Final project

Car Manager

The project done by
Tsimur Halkin, Nerike Bosch, Kinga Mazur

The code available on
GitLab

***
Code discussion

Task of the project

The main purpose of the current project is the development of a car rental system that allows booking of vehicles.

Used Libraries

- Jackson
- Gson
- JavaFx

Main classes

First of all, there was created a common class with the features that all vehicles have. All vehicles include

```
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import com.fasterxml.jackson.annotation.JsonSubTypes;
import com.fasterxml.jackson.annotation.JsonTypeInfo;


// This annotation is used to indicate that this class can be serialized/deserialized with type information.
// The `use` attribute specifies that the type information will be included as a property in the JSON.
// The `property` attribute specifies the name of the property that will hold the type information.
@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        property = "type"
)
// This annotation specifies the possible subtypes of the class and their corresponding type names.
```

```java
// It is used to help Jackson know which class to instantiate when deserializing JSON objects.
@JsonSubTypes({
        @JsonSubTypes.Type(value = BEVCar.class, name = "BEVCar"),
        @JsonSubTypes.Type(value = Camper.class, name = "Camper"),
        @JsonSubTypes.Type(value = PickupTruck.class, name = "PickupTruck"),
        @JsonSubTypes.Type(value = Motorcycle.class, name = "Motorcycle"),
        @JsonSubTypes.Type(value = ICECar.class, name = "ICECar"),
        @JsonSubTypes.Type(value = HybridCar.class, name = "HybridCar"),
        @JsonSubTypes.Type(value = Car.class, name = "Car")
})

public class Vehicle {
    //
    private String model;
    private int year;
    private String color;
    private int passengers;
    private int price;
    private boolean status;
    private List<Reservation> reservations;
    // added boolean property bookingStatus in order to define booking status of vehicle


    // No-argument constructor
    public Vehicle() {
        //this.reservations = new ArrayList<>();
    }

     // This annotation is used to indicate that this constructor should be used when deserializing JSON data.
    @JsonCreator
    public Vehicle(@JsonProperty("model") String model,
                   @JsonProperty("year") int year,
                   @JsonProperty("color") String color,
                   @JsonProperty("passengers") int passengers,
                   @JsonProperty("status") boolean status,
                   @JsonProperty("price") int price) {
        this.model = model;
        this.year = year;
        this.color = color;
        this.passengers = passengers;
        this.status = status;
        this.price = price;
        this.reservations = new ArrayList<>();
    }

    public void addReservation(Reservation reservation){
        reservations.add(reservation);
    }

    public boolean isStatus() {
```

```
        return status;
    }

    public void setStatus(boolean status) {
        this.status = status;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public int getPassengers() {
        return passengers;
    }

    public void setPassengers(int passengers) {
        this.passengers = passengers;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

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

After that, different classes were developed that inherited the *vehicle class* . Every subclass includes properties that are common for *vehicle class* such as *colour*, *year*, *model ect.* and also has its own features depended on vehicle type (for example, *sleepingCapacity* for *Camper* or *fuelType* for *Car*)

BEV car subclass

```
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class BEVCar extends Vehicle {
    private double batteryCapacity;
    private double range;

    @JsonCreator
    public BEVCar(
            @JsonProperty("model") String model,
            @JsonProperty("year") int year,
            @JsonProperty("color") String color,
            @JsonProperty("passengers") int passengers,
            @JsonProperty("price") int price,
            @JsonProperty("status") boolean status,
            @JsonProperty("batteryCapacity") double batteryCapacity,
            @JsonProperty("range") double range
    ) {
        super(model, year, color, passengers, status, price);
        this.batteryCapacity = batteryCapacity;
        this.range = range;
    }

    public double getBatteryCapacity() {
        return batteryCapacity;
    }

    public void setBatteryCapacity(double batteryCapacity) {
        this.batteryCapacity = batteryCapacity;
    }

    public double getRange() {
        return range;
    }

    public void setRange(double range) {
        this.range = range;
    }
}
```

Camper subclass

```java
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Camper extends Vehicle {

    private int sleepingCapacity;

    @JsonCreator
    public Camper(@JsonProperty("model") String model,
                  @JsonProperty("year") int year,
                  @JsonProperty("color") String color,
                  @JsonProperty("passengers") int passengers,
                  @JsonProperty("status") boolean status,
                  @JsonProperty("price") int price,
                  @JsonProperty("sleepingCapacity") int sleepingCapacity) {
        super(model, year, color, passengers, status, price);
        this.sleepingCapacity = sleepingCapacity;
    }


    public int getSleepingCapacity() {
        return sleepingCapacity;
    }

    public void setSleepingCapacity(int sleepingCapacity) {
        this.sleepingCapacity = sleepingCapacity;
    }
}
```

Car subclass

```java
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Car extends Vehicle {

    private String fuelType;
    @JsonCreator
    public Car(@JsonProperty("model") String model,
               @JsonProperty("year") int year,
               @JsonProperty("color") String color,
               @JsonProperty("passengers") int passengers,
               @JsonProperty("status") boolean status,
               @JsonProperty("price") int price,
               @JsonProperty("fuelType") String fuelType) {
        super(model, year, color, passengers, status, price);
        this.fuelType = fuelType;
```

```
    }

    public String getFuelType() {
        return fuelType;
    }

    public void setFuelType(String fuelType) {
        this.fuelType = fuelType;
    }
}
```

Hybrid car subclass

```
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class HybridCar extends Vehicle {
    private double engineSize;
    private String fuelType;
    private double electricRange;

    @JsonCreator
    public HybridCar(@JsonProperty("model") String model,
                     @JsonProperty("year") int year,
                     @JsonProperty("color") String color,
                     @JsonProperty("passengers") int passengers,
                     @JsonProperty("status") boolean status,
                     @JsonProperty("price") int price,
                     @JsonProperty("engineSize") double engineSize,
                     @JsonProperty("fuelType") String fuelType,
                     @JsonProperty("electricRange") double electricRange) {
        super(model, year, color, passengers, status, price);
        this.engineSize = engineSize;
        this.fuelType = fuelType;
        this.electricRange = electricRange;
    }

    public double getEngineSize() {
        return engineSize;
    }

    public void setEngineSize(double engineSize) {
        this.engineSize = engineSize;
    }

    public String getFuelType() {
        return fuelType;
    }

    public void setFuelType(String fuelType) {
        this.fuelType = fuelType;
```

```
    }

    public double getElectricRange() {
        return electricRange;
    }

    public void setElectricRange(double electricRange) {
        this.electricRange = electricRange;
    }
}
```

ICE car subclass

```
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class ICECar extends Vehicle {

    private double engineSize;
    private String range;

    @JsonCreator
    public ICECar(@JsonProperty("model") String model,
                  @JsonProperty("year") int year,
                  @JsonProperty("color") String color,
                  @JsonProperty("passengers") int passengers,
                  @JsonProperty("status") boolean status,
                  @JsonProperty("price") int price,
                  @JsonProperty("engineSize") double engineSize,
                  @JsonProperty("range") String range) {
        super(model, year, color, passengers, status, price);
        this.engineSize = engineSize;
        this.range = range;
    }

    public double getEngineSize() {
        return engineSize;
    }

    public void setEngineSize(double engineSize) {
        this.engineSize = engineSize;
    }

    public String getRange() {
        return range;
    }

    public void setRange(String range) {
        this.range = range;
    }
}
```

Motorcycle

```
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Motorcycle extends Vehicle {
   private String motorcycleType;

   @JsonCreator
   public Motorcycle(@JsonProperty("model") String model,
                     @JsonProperty("year") int year,
                     @JsonProperty("color") String color,
                     @JsonProperty("passengers") int passengers,
                     @JsonProperty("status") boolean status,
                     @JsonProperty("price") int price,
                     @JsonProperty("motorcycleType") String motorcycleType) {
       super(model, year, color, passengers, status, price);
       this.motorcycleType = motorcycleType;
   }

   public String getMotorcycleType() {
       return motorcycleType;
   }

   public void setMotorcycleType(String motorcycleType) {
       this.motorcycleType = motorcycleType;
   }
}
```

Pickup Truck

```
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class PickupTruck extends Vehicle {

   private double payloadCapacity;

   @JsonCreator
   public PickupTruck(@JsonProperty("model") String model,
                      @JsonProperty("year") int year,
                      @JsonProperty("color") String color,
                      @JsonProperty("passengers") int passengers,
                      @JsonProperty("status") boolean status,
                      @JsonProperty("price") int price,
                                  @JsonProperty("payloadCapacity")  double
payloadCapacity) {
       super(model, year, color, passengers, status, price);
       this.payloadCapacity = payloadCapacity;
```

```
    }

    public double getPayloadCapacity() {
        return payloadCapacity;
    }

    public void setPayloadCapacity(double payloadCapacity) {
        this.payloadCapacity = payloadCapacity;
    }
}
```

Customer class

This class was developed in order to add new customers.

```
package com.example.carmanager;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Customer {
    private String name;
    private String surname;
    private String ID;
    private String email;
    private String phoneNumber;

    // No-argument constructor
    public Customer() {}

    @JsonCreator
    public Customer(@JsonProperty("name") String name,
                    @JsonProperty("surname") String surname,
                    @JsonProperty("ID") String ID,
                    @JsonProperty("email") String email,
                    @JsonProperty("phoneNumber") String phone) {
        this.name = name;
        this.surname = surname;
        this.ID = ID;
        this.email = email;
        this.phoneNumber = phone;
    }

    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 getID() {
        return ID;
    }

    public void setID(String ID) {
        this.ID = ID;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phoneNumber;
    }

    public void setPhone(String phone) {
        this.phoneNumber = phone;
    }
}
```

Work with data

The project includes a few classes in order to add info about customers and status of booking for a particular vehicle. The data about vehicles is stored in the folder *Data* in the *.json* format .

Data Manager

The class is responsible for addition of new data to the *json* file, parsing vehicle's data and editing of the data. For parsing the following libraries: *Jackson* and *Gson*.

```
package com.example.carmanager;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
import
com.fasterxml.jackson.databind.jsontype.BasicPolymorphicTypeValidator;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;


import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class DataManager {

   private final String VehiclesDB = "Data/VehicleDB.json";
   private final String CustomerDB = "Data/CustomerDB.json";
   private final String ReservationDB = "Data/ReservationDB.json";
   private ObjectMapper objectMapper;

   public DataManager() {
       // Configure object mapper for vehicles with polymorphic type handling
                                   BasicPolymorphicTypeValidator     ptv     =
BasicPolymorphicTypeValidator.builder()
               .allowIfBaseType(Vehicle.class)
               .build();
       this.objectMapper = new ObjectMapper();
                                  this.objectMapper.activateDefaultTyping(ptv,
ObjectMapper.DefaultTyping.NON_FINAL);

   }

   //Vehicle Parser
       public   List<Vehicle>   sortallVehicles(ObjectMapper   mapper)   throws
IOException {
       JsonNode node = mapper.readTree(new File(VehiclesDB));
       List<Vehicle> vehicles = new ArrayList<>();
       if (node.isArray()) {
           for (JsonNode jsonNode : node) {
               Vehicle vehicle = mapper.treeToValue(jsonNode, Vehicle.class);
               vehicles.add(vehicle);
           }
       }
       return vehicles;
   }


   // Vehicles sorting by type
   //BEVCars
   public List<BEVCar> sortBevCars(List<Vehicle> vehicles) {
```

```
    List<BEVCar> bevCars = vehicles.stream()
            .filter(BEVCar.class::isInstance)
            .map(BEVCar.class::cast)
            .collect(Collectors.toList());
    return bevCars;
}

//Camper
public List<Camper> sortCampers(List<Vehicle> vehicles) {
    List<Camper> campers = vehicles.stream()
            .filter(Camper.class::isInstance)
            .map(Camper.class::cast)
            .collect(Collectors.toList());
    return campers;
}

// Cars
public List<Camper> sortCars(List<Vehicle> vehicles) {
    List<Camper> campers = vehicles.stream()
            .filter(Camper.class::isInstance)
            .map(Camper.class::cast)
            .collect(Collectors.toList());
    return campers;
}


//Motorcycles
public List<Motorcycle> sortMotorcycles(List<Vehicle> vehicles) {
    List<Motorcycle> motorcycles = vehicles.stream()
            .filter(Motorcycle.class::isInstance)
            .map(Motorcycle.class::cast)
            .collect(Collectors.toList());
    return motorcycles;
}

//Hybrid cars
public List<HybridCar> sortHybridCars(List<Vehicle> vehicles) {
    List<HybridCar> hybridCars = vehicles.stream()
            .filter(HybridCar.class::isInstance)
            .map(HybridCar.class::cast)
            .collect(Collectors.toList());
    return hybridCars;
}

//ICECars
public List<ICECar> sortIceCars(List<Vehicle> vehicles) {
    List<ICECar> iceCars = vehicles.stream()
            .filter(ICECar.class::isInstance)
            .map(ICECar.class::cast)
            .collect(Collectors.toList());
    return iceCars;
}
```

```
    //PickupTracks
    public List<PickupTruck> sortPickupTrucks(List<Vehicle> vehicles) {
        List<PickupTruck> pickupTrucks = vehicles.stream()
                .filter(PickupTruck.class::isInstance)
                .map(PickupTruck.class::cast)
                .collect(Collectors.toList());
        return pickupTrucks;
    }

    // Consumers Parser
    //error
    public List<Customer> parseCustomers() {
        try (FileReader fr = new FileReader(CustomerDB)) {
            Gson gson = new Gson();
            Type listType = new TypeToken<List<Customer>>() {
            }.getType();
            List<Customer> customers = gson.fromJson(fr, listType);
            return customers;
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }

    }

    // Method to add info about Customer into json file
    public void addCustInfo(Customer newCustomer) {
        try {
            Gson gson = new Gson();
            // Read existing customers from the JSON file
            FileReader reader = new FileReader(CustomerDB);
            Type customerListType = new TypeToken<List<Customer>>() {
            }.getType();
                            List<Customer>  customers  =  gson.fromJson(reader,
customerListType);
            reader.close();

            // Add the new customer to the list
            customers.add(newCustomer);

            // Write the updated list back to the JSON file
            FileWriter writer = new FileWriter(CustomerDB);
            gson.toJson(customers, writer);
            writer.close();

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

    // Method to add info about reservation into json file
    public void addReservationInfo(Reservation newReservation) {
```

```
        try {
            Gson gson = new Gson();
            // Read existing reservation from the JSON file
            FileReader reader = new FileReader(ReservationDB);
            Type reservationListType = new TypeToken<List<Reservation>>() {
            }.getType();
                    List<Reservation>  reservations  =  gson.fromJson(reader,
reservationListType);
            reader.close();
            // Add the new customer to the list
            reservations.add(newReservation);

            // Write the updated list back to the JSON file
            FileWriter writer = new FileWriter(ReservationDB);
            gson.toJson(reservations, writer);
            writer.close();

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

    //Method to book particular Vehicle
            public    void    bookVehicle(ObjectMapper    mapper,List<Vehicle>
vehicles,Vehicle bookedVehicle) throws IOException {
        for (Vehicle vehicle : vehicles) {
            if (bookedVehicle.getModel().equals(vehicle.getModel()) ) {
                vehicle.setStatus(false);
                System.out.println("Vehicle status updated to false.");
                break;
            }
        }


        ArrayNode arrayNode = mapper.createArrayNode();
        for (Vehicle vehicle : vehicles) {
            JsonNode vehicleNode = mapper.valueToTree(vehicle);
            arrayNode.add(vehicleNode);
        }
        mapper.writeValue(new File(VehiclesDB), arrayNode);
        System.out.println("Vehicle data saved to file.");
    }

    }
```

Reservation

The class is used to unite info customers and chosen vehicles. All changes are saved in the corresponding file in the folder *Data*.

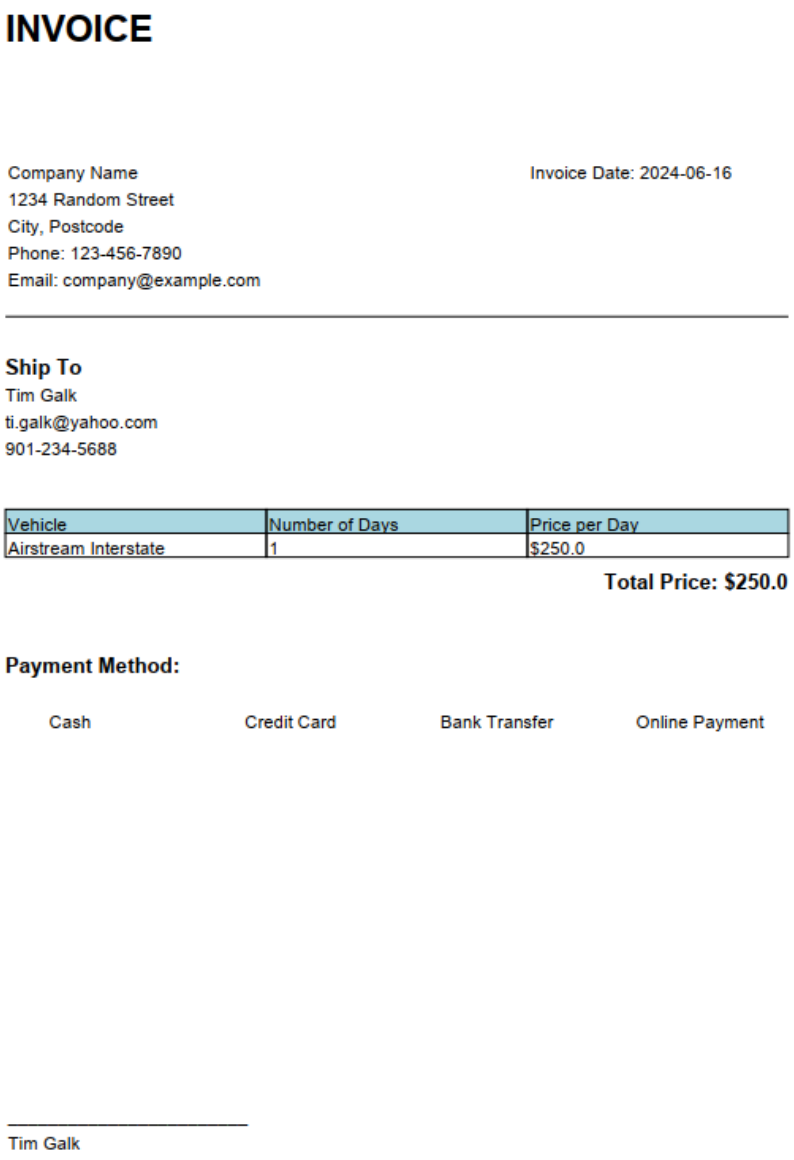
```
package com.example.carmanager;
```

```java
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

public class Reservation {
    private Vehicle vehicle;
    private Customer customer;
    private String startDate;
    private String endDate;

    // No-argument constructor
    public Reservation() {}

    @JsonCreator
    public Reservation(@JsonProperty("vehicle") Vehicle vehicle,
                       @JsonProperty("customer") Customer customer,
                       @JsonProperty("startDate") String startDate,
                       @JsonProperty("endDate") String endDate) {
        this.vehicle = vehicle;
        this.customer = customer;
        this.startDate = startDate;
        this.endDate = endDate;
    }

    public double calculateTotalPrice() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date start = sdf.parse(startDate);
            Date end = sdf.parse(endDate);
            long duration = end.getTime() - start.getTime();
                        long numberOfDays = TimeUnit.DAYS.convert(duration,
TimeUnit.MILLISECONDS);
            return (numberOfDays * vehicle.getPrice());
        } catch (ParseException e) {
            e.printStackTrace();
                    return 0; // In case of an error, return 0 or handle
appropriately
        }
    }

    public Vehicle getVehicle() {
        return vehicle;
    }

    public void setVehicle(Vehicle vehicle) {
        this.vehicle = vehicle;
    }
```

```
    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public String getStartDate() {
        return startDate;
    }

    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }

    public String getEndDate() {
        return endDate;
    }

    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }
}
```

PDF Generator of invoices

The invoice with all details of order will be generated after the process of booking.



INVOICE

Company Name
1234 Random Street
City, Postcode
Phone: 123-456-7890
Email: company@example.com

Invoice Date: 2024-06-16

Ship To
Tim Galk
ti.galk@yahoo.com
901-234-5688

Vehicle	Number of Days	Price per Day
Airstream Interstate	1	$250.0

Total Price: $250.0

Payment Method:

Cash Credit Card Bank Transfer Online Payment

Tim Galk

Example of generated invoice

The following code is responsible for this process :

```
package com.example.carmanager;

import com.itextpdf.text.*;
import com.itextpdf.text.pdf.*;
import com.itextpdf.text.pdf.draw.LineSeparator;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.stream.Stream;
```

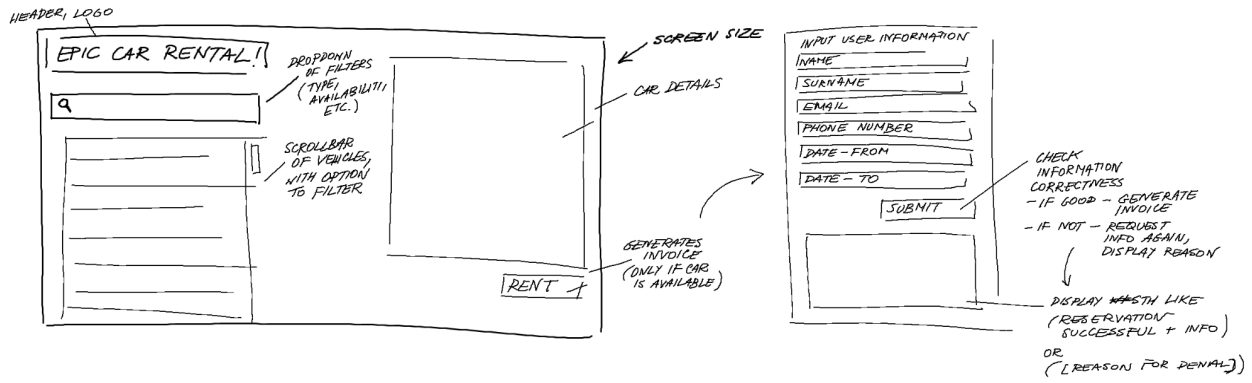
```
public class PDFInvoiceGenerator {
   private Document document;
   private String pdfName;
   private Reservation reservation;
   private Font titleFont;
   private Font boldFont;
   private Font normalFont;

   public PDFInvoiceGenerator(String pdfName, Reservation reservation) {
        this.pdfName = pdfName;
        this.reservation = reservation;
        titleFont = FontFactory.getFont(FontFactory.HELVETICA_BOLD, 24,
BaseColor.BLACK);
        boldFont = FontFactory.getFont(FontFactory.HELVETICA_BOLD, 14,
BaseColor.BLACK);
        normalFont = FontFactory.getFont(FontFactory.HELVETICA, 12,
BaseColor.BLACK);
   }

   public void createDocument() throws FileNotFoundException,
DocumentException {
        document = new Document();
        PdfWriter.getInstance(document, new FileOutputStream(pdfName));
        document.open();
   }

   public void addContent() throws DocumentException {
        // Add title
        Chunk titleChunk = new Chunk("INVOICE", titleFont);
        document.add(titleChunk);

        // Adding some space before the company details
        document.add(new Paragraph("\n\n\n\n"));

        // Create a table for company details and invoice date
        PdfPTable companyTable = new PdfPTable(new float[]{2, 1});
        companyTable.setWidthPercentage(100);

        // Company details
        PdfPCell companyDetailsCell = new PdfPCell();
        companyDetailsCell.setBorder(PdfPCell.NO_BORDER);
        Paragraph companyDetails = new Paragraph(
                "Company Name\n" +
                        "1234 Random Street\n" +
                        "City, Postcode\n" +
                        "Phone: 123-456-7890\n" +
                        "Email: company@example.com", normalFont);
        companyDetailsCell.addElement(companyDetails);
        companyTable.addCell(companyDetailsCell);

        // Invoice date
        PdfPCell invoiceDateCell = new PdfPCell();
        invoiceDateCell.setBorder(PdfPCell.NO_BORDER);
```

```
        invoiceDateCell.setHorizontalAlignment(Element.ALIGN_RIGHT);
        LocalDate invoiceDate = LocalDate.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd");
        Paragraph invoiceDateParagraph = new Paragraph("Invoice Date: " +
invoiceDate.format(formatter), normalFont);
        invoiceDateCell.addElement(invoiceDateParagraph);
        companyTable.addCell(invoiceDateCell);

        document.add(companyTable);

        // Add a line separator
        document.add(new Paragraph("\n"));
        document.add(new LineSeparator());

        // Add some space before the "Ship To" section
        document.add(new Paragraph("\n"));

        // Add "Ship To" section
        Paragraph shipToParagraph = new Paragraph("Ship To", boldFont);
        document.add(shipToParagraph);

        Customer customer = reservation.getCustomer();
        Paragraph customerDetails = new Paragraph(
                customer.getName() + " " + customer.getSurname() + "\n" +
                        customer.getEmail() + "\n" +
                        customer.getPhone(), normalFont);
        document.add(customerDetails);

        // Adding some space before the table
        document.add(new Paragraph("\n\n"));

        PdfPTable table = new PdfPTable(3);  // Changed to 3 columns
        table.setWidthPercentage(100);
        addTableHeader(table);
        addRows(table);
        document.add(table);

        // Add total price below the table
        double pricePerDay = reservation.getVehicle().getPrice();
        long numberOfDays = calculateNumberOfDays(reservation.getStartDate(),
reservation.getEndDate());
        double totalSum = pricePerDay * numberOfDays;
        Paragraph totalParagraph = new Paragraph("Total Price: $" + totalSum,
boldFont);
        totalParagraph.setAlignment(Element.ALIGN_RIGHT);
        document.add(totalParagraph);

        // Adding payment methods
        document.add(new Paragraph("\n\n"));
        Paragraph paymentMethodParagraph = new Paragraph("Payment
Method:\n\n", boldFont);
        document.add(paymentMethodParagraph);
```

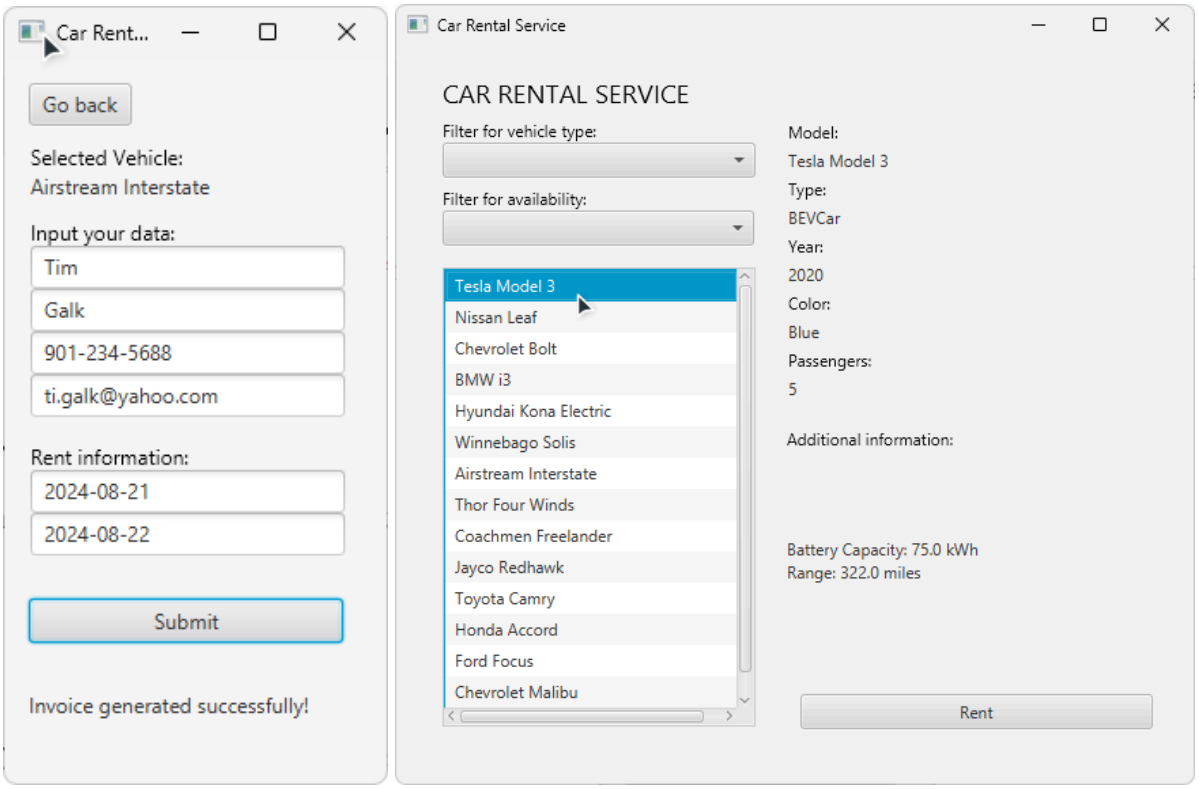
```
        PdfPTable paymentMethodTable = new PdfPTable(new float[]{1, 1, 1,
1});
        paymentMethodTable.setWidthPercentage(100);

        addCheckbox(paymentMethodTable, "Cash");
        addCheckbox(paymentMethodTable, "Credit Card");
        addCheckbox(paymentMethodTable, "Bank Transfer");
        addCheckbox(paymentMethodTable, "Online Payment");

        document.add(paymentMethodTable);

        // Adding space after payment methods
        document.add(new Paragraph("\n\n"));

        // Add signature line at the bottom right
        PdfPTable signatureTable = new PdfPTable(1);
        signatureTable.setWidthPercentage(100);
        signatureTable.setHorizontalAlignment(Element.ALIGN_RIGHT);

        PdfPCell signatureCell = new PdfPCell();
        signatureCell.setBorder(PdfPCell.NO_BORDER);
        signatureCell.setFixedHeight(50);
        signatureCell.setHorizontalAlignment(Element.ALIGN_RIGHT);
        Paragraph signatureLine = new Paragraph("_____",
normalFont);
        Paragraph signatureName = new Paragraph(customer.getName() + " " +
customer.getSurname(), normalFont);
        signatureCell.addElement(signatureLine);
        signatureCell.addElement(signatureName);
        signatureTable.addCell(signatureCell);



        // Move signature section to the bottom of the page
        document.add(new Paragraph("\n\n\n\n\n\n\n\n\n\n\n"));
        document.add(signatureTable);
    }

    private void addTableHeader(PdfPTable table) {
        Stream.of("Vehicle", "Number of Days", "Price per Day")  // Changed
column headers
                .forEach(columnTitle -> {
                    PdfPCell header = new PdfPCell();
                    header.setBackgroundColor(new BaseColor(173, 216, 230));
// Light blue background
                    header.setBorderWidth(1);
                    header.setPhrase(new Phrase(columnTitle,
FontFactory.getFont(FontFactory.HELVETICA, 12, BaseColor.BLACK)));  // Black
font color
                    table.addCell(header);
                });
    }
```

```
    private void addRows(PdfPTable table) {
        Vehicle vehicle = reservation.getVehicle();
        double pricePerDay = vehicle.getPrice();
        long numberOfDays = calculateNumberOfDays(reservation.getStartDate(),
reservation.getEndDate());

        table.addCell(vehicle.getModel());
        table.addCell(String.valueOf(numberOfDays));
        table.addCell("$" + pricePerDay);
    }

    private void addCheckbox(PdfPTable table, String label) {
        PdfPCell checkboxCell = new PdfPCell();
        checkboxCell.setBorder(PdfPCell.NO_BORDER);
        PdfPCell labelCell = new PdfPCell(new Phrase(label));
        labelCell.setBorder(PdfPCell.NO_BORDER);

        PdfPTable innerTable = new PdfPTable(new float[]{1, 4});
        innerTable.setWidthPercentage(100);

        PdfPCell cbCell = new PdfPCell();
        cbCell.setBorder(PdfPCell.NO_BORDER);
        cbCell.addElement(new Chunk("\u25A1",
FontFactory.getFont(FontFactory.HELVETICA, 12)));  // Unicode for empty
checkbox

        innerTable.addCell(cbCell);
        innerTable.addCell(labelCell);

        checkboxCell.addElement(innerTable);
        table.addCell(checkboxCell);
    }

    public void closeDocument() {
        document.close();
    }

    private long calculateNumberOfDays(String startDate, String endDate) {
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd");
        LocalDate start = LocalDate.parse(startDate, formatter);
        LocalDate end = LocalDate.parse(endDate, formatter);
        return java.time.temporal.ChronoUnit.DAYS.between(start, end);
    }

}
```

UI and Interface implementation



Concept art of UI



Developed interface

Vehicle selection controller

```
package com.example.carmanager;

import com.fasterxml.jackson.databind.ObjectMapper;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
```

```
import javafx.collections.transformation.FilteredList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

public class VehicleSelectionController implements Initializable {
   private Stage stage;
   private Scene scene;
   private Parent root;
   private Vehicle selectedVehicle;
     private String[] vehicleTypes = {"All", "BEVCar", "Camper", "Car",
"HybridCar","ICECar", "Motorcycle", "PickupTruck"};
        private String[] availabilityOptions = {"All","Available","Not
Available"};
   private FilteredList filteredList;
   private ObservableList<Vehicle> vehicleList;
   @FXML
   private AnchorPane anchorPane_scene1;

   @FXML
   private Button button_rent;

   @FXML
   private ChoiceBox<String> choiceBox_availability;

   @FXML
   private ChoiceBox<String> choiceBox_vehicleType;

   @FXML
   private Label label_color;

   @FXML
   private Label label_model;

   @FXML
   private Label label_passengers;

   @FXML
   private Label label_type;
```

```
    @FXML
    private Label label_year;

    @FXML
    public Label label_additionalInformation;

    @FXML
    private ListView<Vehicle> listView_vehicleList;

    @FXML
    private VBox vbox_additionalInformation;


    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        ObjectMapper mapper = new ObjectMapper();
        DataManager dataManager = new DataManager();
        // Get the list of all vehicles
        List<Vehicle> exampleList = null;
        try {
            exampleList = dataManager.sortallVehicles(mapper);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        // Initialize the observable list with the vehicle list
        vehicleList = FXCollections.observableArrayList(exampleList);

        // Add vehicle types to the choice box
        choiceBox_vehicleType.getItems().addAll(vehicleTypes);
        choiceBox_vehicleType.setOnAction(event -> {
            try {
                filterList(event); // Filter the list based on selected type
            } catch (ClassNotFoundException e) {
                throw new RuntimeException(e);
            }
        });

        // Add availability types to the choice box
        choiceBox_availability.getItems().addAll(availabilityOptions);
        choiceBox_availability.setOnAction(event -> {

filterListByAvailability(event,filteredList,choiceBox_availability);        //
Filter the list based on selected availability
        });

        // Initialize filtered list with all vehicles
        filteredList = new FilteredList<>(vehicleList, p -> true);
        listView_vehicleList.setItems(filteredList);

        // Add listener for selecting a vehicle from the list
```

```
listView_vehicleList.getSelectionModel().selectedItemProperty().addListener(n
ew ChangeListener<Vehicle>() {
            @Override
                    public  void  changed(ObservableValue<?  extends  Vehicle>
observableValue, Vehicle vehicle, Vehicle t1) {
                                                                selectedVehicle    =
listView_vehicleList.getSelectionModel().getSelectedItem();
                //displayDetails();
            }
        });
    }

                    public      void      filterListByAvailability(ActionEvent
event,FilteredList<Vehicle>                    filteredList,ChoiceBox<String>
choiceBox_availability) {
        String selectedAvailability = choiceBox_availability.getValue();

        filteredList.setPredicate(vehicle -> {
            if ("All".equals(selectedAvailability)) {
                return true; // Show all vehicles if "All" is selected
            } else if ("Available".equals(selectedAvailability)) {
                return vehicle.isStatus(); // Show only available vehicles
            } else {
                      return  !vehicle.isStatus();  //  Show  only  not  available
vehicles
            }
        });
    }

    // Method to filter the vehicle list based on selected type
    public void filterList(ActionEvent event) throws ClassNotFoundException {
        String selectedType = choiceBox_vehicleType.getValue();

        filteredList.setPredicate(vehicle -> {
            if ("All".equals(selectedType)) {
                return true; // Show all vehicles if "All" is selected
            }
            try {
                    Class<?> clas = Class.forName("com.example.carmanager." +
selectedType);
                return clas.isInstance(vehicle);
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
                return false;
            }
        });
    }

    // Method to display details of the selected vehicle
    @FXML
    public void displayDetails(){
```

```
        // this method should utilize the declared labels and properties of
the selected object
       // the selected vehicle is selectedVehicle
        // one label exists for individual properties of all extensions of
Vehicle - label_additionalInformation
       label_model.setText(selectedVehicle.getModel());
       label_type.setText(selectedVehicle.getClass().getSimpleName());
       label_color.setText(selectedVehicle.getColor());
       label_year.setText(String.valueOf(selectedVehicle.getYear()));

label_passengers.setText(String.valueOf(selectedVehicle.getPassengers()));

       // Display additional information based on the type of vehicle
       String additionalInfo = " ";
       if (selectedVehicle instanceof BEVCar) {
           BEVCar bevCar = (BEVCar) selectedVehicle;
                               additionalInfo = "Battery Capacity: " +
bevCar.getBatteryCapacity() + " kWh\nRange: " + bevCar.getRange() + " miles";
       } else if (selectedVehicle instanceof HybridCar) {
           HybridCar hybridCar = (HybridCar) selectedVehicle;
            additionalInfo = "Engine Size: " + hybridCar.getEngineSize() + "
L\nFuel Type: " + hybridCar.getFuelType() + "\nElectric Range: " +
hybridCar.getElectricRange() + " miles";
       } else if (selectedVehicle instanceof ICECar) {
           ICECar iceCar = (ICECar) selectedVehicle;
              additionalInfo = "Engine Size: " + iceCar.getEngineSize() + "
L\nEngine Size: " + iceCar.getEngineSize() + "\nRange: " + iceCar.getRange();
       } else if (selectedVehicle instanceof Camper) {
           Camper camper = (Camper) selectedVehicle;
                               additionalInfo = "Sleeping Capacity: " +
camper.getSleepingCapacity();
       } else if (selectedVehicle instanceof Motorcycle) {
           Motorcycle motorcycle = (Motorcycle) selectedVehicle;
                                additionalInfo = "Motorcycle Type: " +
motorcycle.getMotorcycleType();
       } else if (selectedVehicle instanceof PickupTruck) {
           PickupTruck pickupTruck = (PickupTruck) selectedVehicle;
                               additionalInfo = "Payload Capacity: " +
pickupTruck.getPayloadCapacity() + " lbs";
       }
       label_additionalInformation.setText(additionalInfo);
   }

   // Method to handle the rent button click event
   public void rent(ActionEvent event) throws IOException {
       if (selectedVehicle != null && !selectedVehicle.isStatus()) {
           // Show an alert if the selected vehicle is not available
           Alert alert = new Alert(Alert.AlertType.ERROR);
           alert.setTitle("Vehicle Not Available");
           alert.setHeaderText(null);
             alert.setContentText("The selected vehicle is not available for
rent. Please select a different vehicle.");
           alert.showAndWait();
```

```
        } else if (selectedVehicle != null) {
            // Proceed with renting the vehicle if it is available
                                                FXMLLoader loader = new
FXMLLoader(getClass().getResource("customerInput-view.fxml"));
            root = loader.load();
                                CustomerInputController customerInputController =
loader.getController();
            if (selectedVehicle != null) {
                customerInputController.displayVehicle(selectedVehicle);
                customerInputController.setVehicle(selectedVehicle);
                                                    stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();
                scene = new Scene(root);
                stage.setScene(scene);
                stage.show();
            }
        }
    }
}
```

Customer input controller

```
package com.example.carmanager;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.itextpdf.text.DocumentException;
import javafx.application.Platform;
import javafx.concurrent.Task;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;

import java.io.IOException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class CustomerInputController {
```

```
    private Stage stage;
    private Scene scene;
    private Parent root;
    private Vehicle selectedVehicle;
    private List<Reservation> reservations = new ArrayList<>();

    @FXML
    private Button button_goBack;

    @FXML
    private Button button_submit;

    @FXML
    private Label label_invalidSubmission;

    @FXML
    private Label label_selectedVehicle;

    @FXML
    private TextField textField_dateFrom;

    @FXML
    private TextField textField_dateTo;

    @FXML
    private TextField textField_email;

    @FXML
    private TextField textField_name;

    @FXML
    private TextField textField_phoneNumber;

    @FXML
    private TextField textField_surname;

                    private      DateTimeFormatter      dateFormatter      =
DateTimeFormatter.ofPattern("yyyy-MM-dd");

     //information provided in the text fields should be used to create a new
Customer object
    //check for availability of the car with its rentFrom and rentTo LocalDate
properties
      //if the car isn't available for rent, display a message in the
invalidSubmission label

    //vehicle selected in the display list in the previous scene is used here
    public void displayVehicle(Vehicle selectedVehicle){
        label_selectedVehicle.setText(selectedVehicle.getModel());
    }
    public void setVehicle(Vehicle vehicle){
        selectedVehicle = vehicle;
    }
```

```
    public void switchToScene1(ActionEvent event) throws IOException {
                                                                root          =
FXMLLoader.load(getClass().getResource("vehicleSelection-view.fxml"));
        stage = (Stage)((Node)event.getSource()).getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }

    public void btb_Submit(ActionEvent actionEvent) throws IOException {
        try {
             // Parse the dates from the text fields using the specified date
formatter
            LocalDate dateFrom = LocalDate.parse(textField_dateFrom.getText(),
dateFormatter);
              LocalDate dateTo = LocalDate.parse(textField_dateTo.getText(),
dateFormatter);

            // Ensure "date from" is not after "date to"
            if (dateFrom.isAfter(dateTo)) {
                label_invalidSubmission.setText("\"Date From\" cannot be after
\"Date To\".");
                return;
            }

             // Check if the selected vehicle is available for the specified
dates
            if (!isVehicleAvailable(dateFrom, dateTo)) {
                 label_invalidSubmission.setText("Vehicle is not available for
the selected dates.");
                return;
            }

            // Validate email format
            if (!isValidEmail(textField_email.getText())) {
                label_invalidSubmission.setText("Invalid email format.");
                return;
            }

            // Validate phone number format
            if (!isValidPhoneNumber(textField_phoneNumber.getText())) {
                 label_invalidSubmission.setText("Invalid phone number format.
\nPlease use 123-123-1234.");
                return;
            }

            // Generate a random ID for the customer
            Random random = new Random();
            int id = random.nextInt(999999);

            // Create a new Vehicle object using the selected vehicle
            Vehicle vehicle = selectedVehicle;
```

```
            // Create a new Customer object using the information from the
text fields and the generated ID
                Customer customer = new Customer(textField_name.getText(),
textField_surname.getText(),            String.format("%06d",            id),
textField_email.getText(), textField_phoneNumber.getText());

            // Create a new Reservation object using the vehicle, customer,
and dates
              Reservation reservation = new Reservation(vehicle, customer,
dateFrom.toString(), dateTo.toString());

            // Update Vehicle's database and mark selected vehicles as booked
            DataManager dataManager = new DataManager();
            ObjectMapper mapper = new ObjectMapper();
            List<Vehicle> vehicles = dataManager.sortallVehicles(mapper);

            dataManager.bookVehicle(mapper, vehicles, selectedVehicle);
              // Create an instance of DataManager to handle data storage and
add to customer database
            dataManager.addCustInfo(customer);

            // Add the new reservation and add to database
            reservations.add(reservation);
            dataManager.addReservationInfo(reservation);

            // Generate a PDF invoice for the reservation
               String filename = "Data/Invoice/" + customer.getName() + " " +
customer.getSurname() + " Invoice.pdf";
              PDFInvoiceGenerator invoice = new PDFInvoiceGenerator(filename,
reservation);
            invoice.createDocument();
            invoice.addContent();
            invoice.closeDocument();

            // Display a success message that the invoice is created
                            label_invalidSubmission.setText("Invoice  generated
successfully!");
        } catch (DateTimeParseException e) {
              label_invalidSubmission.setText("Invalid date format. Please use
yyyy-MM-dd.");
        } catch (DocumentException e) {
            throw new RuntimeException(e);
        }
    }


    private boolean isVehicleAvailable(LocalDate dateFrom, LocalDate dateTo)
{
        for (Reservation reservation : reservations) {
            if (reservation.getVehicle().equals(selectedVehicle)) {
                                                LocalDate   reservedFrom   =
LocalDate.parse(reservation.getStartDate());
```

```
                                                    LocalDate    reservedTo    =
LocalDate.parse(reservation.getEndDate());
                                        if   ((dateFrom.isBefore(reservedTo)   &&
dateTo.isAfter(reservedFrom))      ||     dateFrom.equals(reservedFrom)      ||
dateTo.equals(reservedTo)) {
                    return false;
                }
            }
        }
        return true;
    }

    private boolean isValidEmail(String email) {
        // Define the email pattern
        String emailPattern = "^[A-Za-z0-9+_.-]+@(.+)$";
        Pattern pattern = Pattern.compile(emailPattern);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }

    private boolean isValidPhoneNumber(String phoneNumber) {
        // Define the phone number pattern (123-123-1234 format)
        String phonePattern = "^\\d{3}-\\d{3}-\\d{4}$";
        Pattern pattern = Pattern.compile(phonePattern);
        Matcher matcher = pattern.matcher(phoneNumber);
        return matcher.matches();
    }
}
```