ADA UNIVERSITY
SCHOOL OF IT & ENGINEERING

Course: Object-Oriented Analysis & Design

HOMEWORK ASSIGNMENT 2

Project title: ***“Taxi Dispatch Application”***

Students: Farida Aliyeva
Mustafa Aslanov
Aliya Bannayeva
Fidan Ismayilova

Instructor: **Dr. Abzatdin Adamov**

Baku 2020

Contents

Introduction

This document describes parts of system analysis for a Taxi Dispatch Application. Concretely, it consists of the functional and nonfunctional software requirements, which are also divided into core and optional, list of possible use-cases, class diagram and a sequence diagram for one of the most important use-cases.

The Taxi Dispatch Application's primary purpose is to allow its users order a taxi wherever they are and get an immediate response. In the busy city we live in, getting places like work in time is crucial, and that is why our Taxi Dispatch Application ensures taxi availability and ease of order.

The analysis in this document aims to show, at first glance, the general structure of this application.

Requirements

ID	Type	Functional Requirements
R1	core	Both cash and credit card payment methods.
R2		Customer & Driver information
R3		Algorithm to calculate best and quickest road
R4		Multilanguage interface (Azerbaijani, English, Russian minimum)
R5		User-friendly interface
R6		Chat feature (driver-customer, customer-customer support)
R7		Rating system for both drivers and customers (as well as for the whole application)
R8	optional	Social media integration
R9		Different type of cars including: basic, premium and special care (for people with disabilities or big families)
R10		Payment will be done in Azerbaijani currency (AZN)
R11		Multiple destination points
R12		Bonus system/Promo codes
R13		Management of notifications and alerts
R14		Pre-booking feature
R15		Airport and Hotel booking terminals
R16		Feedback options
R17		Fare calculation (Basic Fare, Cost per minute/kilometer, Pre-booking fee)
R18		Built in maps
R19		Geo-location tracking (Geo-fencing)
R20		Customer usage analytics

ID	Non-functional Requirements
R21	No lags or freezes in case if there is sudden overflow of customers (Scalability).
R22	No errors especially resulting in crash (Reliability - Fault)
R23	The application won't fail on basic requirements like proper location detection, failure on payment through credit card, incorrect timing, correct destination points, etc. (Reliability – Failure)
R24	Provision of the proper prices for rides, information about previous and current rides, ability to switch from cash to credit card payment (Usability)
R25	Quick response time when ordering taxi, constant connection with server (Performance)
R26	Security of using credit card information for payments (Security)
R27	Personal information being hidden from each side (driver and customer) (Security)
R28	Ability to use application on both Android (5.0+) and iOS (6.0+) systems. (Supportability)
R29	Easiness of maintainability and room for improvements. (Maintainability)

Use Cases

Use Case Name: Customer & Driver Info	**ID: UC-1**	**Priority:** High
Actor: Customer		
Description: Customer is able to check their information as well as their designated driver's information.		
Trigger: Customer checks own or driver's details. **Type:** Service		
Preconditions: The user has entered their own details and have ordered a taxi.		
Normal Course: 1. User opens application 2. User proceeds to the "User Info" page 3. User can see their details.	**Information for Steps:** 1. Opening app from applications list 2. In the list of pages selecting the one that says "User Info" 3. Being able to see their name, mobile number and current location.	
Alternative Courses: User wants to view driver info. 1. User opens application 2. User proceeds to the "Driver Info" page 3. User sees their designated driver info.	User sees their driver's name, availability, rating, taxi car model and taxi class.	
Postconditions: The user able to see their personal details they have entered and the details of the taxi driver and their car that has been assigned to them.		

Use Case Name: Payment	**ID: UC-2**	**Priority:** High
Actor: Customer		
Description: Customer is willing to choose the way to pay for the rides and be able to switch whenever they want to.		
Trigger: Customer choosing the payment method. **Type:** Security		
Preconditions: The user choosing one of the methods for paying for the ride.		
Normal Course: 1. User opens application 2. User proceeds to the "Payment" page. 3. User selects either "Cash" or "Credit Card" payment.	**Information for Steps:** 1. Opening app from applications list 2. In the list of pages selecting the one that says "Payment". 3. From the given option selects the one that's suitable for them.	
Alternative Courses: User can't use either of the paying methods or the one they are willing to choose is not listed (ex.: PayPal, Bitcoins, etc.). 1. User opens application 2. User proceeds to the "Payment" page. 3. User is unable to find the preferred method.	If there were no preferred methods for paying, user won't be able to request taxi.	
Postconditions: The user now can pay in the end of the ride if chose "Cash" method, or pre-pay for the ride if chose "Credit Card" method.		

Use Case Name: Shortest Route	**ID: UC-3**	**Priority:** High
Actor: Customer		
Description: Customer requires to be transported via the shortest available route for maximum efficiency, using distance calculating algorithm.		
Trigger: Customer orders taxi. **Type:** Core		
Preconditions: The user provides their location and orders taxi.		
Normal Course: 1. User opens application 2. User orders taxi. 3. User sees the time when the driver will arrive **include** (Customer Location)	**Information for Steps:** 1. Customer broadcasts their location and requests taxi. 2. Taxi driver responds to the request if they're close enough to customer's location. 3. User sees driver's arrival time.	
Alternative Courses: [no available route] User gets no time information.	If the route the user requested is not available, no approximate driver arrival time will be calculated.	
Postconditions: The user will view driver arrival time.		

Use Case Name: Customer Location	**ID: UC-4**	**Priority:** High
Actor: Customer		
Description: Customer can input their location.		
Trigger: Customer opens the application. **Type:** Service		
Preconditions: The user opening the maps page.		
Normal Course: 1. User opens application 2. User enters their current location.	**Information for Steps:** 1. Opening app from applications list 2. Entering their credentials followed by their location.	
Alternative Courses: [user unable to enter their location] 1. Application cannot proceed.	If there are issues where customer does not know their location, a taxi will not be able to be assigned to them.	
Postconditions: The user can now order taxi.		

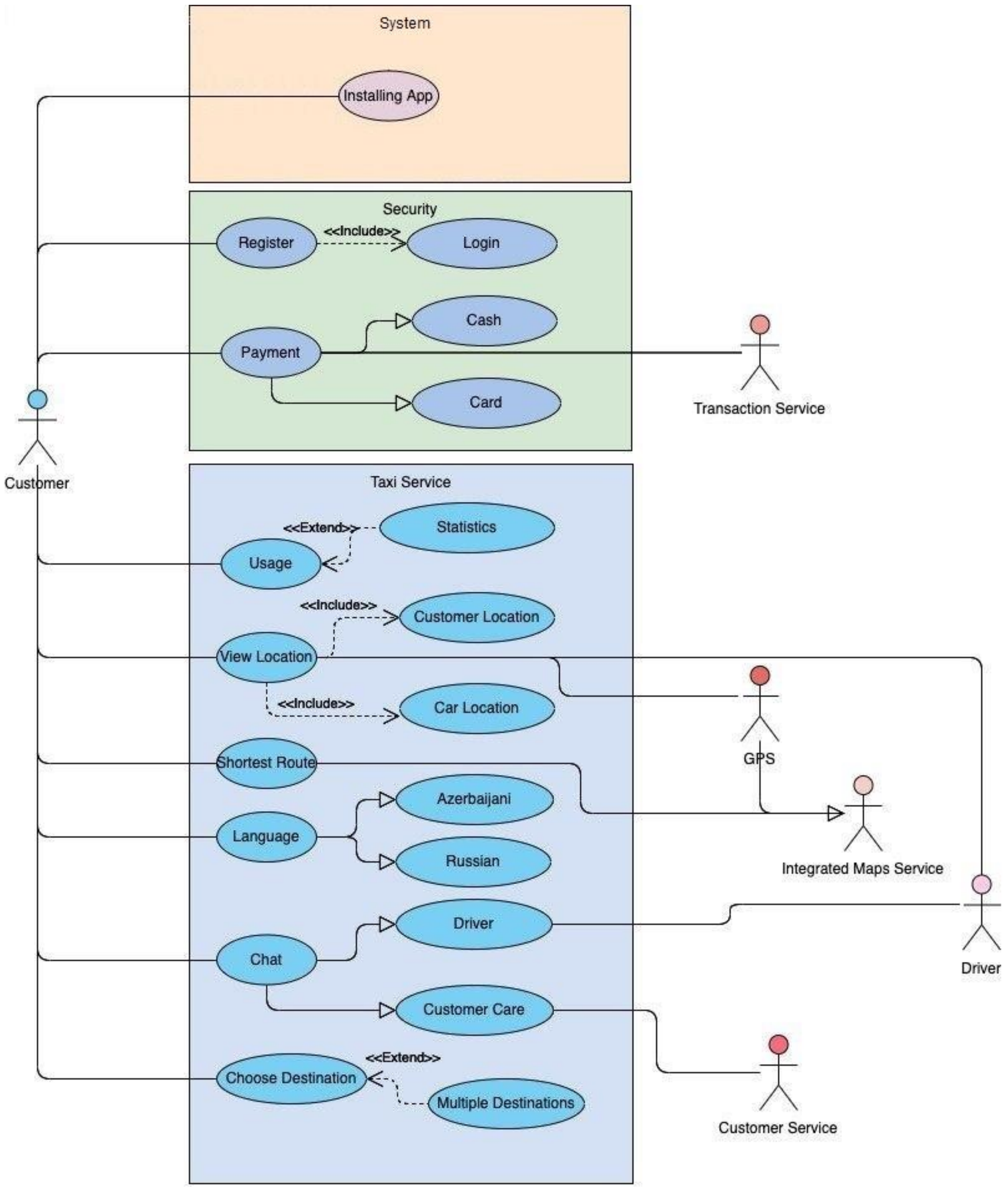
Use Case Name: Chat	**ID: UC-5**	**Priority:** High
Actor: Customer		
Description: Customer is willing to able to contact driver/customer support using app’s integrated chat system.		
Trigger: Customer’s need to contact either driver and/or customer support. **Type:** Service		
Preconditions: The user choosing whom to chat.		
Normal Course: 1. User opens application 2. User proceeds to the “Chat” page. 3. User selects either “Driver” or “Customer Support” options.	**Information for Steps:** 1. Opening app from applications list 2. In the list of pages selecting the one that says “Chat”. 3. From the given option selects the one that’s suitable for them.	
Alternative Courses: User unable to contact either recipient using integrated chat system. 1. User opens application 2. User proceeds to the “Chat” page. 3. User is unable to send messages to either driver or customer support.	If there are issues with the sending messages, user may call either driver or customer support in case of emergency need.	
Postconditions: The user now can chat with either driver and/or customer support agent.		

Use Case Name: Booking varieties	**ID: UC-6**	**Priority:** Low
Actor: Customer		
Description: Customer is willing to be able to pre-book the ride and/or have multiple destination points in one ride.		
Trigger: Customer specifying the type of a ride. **Type:** Service		
Preconditions: The user choosing one or more of the features for the ride.		
Normal Course: 1. User opens application 2. User proceeds to the "Order" page. At Step 2 user is able to specify at what time driver should come (either at the moment or later) AND/OR specify one or multiple destinations. 3. Taxi is ordered with or without extended features.	**Information for Steps:** 1. Opening app from applications list 2. While user is ordering the taxi, they can extend the application services by: a. Picking special time for the arrival of the driver. b. Picking more than one destination location. 3. If the user does not choose the extra options, they are still able to order a regular taxi which will arrive immediately.	
Alternative Courses: User wants to pre-book for more than a week time period and/or more than 5 destination points: 1. User opens application 2. User proceeds to the "Order" page. 3. User is unable to pick time which exceeds 1-week period. 4. User is unable to pick more than 5 locations within one ride.	If there a case when user needs to pre-order they will have to wait until there isn't more than 1-week gap. If there's a need for more than 5 destinations, user can re-order new taxi.	
Postconditions: The user now can pre-book the ride or pick more than 1 (but no more than 5) destination points.		

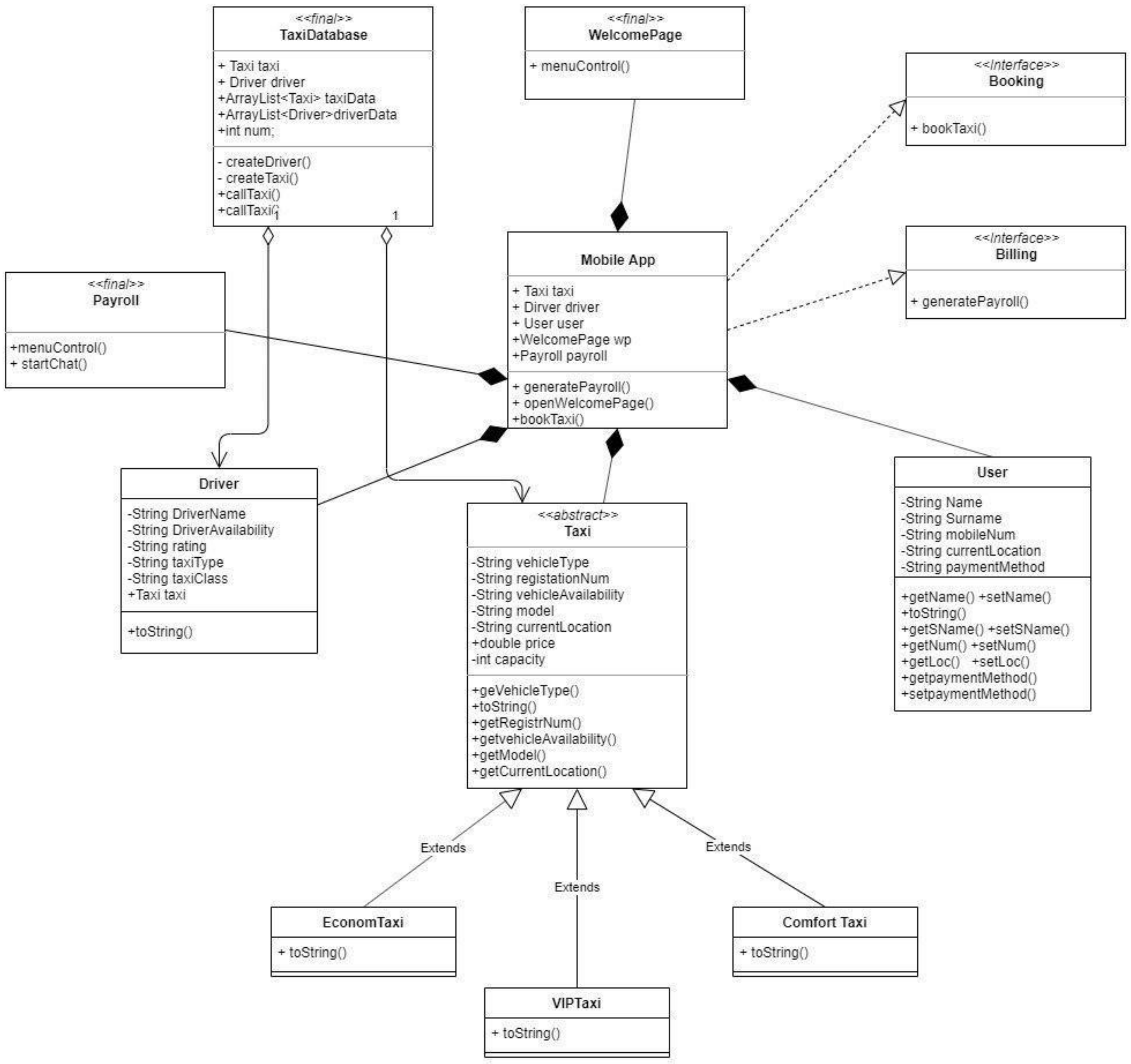
Use Case Name: Language	**ID: UC-7**	**Priority:** Medium
Actor: Customer		
Description: Customer requires to be able to choose the language in which they would like to use the application.		
Trigger: Customer enters language options. **Type:**		
Preconditions: The user is using the application and wants to change the language.		
Normal Course: 1. Customer clicks on language options. 2. Customer finds and chooses their desired language.	**Information for Steps:** 1. Customer can view the languages the application provides. At first, there will be English, Azerbaijani and Russian. 2. Customer can choose a single language to use the application in.	
Alternative Courses: [language unavailability] Repeat step 2 opting for a different language.	If customer does not find their native/desired language in the list of available language, it means it has not been added yet and they will have to use the application in their second language that is available.	
Postconditions: The user will be able to use the application in their desired language.		

Use Case Name: Ease of use	**ID: UC-8**	**Priority:** High
Actor: Customer		
Description: Customer requires to be able to navigate the application intuitively.		
Trigger: Customer opens the application. **Type:**		
Preconditions: The user is using the application.		
Normal Course: 1. User immediately recognizes taxi order button. 2. Customer takes less than 3 clicks to order the taxi.	**Information for Steps:** 1. No part of the UI makes the customer think about its functionality, the customer immediately understands what each button/display does. 2. If a customer needs to access one of the application's features, it should not take more than 3 clicks to reach.	
Alternative Courses: [customer fails to easily locate taxi order button] It takes the user longer time to use the application's features, but eventually they repeat steps 1 & 2. [it takes more than 3 clicks] 1. Customer takes more than 3 clicks to order a taxi, making the service less efficient.	If a customer does not find the application UI intuitive and quick to use, their user experience will be impeded.	
Postconditions: The user will be able to quickly order a taxi.		

Use Case Diagram



Class Diagram



Java Code:

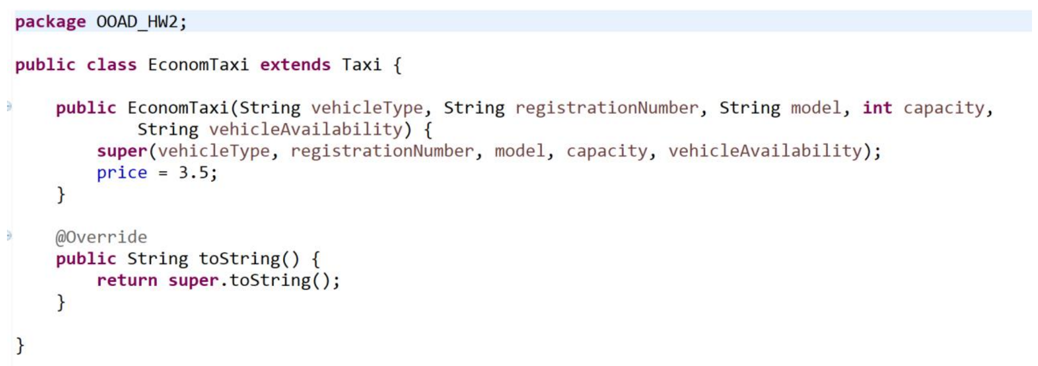
```java
package OOAD_HW2;

public class MobileApp implements Booking, Billing {
    Taxi taxi;
    User user;
    Driver driver;
    WelcomePage wp;
    Payroll payroll;

    public MobileApp(TaxiDatabase db, User user) {
        this.taxi = db.callTaxi();
        this.user = user;
        this.driver = db.callDriver();
        openWelcomePage();
    }

    private void openWelcomePage() {
        wp = new WelcomePage();
        wp.menuControl(taxi, user, driver);
        generatePayroll();
    }

    @Override
    public void generatePayroll() {
        payroll = new Payroll();
        payroll.menuControl(taxi, user, driver);
        bookTaxi();
    }

    @Override
    public void bookTaxi() {
        System.out.println("Taxi booked");
    }

}
```

```java
package OOAD_HW2;

public interface Booking {
    public void bookTaxi();
}
```

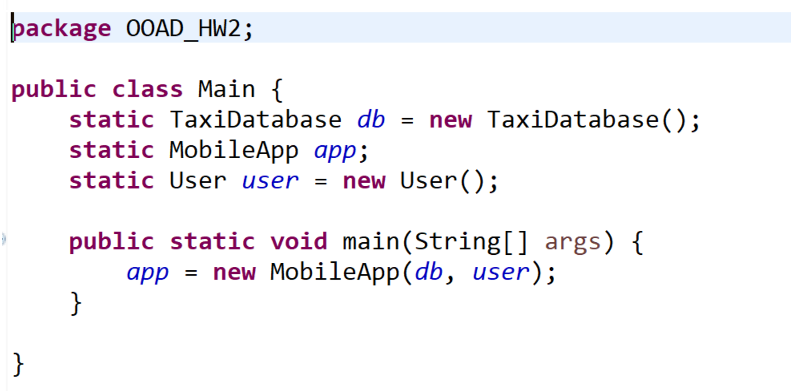
```java
pa OOAD/src/OOAD_HW2/Billing.java

public interface Billing {

    public void generatePayroll();
}
```

```java
package OOAD_HW2;

public class ComfortTaxi extends Taxi {

    public ComfortTaxi(String vehicleType, String registrationNumber, String model, int capacity,
            String vehicleAvailability) {
        super(vehicleType, registrationNumber, model, capacity, vehicleAvailability);
        price = 6.99;
    }

    @Override
    public String toString() {
        return super.toString();
    }
}
```

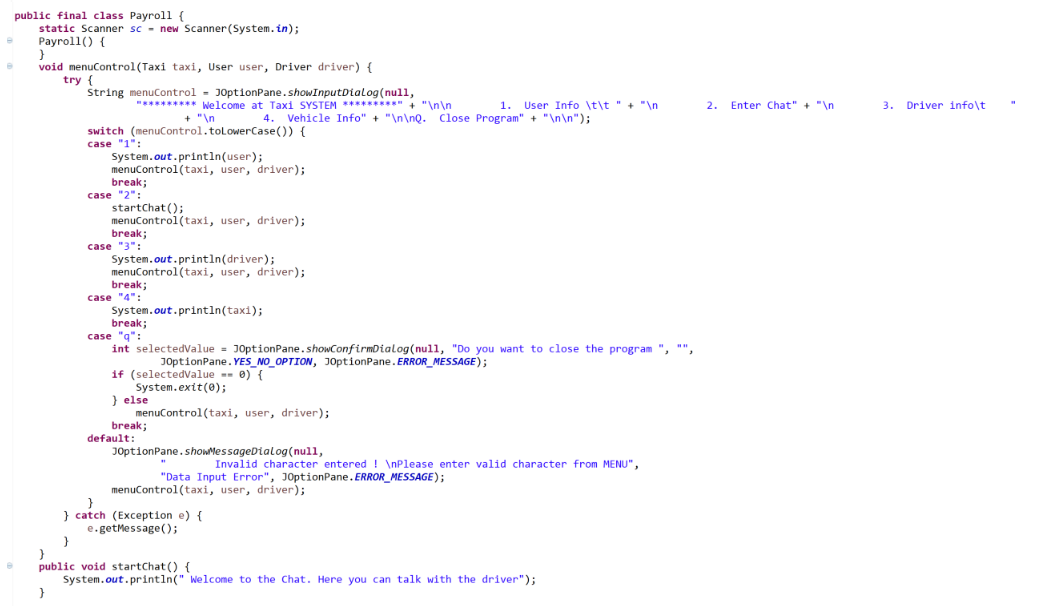
```java
package OOAD_HW2;

public class Driver {

    private String driverName, driverAvailability, rating, taxiType, taxiClass;
    Taxi taxi;

    public Driver(String driverName, String rating, Taxi taxi) {
        this.driverName = driverName;
        this.rating = rating;
        this.taxiType = taxi.getModel();
        this.taxiClass = taxi.getVehicleType();
        this.driverAvailability = taxi.getVehicleAvailability();
    }

    @Override
    public String toString() {
        return "\nNAME:\t      " + driverName + "\nAvailability: " + driverAvailability + "\nRating: " + rating
                + "\nTaxi Car Model: " + taxiType + "\nTaxi Class: " + taxiClass
                + "\n*********************************";
    }

}
```



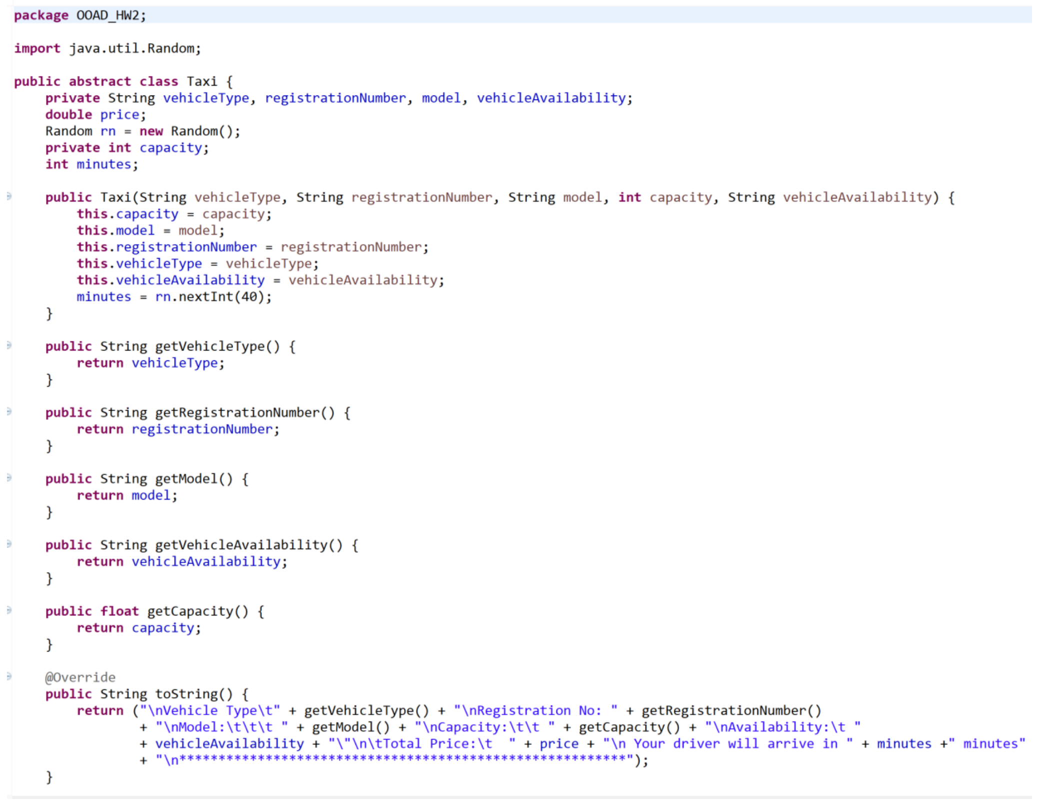
```java
package OOAD_HW2;

public class EconomTaxi extends Taxi {

    public EconomTaxi(String vehicleType, String registrationNumber, String model, int capacity,
            String vehicleAvailability) {
        super(vehicleType, registrationNumber, model, capacity, vehicleAvailability);
        price = 3.5;
    }

    @Override
    public String toString() {
        return super.toString();
    }

}
```



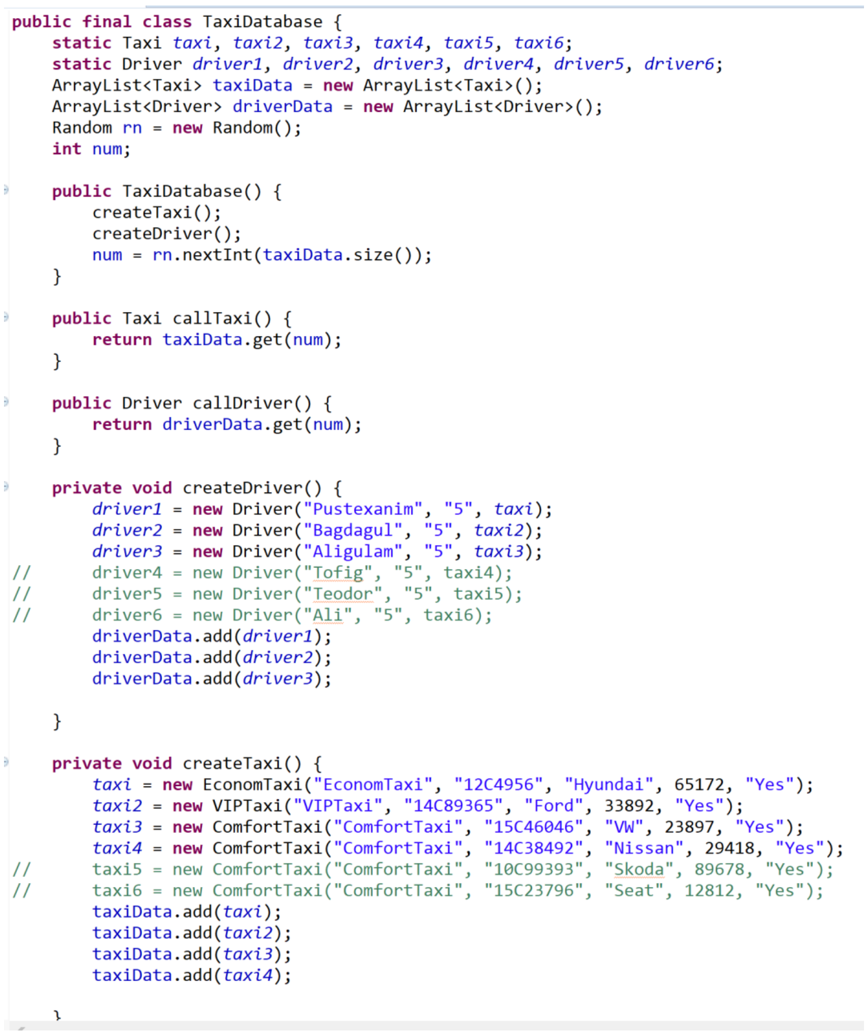
```java
package OOAD_HW2;

public class Main {
    static TaxiDatabase db = new TaxiDatabase();
    static MobileApp app;
    static User user = new User();

    public static void main(String[] args) {
        app = new MobileApp(db, user);
    }

}
```



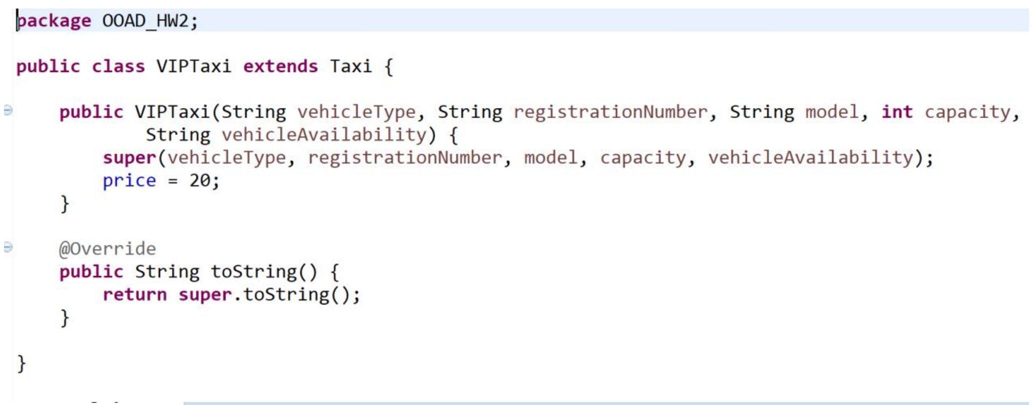
```java
public final class Payroll {
    static Scanner sc = new Scanner(System.in);
    Payroll() {
    }
    void menuControl(Taxi taxi, User user, Driver driver) {
        try {
            String menuControl = JOptionPane.showInputDialog(null,
                    "********* Welcome at Taxi SYSTEM *********" + "\n\n        1.  User Info \t\t " + "\n        2.  Enter Chat" + "\n        3.  Driver info\t    "
                            + "\n        4.  Vehicle Info" + "\n\nQ.  Close Program" + "\n\n");
            switch (menuControl.toLowerCase()) {
            case "1":
                System.out.println(user);
                menuControl(taxi, user, driver);
                break;
            case "2":
                startChat();
                menuControl(taxi, user, driver);
                break;
            case "3":
                System.out.println(driver);
                menuControl(taxi, user, driver);
                break;
            case "4":
                System.out.println(taxi);
                break;
            case "q":
                int selectedValue = JOptionPane.showConfirmDialog(null, "Do you want to close the program ", "",
                        JOptionPane.YES_NO_OPTION, JOptionPane.ERROR_MESSAGE);
                if (selectedValue == 0) {
                    System.exit(0);
                } else
                    menuControl(taxi, user, driver);
                break;
            default:
                JOptionPane.showMessageDialog(null,
                        "         Invalid character entered ! \nPlease enter valid character from MENU",
                        "Data Input Error", JOptionPane.ERROR_MESSAGE);
                menuControl(taxi, user, driver);
            }
        } catch (Exception e) {
            e.getMessage();
        }
    }
    public void startChat() {
        System.out.println(" Welcome to the Chat. Here you can talk with the driver");
    }
```



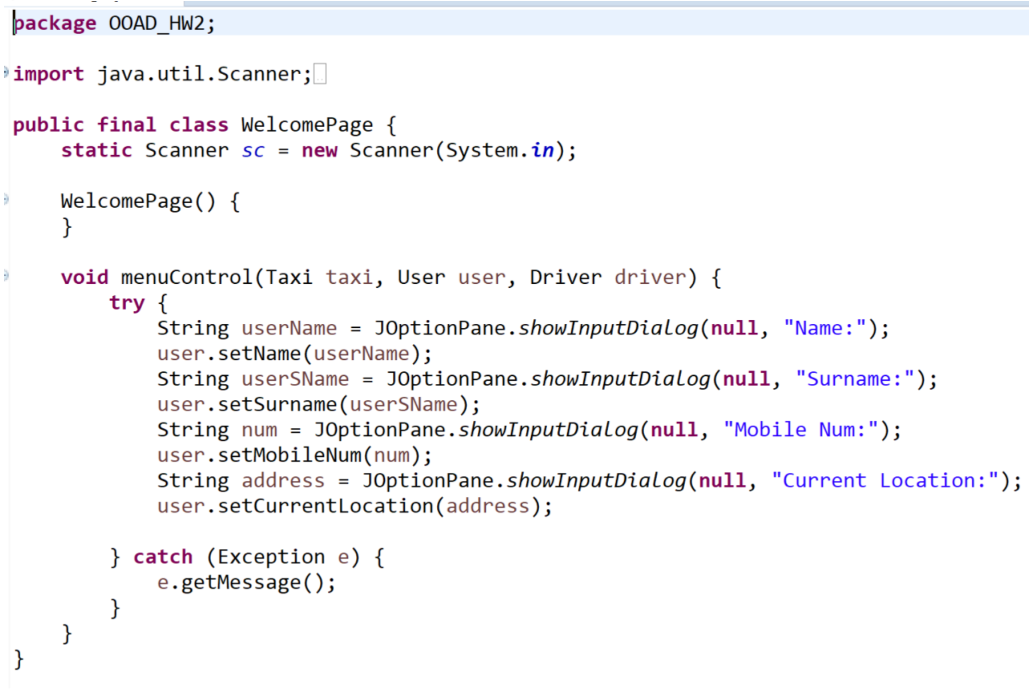
```java
package OOAD_HW2;

import java.util.Random;

public abstract class Taxi {
    private String vehicleType, registrationNumber, model, vehicleAvailability;
    double price;
    Random rn = new Random();
    private int capacity;
    int minutes;

    public Taxi(String vehicleType, String registrationNumber, String model, int capacity, String vehicleAvailability) {
        this.capacity = capacity;
        this.model = model;
        this.registrationNumber = registrationNumber;
        this.vehicleType = vehicleType;
        this.vehicleAvailability = vehicleAvailability;
        minutes = rn.nextInt(40);
    }

    public String getVehicleType() {
        return vehicleType;
    }

    public String getRegistrationNumber() {
        return registrationNumber;
    }

    public String getModel() {
        return model;
    }

    public String getVehicleAvailability() {
        return vehicleAvailability;
    }

    public float getCapacity() {
        return capacity;
    }

    @Override
    public String toString() {
        return ("\nVehicle Type\t" + getVehicleType() + "\nRegistration No:  " + getRegistrationNumber()
                + "\nModel:\t\t\t " + getModel() + "\nCapacity:\t\t " + getCapacity() + "\nAvailability:\t "
                + vehicleAvailability + "\"\n\tTotal Price:\t  " + price + "\n Your driver will arrive in " + minutes +" minutes"
                + "\n**********************************************************");
    }
```



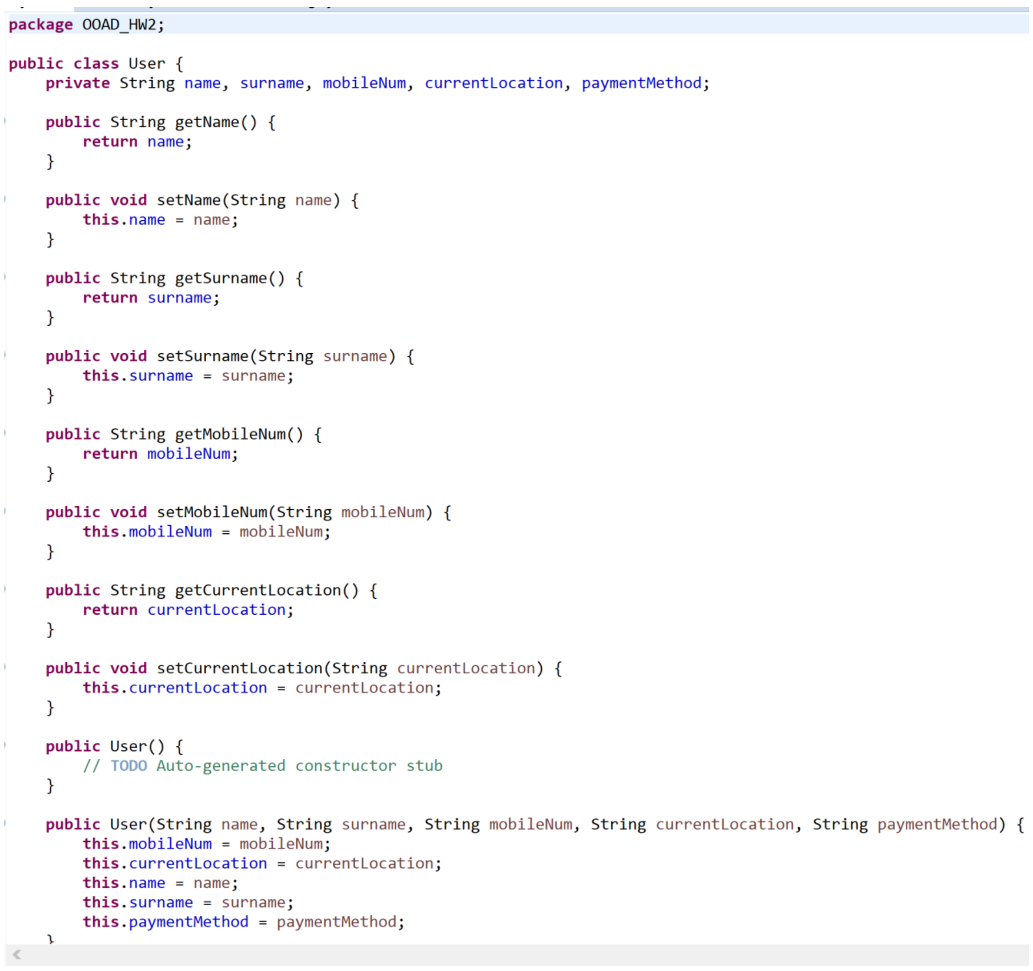
```java
public final class TaxiDatabase {
    static Taxi taxi, taxi2, taxi3, taxi4, taxi5, taxi6;
    static Driver driver1, driver2, driver3, driver4, driver5, driver6;
    ArrayList<Taxi> taxiData = new ArrayList<Taxi>();
    ArrayList<Driver> driverData = new ArrayList<Driver>();
    Random rn = new Random();
    int num;

    public TaxiDatabase() {
        createTaxi();
        createDriver();
        num = rn.nextInt(taxiData.size());
    }

    public Taxi callTaxi() {
        return taxiData.get(num);
    }

    public Driver callDriver() {
        return driverData.get(num);
    }

    private void createDriver() {
        driver1 = new Driver("Pustexanim", "5", taxi);
        driver2 = new Driver("Bagdagul", "5", taxi2);
        driver3 = new Driver("Aligulam", "5", taxi3);
//      driver4 = new Driver("Tofig", "5", taxi4);
//      driver5 = new Driver("Teodor", "5", taxi5);
//      driver6 = new Driver("Ali", "5", taxi6);
        driverData.add(driver1);
        driverData.add(driver2);
        driverData.add(driver3);

    }

    private void createTaxi() {
        taxi = new EconomTaxi("EconomTaxi", "12C4956", "Hyundai", 65172, "Yes");
        taxi2 = new VIPTaxi("VIPTaxi", "14C89365", "Ford", 33892, "Yes");
        taxi3 = new ComfortTaxi("ComfortTaxi", "15C46046", "VW", 23897, "Yes");
        taxi4 = new ComfortTaxi("ComfortTaxi", "14C38492", "Nissan", 29418, "Yes");
//      taxi5 = new ComfortTaxi("ComfortTaxi", "10C99393", "Skoda", 89678, "Yes");
//      taxi6 = new ComfortTaxi("ComfortTaxi", "15C23796", "Seat", 12812, "Yes");
        taxiData.add(taxi);
        taxiData.add(taxi2);
        taxiData.add(taxi3);
        taxiData.add(taxi4);

    }
```



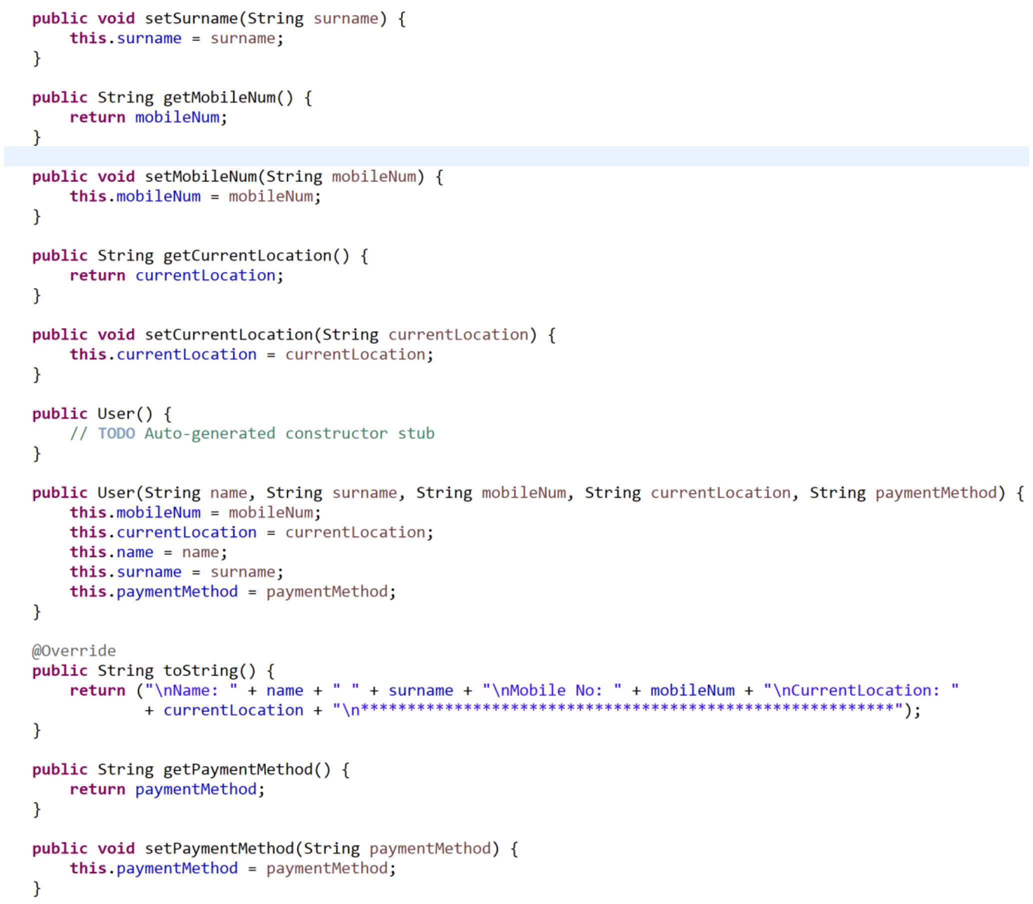
```java
package OOAD_HW2;

public class VIPTaxi extends Taxi {

    public VIPTaxi(String vehicleType, String registrationNumber, String model, int capacity,
            String vehicleAvailability) {
        super(vehicleType, registrationNumber, model, capacity, vehicleAvailability);
        price = 20;
    }

    @Override
    public String toString() {
        return super.toString();
    }

}
```



```java
package OOAD_HW2;

import java.util.Scanner;

public final class WelcomePage {
    static Scanner sc = new Scanner(System.in);

    WelcomePage() {
    }

    void menuControl(Taxi taxi, User user, Driver driver) {
        try {
            String userName = JOptionPane.showInputDialog(null, "Name:");
            user.setName(userName);
            String userSName = JOptionPane.showInputDialog(null, "Surname:");
            user.setSurname(userSName);
            String num = JOptionPane.showInputDialog(null, "Mobile Num:");
            user.setMobileNum(num);
            String address = JOptionPane.showInputDialog(null, "Current Location:");
            user.setCurrentLocation(address);

        } catch (Exception e) {
            e.getMessage();
        }
    }
}
```



```java
package OOAD_HW2;

public class User {
    private String name, surname, mobileNum, currentLocation, paymentMethod;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSurname() {
        return surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getMobileNum() {
        return mobileNum;
    }

    public void setMobileNum(String mobileNum) {
        this.mobileNum = mobileNum;
    }

    public String getCurrentLocation() {
        return currentLocation;
    }

    public void setCurrentLocation(String currentLocation) {
        this.currentLocation = currentLocation;
    }

    public User() {
        // TODO Auto-generated constructor stub
    }

    public User(String name, String surname, String mobileNum, String currentLocation, String paymentMethod) {
        this.mobileNum = mobileNum;
        this.currentLocation = currentLocation;
        this.name = name;
        this.surname = surname;
        this.paymentMethod = paymentMethod;
    }
```

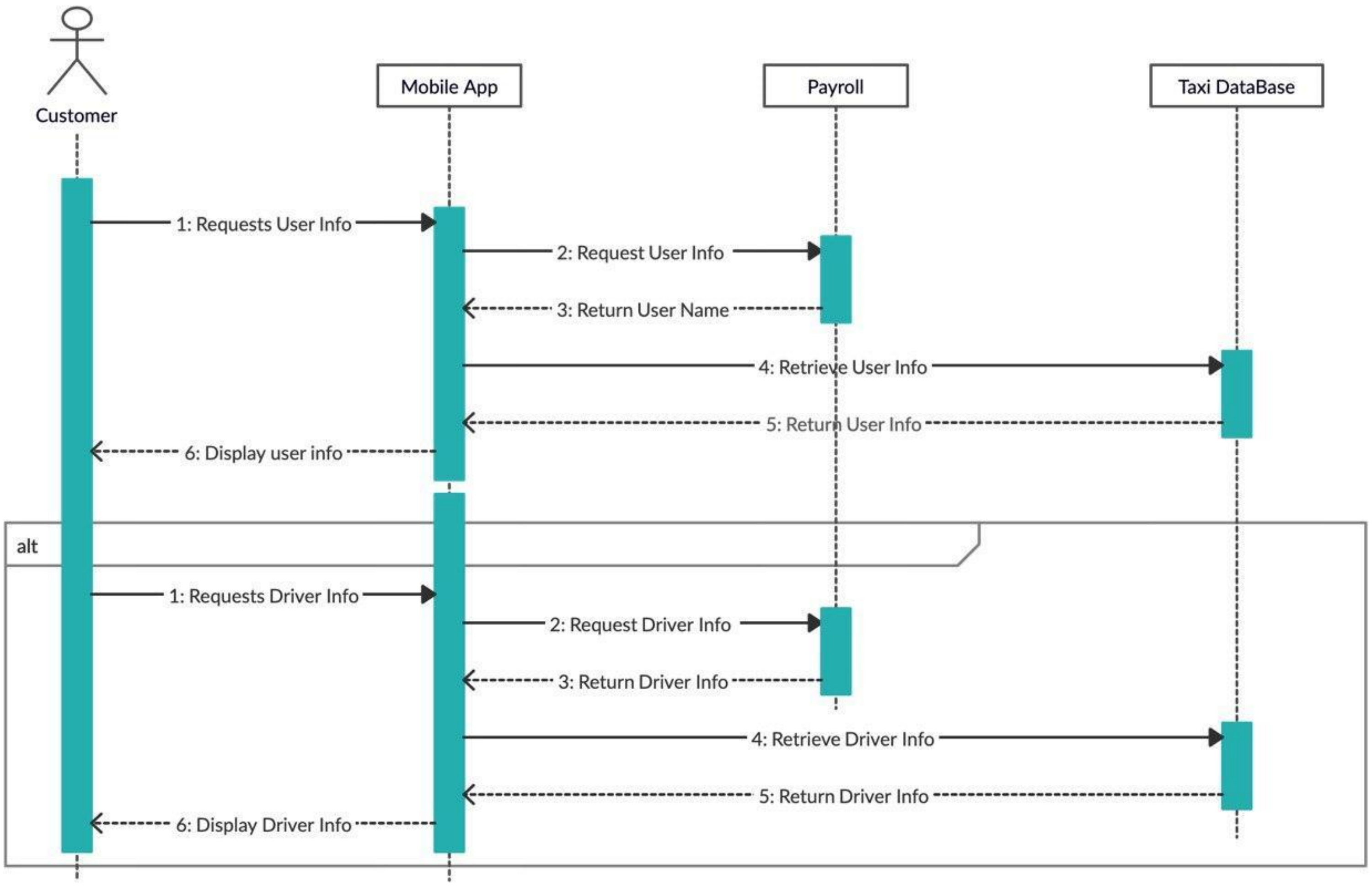


```java
    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getMobileNum() {
        return mobileNum;
    }

    public void setMobileNum(String mobileNum) {
        this.mobileNum = mobileNum;
    }

    public String getCurrentLocation() {
        return currentLocation;
    }

    public void setCurrentLocation(String currentLocation) {
        this.currentLocation = currentLocation;
    }

    public User() {
        // TODO Auto-generated constructor stub
    }

    public User(String name, String surname, String mobileNum, String currentLocation, String paymentMethod) {
        this.mobileNum = mobileNum;
        this.currentLocation = currentLocation;
        this.name = name;
        this.surname = surname;
        this.paymentMethod = paymentMethod;
    }
```

```java
    @Override
    public String toString() {
        return ("\nName: " + name + " " + surname + "\nMobile No: " + mobileNum + "\nCurrentLocation: "
                + currentLocation + "\n**********************************************************");
    }

    public String getPaymentMethod() {
        return paymentMethod;
    }

    public void setPaymentMethod(String paymentMethod) {
        this.paymentMethod = paymentMethod;
    }
```

Sequence Diagram

(for booking system)



Conclusion

This document included in itself a brief system analysis of a Taxi Dispatch Application. Apart from the diagrams, requirements and use cases provided, a demo also exists to show some of the functional parts of this application.