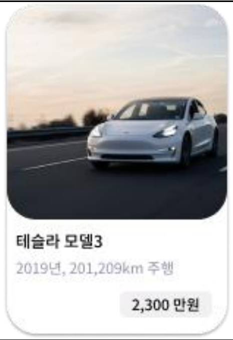
If the customer chooses one, he might buy it. If he presses the purchase button, that car must be subordinated to the customer and deleted on the seller's car list.

We divided these functionalities to each classes properly, and now let me introduce these functionalities one by one.

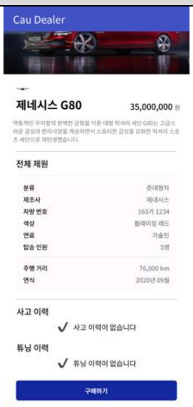
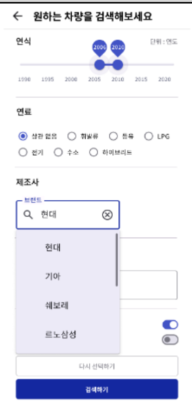
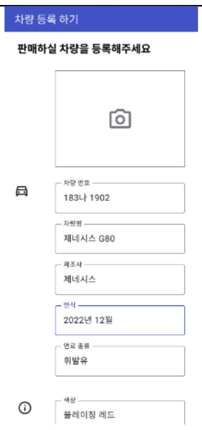
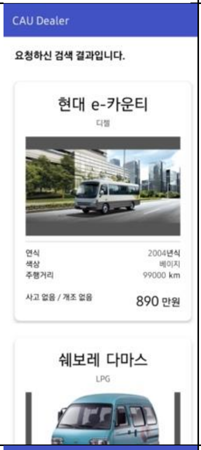
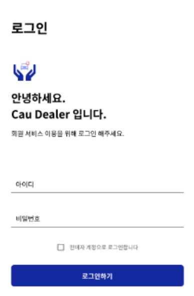
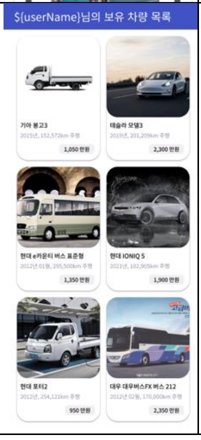
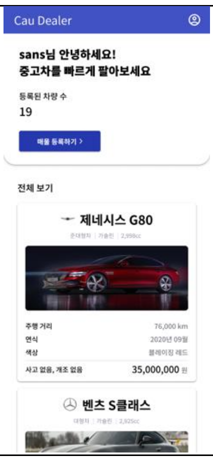
User class

Function	parameter	description
addCar	*Car* ***newCar***	From the seller's point of view, register the car(***newCar***) on the system for sale.
buyCar	*Car* ***targetCar*** *User* ***seller***	From the customer's point of view, buy the car(***targetCar***) from seller(**seller**).

Adapter class

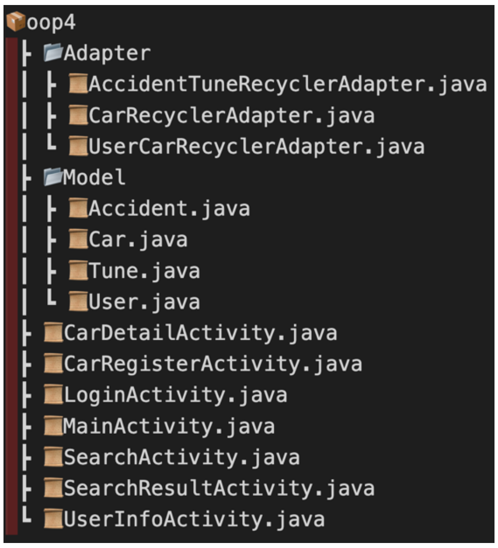
Adapter	figure	description
AccidentTuneRecyclerAdapter.java	 2022. 11. 28. 12:00 차선 변경 중 상대방 과실 100% 접촉사고. 좌측 휀더 교환	The component of display that shows the information of tuning or accident
CarRecyclerAdapter.java	 제네시스 G80 준대형차 | 가솔린 | 2,998cc 주행 거리 76,000 km 연식 2020년 09월 색상 블레이징 레드 사고 없음, 개조 없음 35,000,000 원	The component of display that shows middle-specific information of car from database
UserCarRecyclerAdapter.java	 테슬라 모델3 2019년, 201,209km 주행 2,300 만원	The component of display that shows low-specific information of car from User's car list(*ArrayList<cars> Cars*)

Activity classes

Activity	figure	description	Activity	figure	description
CarDetailActivity.java	 Cau Dealer 제네시스 G80 35,000,000 원 전체 제원 분류 제조사 차량 번호 색상 연료 탑승 인원 주행 거리 연식 사고 이력 사고 이력이 없습니다 튜닝 이력 튜닝 이력이 없습니다 구매하기	Display of specific car detail information	**SearchActivity.java**	 원하는 차량을 검색해보세요 연식 연료 제조사 현대 기아 쉐보레 르노삼성 다시 선택하기 검색하기	Display of choose the option for car which leads to show filtered list of cars.
CarRegisterActivity.java	 차량 등록 하기 판매하실 차량을 등록해주세요 차량 번호 183나 1902 차량명 제네시스 G80 제조사 제네시스 연식 2022년 12월 연료 종류 휘발유 색상 블레이징 레드	Display for register newly dealed car (only for seller user)	**SearchResultActivity.java**	 CAU Dealer 요청하신 검색 결과입니다. 현대 e-카운티 디젤 연식 색상 주행거리 2004년식 베이지 99000 km 사고 없음 / 개조 없음 890 만원 쉐보레 다마스 LPG	Display of the list of result for search
LoginActivity.java	 로그인 안녕하세요. Cau Dealer 입니다. 아이디 비밀번호 로그인하기	The first display that User encounter once launch the application. Correct ID and password is required	**UserInfoActivity.java**	 ${userName}님의 보유 차량 목록 기아 봉고3 테슬라 모델3 현대 e카운티 버스 표준형 현대 IONIQ 5 현대 포터2 대우 대우버스FX 버스 212	Display of car list whose car is owned by logined User.
MainActivity.java	 Cau Dealer sans님 안녕하세요! 중고차를 빠르게 팔아보세요 등록된 차량 수 19 매물 등록하기 › 전체 보기 제네시스 G80 주행 거리 연식 색상 76,000 km 2020년 09월 블레이징 레드 사고 없음, 개조 없음 35,000,000 원 벤츠 S클래스	Display that shows the list of all the used-car registered on the application server			

04 Implementations

Package Structure



```
oop4
├ Adapter
│ ├ AccidentTuneRecyclerAdapter.java
│ ├ CarRecyclerAdapter.java
│ └ UserCarRecyclerAdapter.java
├ Model
│ ├ Accident.java
│ ├ Car.java
│ ├ Tune.java
│ └ User.java
├ CarDetailActivity.java
├ CarRegisterActivity.java
├ LoginActivity.java
├ MainActivity.java
├ SearchActivity.java
├ SearchResultActivity.java
└ UserInfoActivity.java
```

Detail Description

<Model>

Model/User.java

Member variables

In the system, we classified users into two types. One is seller, and another should be customer. By the private boolean type variable isSeller, if the value of it is True, it should be seller and if False, it should be customer.

Both seller and customer need the array of car, which represents the cars they own; if seller case, it means the car they registered on the system for sale, and if customer case, it means the car they purchased on the system.

Constructor

The constructor needs String type variable name, and boolean type variable *isSeller*. These are the basic information for the system to make a new user object.

Member methods

For the seller case, the seller needs an interface to register the car on the system, so we add *addCar()* method. It needs Car object(*Car newCar*) as a parameter, the car he's going to register. First, we added if phrase to check if the user is Seller. Only sellers can add cars. Second, if the user is seller, check if the passed Car object(*newCar*)'s number which identifies the object is already in the car list(*Cars*). If there's same number, return false because the duplicate must be avoided in the system. And if not, add *newCar* to *Cars* with exception handling. In short, method returns true if and only if the user is seller, the new car is not yet registered, and there's no exception during adding process.

For the customer case, the customer needs an interface to purchase the car, so we add *buyCar()* method. It needs Car object(*Car targetCar*) and the seller(*User seller*) who registered the car. First, we added if phrase to check if it's customer. Second,

```java
public class User{
    private String userName;
    private boolean isSeller;
    protected ArrayList<Car> Cars = new ArrayList<>();

    public User(String name, boolean isSeller){
        this.userName = name;
        this.isSeller = isSeller;
    }

    // for Seller
    public boolean addCar(Car newCar){
        if(isSeller){
            for (Car car : Cars) { //이미 등록된 차량인지 확인
                if (car.getNumber().equals(newCar.getNumber())) {
                    return false; //이미 등록한 차량일 경우 False
                }
            }
            try {
                Cars.add(newCar);
                return true; //정상적으로 등록된 경우 True
            } catch (Exception e) {
                e.printStackTrace(); //혹시 모를 exception
            }
        }else{
            return false; //판매자 계정이 아니면 false
        }
        return false; //기본값 false
    }

    // for Customer
    public boolean buyCar(Car targetCar, User seller){
          //정상적으로 구매가 이루어졌는지 반환
        if(!isSeller){
            for (Car car : Cars) {
                if (car.getNumber().equals(targetCar.getNumber())) {
                    return false; //이미 구매한 차량이면 false
                }
            }
            try {
                for(int i= 0 ;i < seller.getCarList().size(); i++) {
                    if (seller.getCarList().get(i).getNumber().equals(targetCar.getNumber())) {
```

```
                    Cars.add(targetCar);
                    seller.getCarList().remove(targetCar);
                    return true; //정상적으로 구매한 경우 true
                }
            }
            return false; //끝까지 스캔했는데도 차량이 나오지 않으면 false
        }catch (Exception e){
            e.printStackTrace(); //혹시 모를 exception
        }
    }else{
        return false; //seller 계정이면 false
    }
    return false; //기본값은 false
}
public void showCarList(){
    for(Car car : Cars){
        car.printCarInfo();
    }
}
}
```

Model/Car.java

Member variables

We made an abstraction of the real-world car, so there's quite a lot of member variables. Not to be changed out of the object, all member variables are declared as private. What's notable is, the final variable, ArrayList<Accident> type variable accidents and ArrayList<Tune> type variable tunes. After the registration, accident and tune history must not be modified, because it can be fraud. To avoid this kind of unpleasant things, we declared them as final.

Constructor

The constructor needs all member variables as a parameter. None of these can be missed.

Member methods

Except some getter, setter methods, there's *printCarInfo()* method. As I mentioned above, at the user's *showCarInfo()* method, this *printCarInfo()* method is called to retrieve the information of the cars. So, this function should print all attributes.

```
public class Car{

    private enum Fuel{
        All,
        Diesel,
        Electric,
        Gasoline,
        LPG
    }

    private enum Type{
        All,
        Car,
        Truck,
        Bus
    }

    private String name;
    private String manufacture;
    private String number;
    private String color;
    private String carImage;
    private int price;
    private int capacity;
    private int distanceDriven;
    private int year;
    private Fuel fuel;
    private Type type;
    private boolean isAccident;
    private boolean isTuned;
    private final ArrayList<Accident> accidents = new ArrayList<>();
    private final ArrayList<Tune> tunes = new ArrayList<>();

    public Car(String name, String manufacture, String number, String color, String type, int
price, int capacity, int distanceDriven, int year, String fuel, boolean isAccident, boolean
isTuned){
        this.name = name;
        this.manufacture = manufacture;
        this.number = number;
        this.color = color;
        this.type = Type.valueOf(type);
        this.price = price;
        this.capacity = capacity;
        this.distanceDriven = distanceDriven;
        this.year = year;
        this.fuel = Fuel.valueOf(fuel);
        this.isAccident = isAccident;
        this.isTuned = isTuned;
    }
}
```

Model/Accident.java

Member variables

Accident class must contain date and content. Each means the date when the accident occurs and the content about it.

Constructor

The constructor needs String type variable date and content.

```
public class Accident{
    private String date;
    private String content;

    public Accident(String date, String content){
        this.date = date;
        this.content = content;
    }
}
```

Model/Tune.java

Member variables

This class is almost same with Accident class. Tune class must contain date and content. Each means the date the seller tuned and the content about it.

Constructor

The constructor needs String type variable date and content.

```
public class Tune{
    private String date;
    private String content;

    public Tune(String date, String content){
        this.date = date;
        this.content = content;
    }
}
```

<Android>

I'll explain about the main point of the Android implementations.

LoginActivity.java

In login process, we should fill in username, and userpw to login. There's EditText type value inputID and inputPW which get the filled in values. Also, in this login activity, we will initialize User and from FirebaseFirestore type variable db (which means database for this system), we'll load the car information whose owner is same with the user's. By the given informations, check from database if there's matching id. And return the result on the user's viewport. And if complete login, initialize user. from db, load the car information mathing with user from User table and put it in intent which makes other activity classes interact with each other. (function like static variables)

MainActivity.java

System loads the login information and car information from intent. So, there should be Boolean type value isLoggedIn, isSeller, etc to initialize the user. Also, there's ArrayList<Car> carList that can contain the car information from intent. What we should focus on is that the car which is sold already should not be contained. And there's many buttons to go to other pages such as user information, car search, car add, etc.

UserInfoActivity.java

In this class, we just load the user information from intent object which match with the user.

CarDetailActivity.java

This activity will display the details of the car and provide an instant buy button to conveniently purchase the car. When the user taps the certain car in the display, it will get the information of the selected car in the carlist and display the information of the car. Each row that displays the information will use the getter of the 'Car' class in order to get the encapsulated internal data and display the information. If the client taps the buy button, it will perform the User.buyCar() and insert the data to the database of the server if operated successfully.

CarRegisterActivity.java

This activity uses the functions of the 'Activity' class provided by android, and the functions of the 'User' class we implemented. Clients will type the information of the car in the textfield created with the library of android, and after that, informations typed by clients will be converted into the string and will be passed into the User.addCar() that we implemented before. If the function performed properly, the data will be sent to the user's database in the firebase, and notify the clients that the car is successfully registered.

SearchActivitiy.java

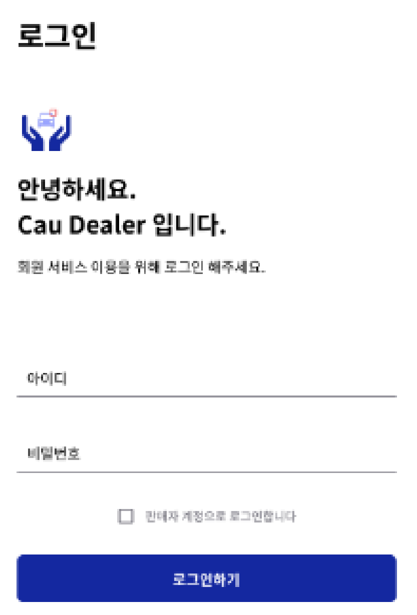
The function will get the properties with parameters, and then find the car object that has the corresponding properties in the database of the firebase. The result can have multiple cars, and thus declaring the array list of the car object and appending into it will produce the proper outcome. Also, we can set the condition, either searching all the type or the fuels, or searching with the restricted condition of the type and fuels.

SearchResult.java

After performing the ‘SearchActivity’ there should be the list of the cars stored by performing search. This activity will make a visual view of the result that will be displayed to the clients. Using the built-in ‘Activity’ class of the android system, will setup the layout to be displayed, by getting the object of the car and inserting the cell row by row.

05 Execution Results

Login



로그인

안녕하세요.
Cau Dealer 입니다.

회원 서비스 이용을 위해 로그인 해주세요.

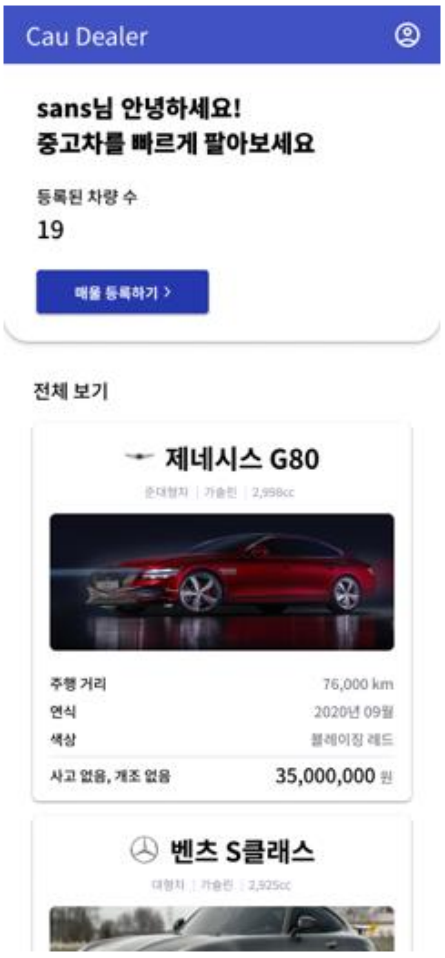
아이디

비밀번호

☐ 판매자 계정으로 로그인합니다

로그인하기

Main



Cau Dealer

sans님 안녕하세요!
중고차를 빠르게 팔아보세요

등록된 차량 수
19

매물 등록하기 >

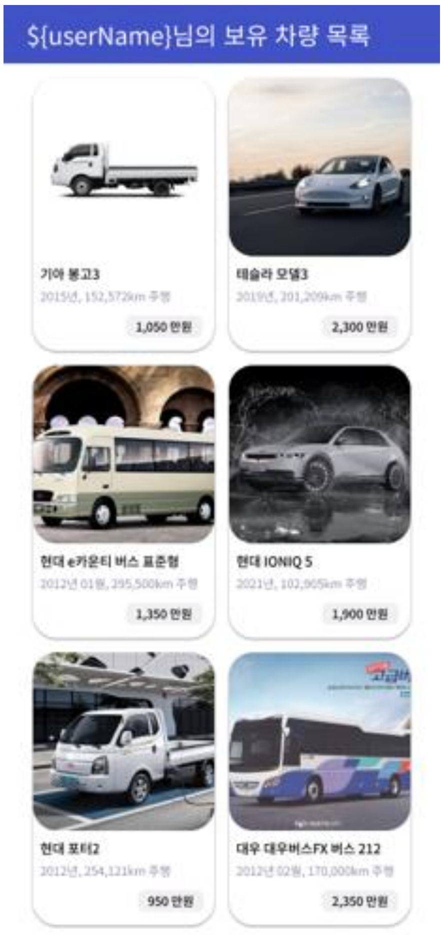
전체 보기

제네시스 G80
준대형차 | 가솔린 | 2,998cc

주행 거리	76,000 km
연식	2020년 09월
색상	블레이징 레드
사고 없음, 개조 없음	35,000,000 원

벤츠 S클래스
대형차 | 가솔린 | 2,925cc

User Info



${userName}님의 보유 차량 목록

기아 봉고3
2015년, 152,572km 주행
1,050 만원

테슬라 모델3
2019년, 201,209km 주행
2,300 만원

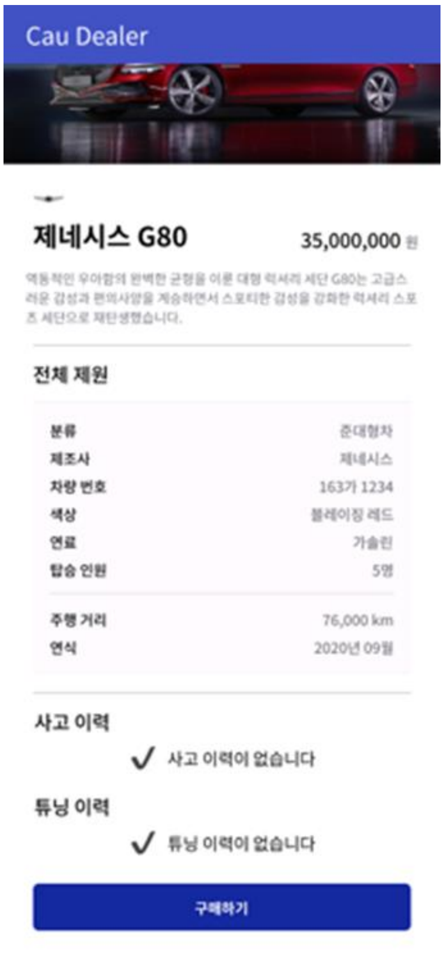
현대 e카운티 버스 표준형
2012년 01월, 295,500km 주행
1,350 만원

현대 IONIQ 5
2021년, 102,905km 주행
1,900 만원

현대 포터2
2012년, 254,121km 주행
950 만원

대우 대우버스FX 버스 212
2012년 02월, 170,000km 주행
2,350 만원

Car Detail



Cau Dealer

제네시스 G80 35,000,000 원

역동적인 우아함의 완벽한 균형을 이룬 대형 럭셔리 세단 G80는 고급스러운 감성과 편의사양을 계승하면서 스포티한 감성을 강화한 럭셔리 스포츠 세단으로 재탄생했습니다.

전체 제원

분류	준대형차
제조사	제네시스
차량 번호	163가 1234
색상	블레이징 레드
연료	가솔린
탑승 인원	5명
주행 거리	76,000 km
연식	2020년 09월

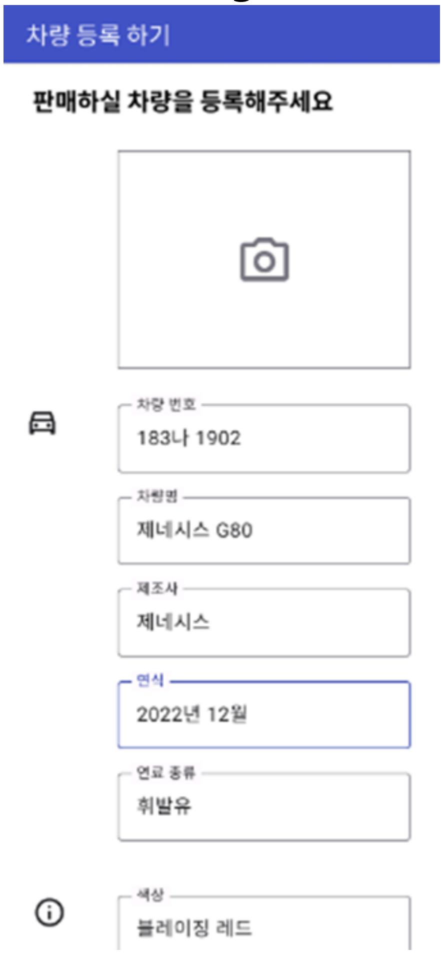
사고 이력

✓ 사고 이력이 없습니다

튜닝 이력

✓ 튜닝 이력이 없습니다

Car Register

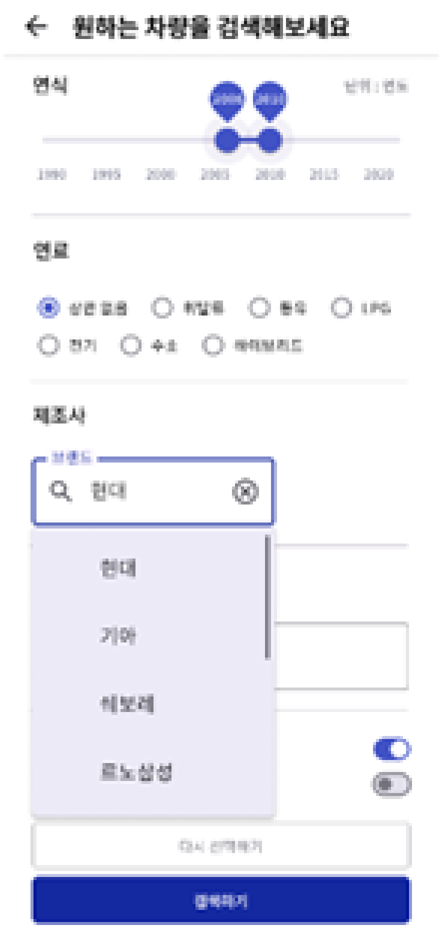


차량 등록 하기

판매하실 차량을 등록해주세요

차량 번호	183나 1902
차량명	제네시스 G80
제조사	제네시스
연식	2022년 12월
연료 종류	휘발유
색상	블레이징 레드

Search



원하는 차량을 검색해보세요

연식 단위 : 연도

1990 1995 2000 2005 2010 2015 2020

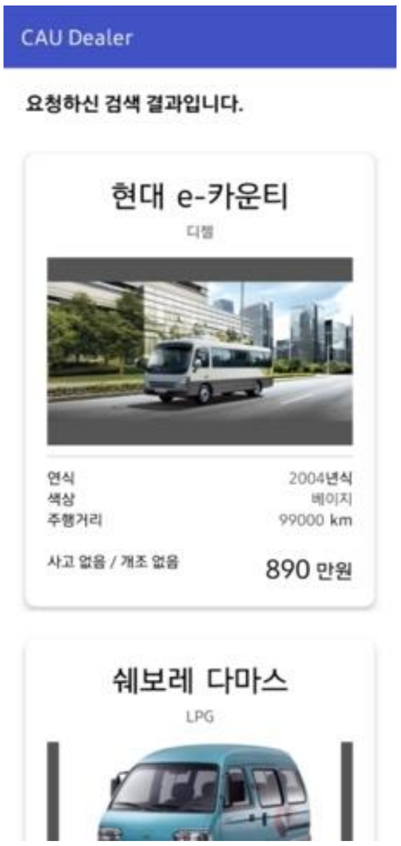
연료

[✓ 상관 없음] 휘발유 경유 LPG 전기 수소 하이브리드

제조사

브랜드: 현대

Search Result



CAU Dealer

요청하신 검색 결과입니다.

현대 e-카운티

디젤

연식	2004년식
색상	베이지
주행거리	99000 km

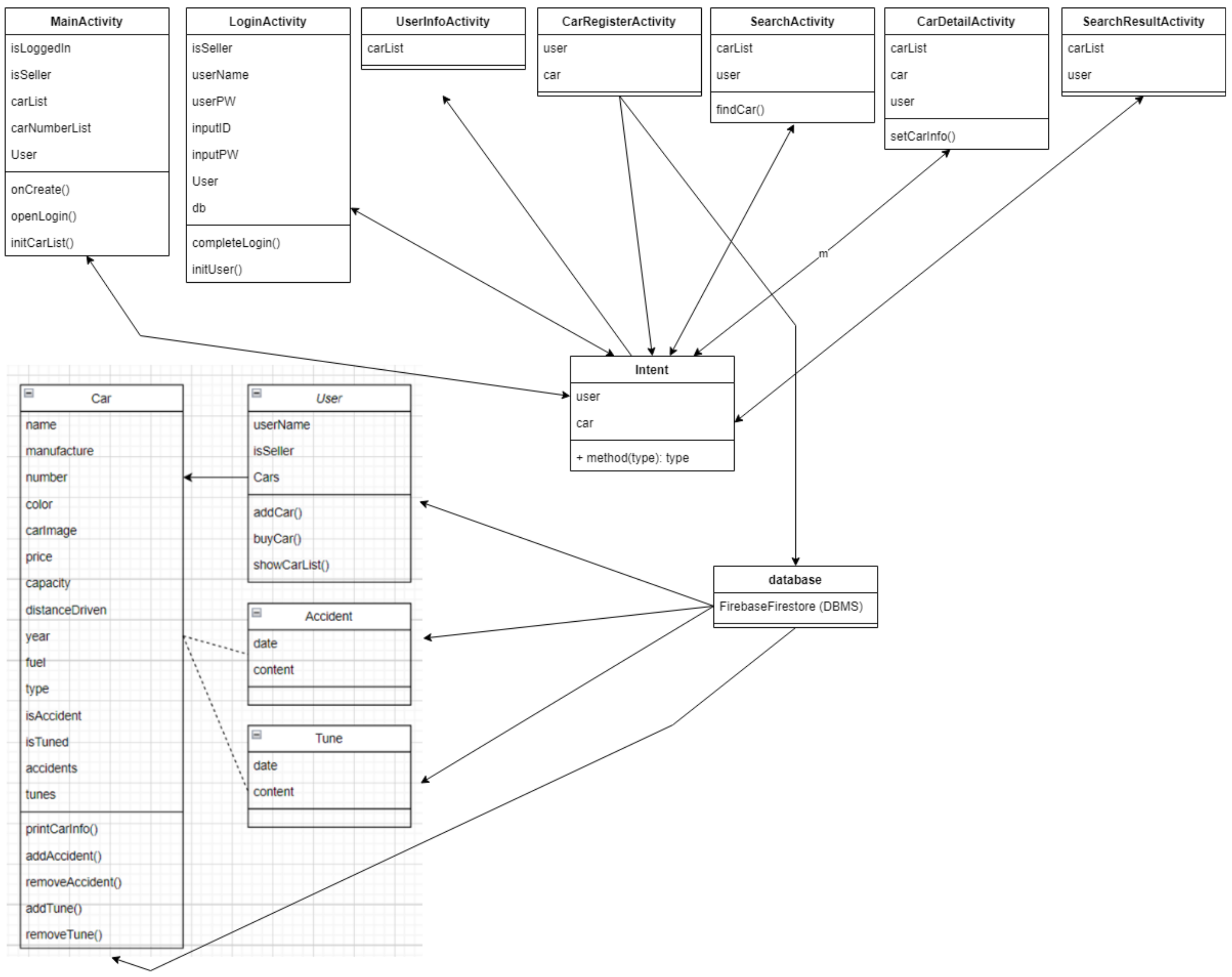
사고 없음 / 개조 없음 890 만원

쉐보레 다마스

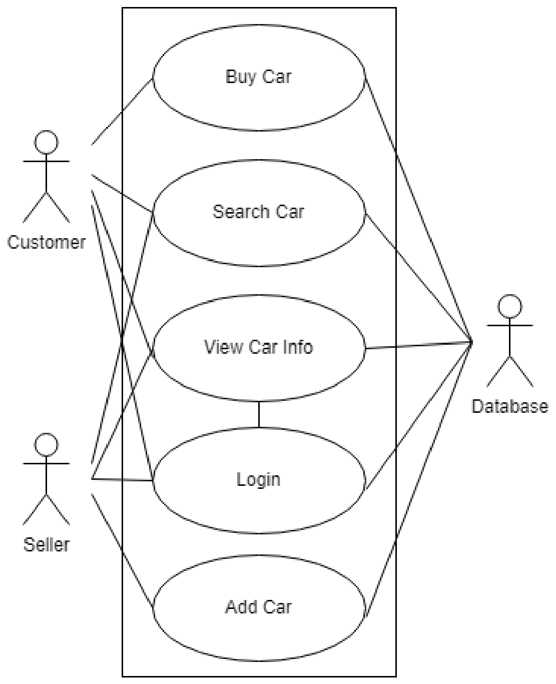
LPG

06 UML Modeling

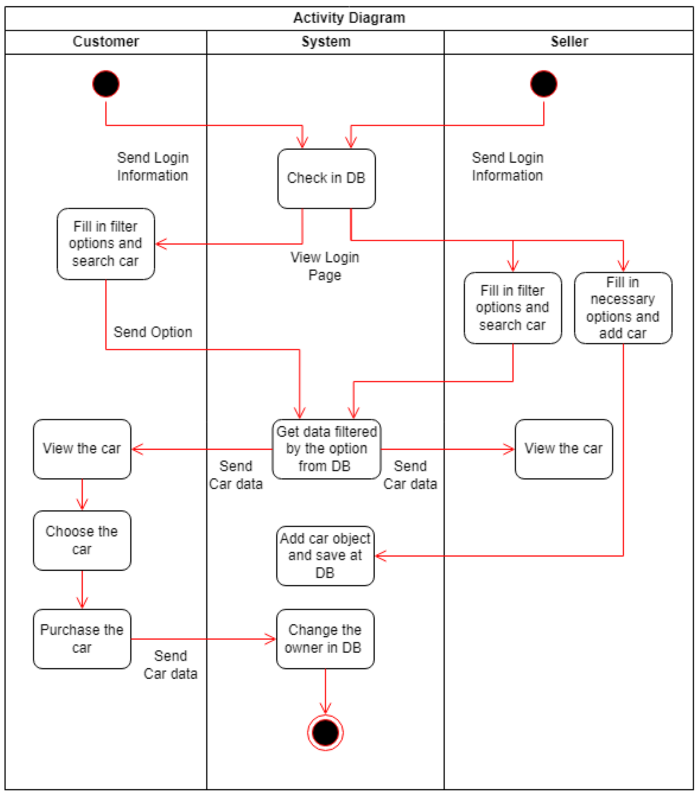
Class Diagram



Use-case diagram



Activity Diagram



07 Conclusion

Our project, 'CAUDealer' is the ultimate composition of the concepts of the 'Object-Oriented-Programming' we studied in this semester, including abstraction, polymorphism, encapsulation, etc. Three main classes we implemented in the project are 'User', 'Car', and 'Activity,' and there exists an inheritance process between the subclasses, for example, 'Customer', and 'Seller', and the parental class 'User.' Using the inheritance allowed us to easily extend the feature of the function, making our project have more various functions. For example, when we had to define the new activities such as login, implementing the login activity without inheriting built-in activity class in android system, would cost numerous time to implement. However, by inheriting the class, we could just simply add required functions while using the function of the built-in activity to operate properly in android.

In addition, while using the built-in library in android, we could found the library's flawless structure of the class designing, that satisfied the requirements of the good program, which helped us to get the knowledge of designing the classes in a better way. And the process of combining the classes we designed, with the classes that android provides, allowed us to understand how the classes interact each other in order to make the program operate properly.

Furthermore, we used the Firebase in order to use the database that exists in the server, which required the conversion of the context of the objects into the compatible context of the database. This brought us some difficulties to overcome, as we had to understand not only the differences of the structure of the data and the objects, but also the proper way to connect them. After hard working on the process, we found that there is the important concept we learned in this semester, called 'polymorphism.' We had to override various properties and functions in order to work well in completely different environment (firebase). Making functions and members be able to take on various forms could increase the compatibility of our program, leading to the successful operation in the firebase, too.

After completing the development of the program, we felt that we could successfully finish developing our program as we learned the important principles of the 'Object Orient Programming' in this semester and facilitated these well.

08 How to compile and execute

Environment

- OS: macOS 13 Ventura
- Architecture: arm64
- SDK: Android 33 Tiramisu

Source Code Structure

- App Source (/app/src/main): Entire Code of Application
- Base Package (/app/src/main/java/com/oop7even/oop4): Java Source Codes
- Inside Base Package
  -> *Activity.java: Each activity Source Codes of Application
  -> Model/Directory: Accident/Car/Tune/User Class
  -> Adpater/Directory: Adapter Source Codes of RecyclerView List in Application

Execution

- If Android SDK is Setted
  -> Open Project with Android Studio

Used Android Libraries

- AndroidX AppCompat by Google
- AndroidX CardView by Google
- Material Components by Google
- Firebase Firestore by Google
- Image Picker by dhaval2404