import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

/\*\*

\* Car class representing each car in the fleet.

\*/

class Car {

private String make;

private String model;

private int year;

private String licensePlate;

private double rentalPricePerDay;

private boolean isAvailable;

public Car(String make, String model, int year, String licensePlate, double rentalPricePerDay) {

this.make = make;

this.model = model;

this.year = year;

this.licensePlate = licensePlate;

this.rentalPricePerDay = rentalPricePerDay;

this.isAvailable = true;

}

public String getMake() { return make; }

public String getModel() { return model; }

public int getYear() { return year; }

public String getLicensePlate() { return licensePlate; }

public double getRentalPricePerDay() { return rentalPricePerDay; }

public boolean isAvailable() { return isAvailable; }

public void setAvailable(boolean available) { this.isAvailable = available; }

@Override

public String toString() {

return String.format("%s %s (%s) - Year: %d, Price/Day: %.2f, Available: %b",

make, model, licensePlate, year, rentalPricePerDay, isAvailable);

}

}

/\*\*

\* Customer class representing a customer.

\*/

class Customer {

private String customerID;

private String name;

private String email;

public Customer(String customerID, String name, String email) {

this.customerID = customerID;

this.name = name;

this.email = email;

}

public String getCustomerID() { return customerID; }

public String getName() { return name; }

public String getEmail() { return email; }

@Override

public String toString() {

return String.format("CustomerID: %s, Name: %s, Email: %s", customerID, name, email);

}

}

/\*\*

\* RentalTransaction class representing each rental record.

\*/

class RentalTransaction {

private String licensePlate;

private String customerID;

private int rentalDays;

private double totalCost;

public RentalTransaction(String licensePlate, String customerID, int rentalDays, double totalCost) {

this.licensePlate = licensePlate;

this.customerID = customerID;

this.rentalDays = rentalDays;

this.totalCost = totalCost;

}

public String getLicensePlate() { return licensePlate; }

public String getCustomerID() { return customerID; }

public int getRentalDays() { return rentalDays; }

public double getTotalCost() { return totalCost; }

@Override

public String toString() {

return String.format("License Plate: %s, CustomerID: %s, Days: %d, Total Cost: %.2f",

licensePlate, customerID, rentalDays, totalCost);

}

}

/\*\*

\* RentalAgency class managing cars, customers, and transactions.

\*/

class RentalAgency {

private String name;

private List<Car> cars;

private List<Customer> customers;

private List<RentalTransaction> transactions;

public RentalAgency(String name) {

this.name = name;

this.cars = new ArrayList<>();

this.customers = new ArrayList<>();

this.transactions = new ArrayList<>();

}

// Add a new car to the fleet

public void addCar(Car car) {

// Check for duplicate license plate

Optional<Car> existing = cars.stream()

.filter(c -> c.getLicensePlate().equals(car.getLicensePlate()))

.findFirst();

if (existing.isPresent()) {

System.out.println("Car with license plate " + car.getLicensePlate() + " already exists. Skipping addition.");

} else {

cars.add(car);

System.out.println("Added car: " + car);

}

}

// Remove a car from the fleet

public void removeCar(String licensePlate) {

Optional<Car> carOpt = cars.stream()

.filter(c -> c.getLicensePlate().equals(licensePlate))

.findFirst();

if (carOpt.isPresent()) {

Car car = carOpt.get();

if (car.isAvailable()) {

cars.remove(car);

System.out.println("Removed car with license plate: " + licensePlate);

} else {

System.out.println("Cannot remove car " + licensePlate + " as it is currently rented.");

}

} else {

System.out.println("Car with license plate " + licensePlate + " does not exist.");

}

}

// Register a new customer

public void registerCustomer(Customer customer) {

// Check for duplicate customer ID

boolean exists = customers.stream()

.anyMatch(c -> c.getCustomerID().equals(customer.getCustomerID()));

if (exists) {

System.out.println("Customer with ID " + customer.getCustomerID() + " already exists. Skipping registration.");

} else {

customers.add(customer);

System.out.println("Registered customer: " + customer);

}

}

// Get list of available cars

public List<Car> getAvailableCars() {

List<Car> available = new ArrayList<>();

for (Car c : cars) {

if (c.isAvailable()) {

available.add(c);

}

}

return available;

}

// Get list of all cars

public List<Car> getAllCars() {

return new ArrayList<>(cars);

}

// Find a car by license plate

private Optional<Car> findCar(String licensePlate) {

return cars.stream()

.filter(c -> c.getLicensePlate().equals(licensePlate))

.findFirst();

}

// Find a customer by ID

private Optional<Customer> findCustomer(String customerID) {

return customers.stream()

.filter(c -> c.getCustomerID().equals(customerID))

.findFirst();

}

// Rent a car

public Optional<RentalTransaction> rentCar(String licensePlate, String customerID, int rentalDays) {

if (rentalDays <= 0) {

System.out.println("Invalid rental duration: " + rentalDays);

return Optional.empty();

}

Optional<Car> carOpt = findCar(licensePlate);

Optional<Customer> customerOpt = findCustomer(customerID);

if (!carOpt.isPresent()) {

System.out.println("Car with license plate " + licensePlate + " not found.");

return Optional.empty();

}

if (!customerOpt.isPresent()) {

System.out.println("Customer with ID " + customerID + " not found.");

return Optional.empty();

}

Car car = carOpt.get();

if (!car.isAvailable()) {

System.out.println("Car " + licensePlate + " is currently not available.");

return Optional.empty();

}

// Rent the car

car.setAvailable(false);

double totalCost = rentalDays \* car.getRentalPricePerDay();

RentalTransaction transaction = new RentalTransaction(licensePlate, customerID, rentalDays, totalCost);

transactions.add(transaction);

System.out.println("Rental successful: " + transaction);

return Optional.of(transaction);

}

// Return a car

public void returnCar(String licensePlate) {

Optional<Car> carOpt = findCar(licensePlate);

if (!carOpt.isPresent()) {

System.out.println("Car with license plate " + licensePlate + " does not exist.");

return;

}

Car car = carOpt.get();

if (car.isAvailable()) {

System.out.println("Car " + licensePlate + " is already available. Cannot return.");

} else {

car.setAvailable(true);

System.out.println("Car " + licensePlate + " has been returned and is now available.");

}

}

// Get all transactions

public List<RentalTransaction> getAllTransactions() {

return new ArrayList<>(transactions);

}

}

/\*\*

\* Main class to demonstrate the Car Rental System.

\*/

public class Main {

public static void main(String[] args) {

System.out.println("--- Welcome to Halcion Motors Car Rental System ---");

// 1. Initialize the Rental Agency

RentalAgency halcionMotors = new RentalAgency("Halcion Motors");

// 2. Add Cars to the Fleet

System.out.println("\n--- Adding Cars ---");

Car car1 = new Car("Toyota", "Camry", 2022, "KDA-235U", 2000.00);

Car car2 = new Car("Honda", "Civic", 2021, "KDE-897G", 3000.00);

Car car3 = new Car("Ford", "Mustang", 2023, "KDS-333D", 7000.00);

Car car4 = new Car("Nissan", "Altima", 2020, "KDT-645G", 5000.00);

halcionMotors.addCar(car1);

halcionMotors.addCar(car2);

halcionMotors.addCar(car3);

halcionMotors.addCar(car4);

halcionMotors.addCar(new Car("Toyota", "Rav4", 2024, "KDU-869H", 2500.00)); // Duplicate license plate

// 3. Register Customers

System.out.println("\n--- Registering Customers ---");

Customer cust1 = new Customer("C001", "Alice Smith", "alice@gmail.com");

Customer cust2 = new Customer("C002", "Johnson Kasongo", "kasongo@gmail.com");

Customer cust3 = new Customer("C003", "Kioko Brown", "kioko@gmail.com");

halcionMotors.registerCustomer(cust1);

halcionMotors.registerCustomer(cust2);

halcionMotors.registerCustomer(cust3);

halcionMotors.registerCustomer(new Customer("C001", "Alice Wanjiru", "wanjiru@gmail.com")); // Duplicate customer ID

// 4. Display Available Cars

System.out.println("\n--- Available Cars ---");

List<Car> availableCars = halcionMotors.getAvailableCars();

if (availableCars.isEmpty()) {

System.out.println("No cars currently available.");

} else {

availableCars.forEach(car -> System.out.println("- " + car.getMake() + " " + car.getModel() + " (" + car.getLicensePlate() + ")"));

}

// 5. Perform Rental Transactions

System.out.println("\n--- Performing Rentals ---");

// Successful rental

Optional<RentalTransaction> rental1 = halcionMotors.rentCar("KDA-235U", "C001", 3);

rental1.ifPresent(System.out::println);

// Attempt to rent an already rented car

halcionMotors.rentCar("KDA-235U", "C002", 2);

// Successful rental with another customer and car

Optional<RentalTransaction> rental2 = halcionMotors.rentCar("KDE-897G", "C002", 5);

rental2.ifPresent(System.out::println);

// Attempt to rent non-existent car

halcionMotors.rentCar("KCB-234R", "C001", 1);

// Attempt to rent with non-existent customer

halcionMotors.rentCar("KAY-897T", "C004", 2);

// Attempt to rent with invalid duration

halcionMotors.rentCar("KDM-098D", "C003", 0);

// 6. Display Available Cars after rentals

System.out.println("\n--- Available Cars After Rentals ---");

availableCars = halcionMotors.getAvailableCars();

if (availableCars.isEmpty()) {

System.out.println("No cars currently available.");

} else {

availableCars.forEach(car -> System.out.println("- " + car.getMake() + " " + car.getModel() + " (" + car.getLicensePlate() + ")"));

}

// 7. Display All Transactions

System.out.println("\n--- All Rental Transactions ---");

List<RentalTransaction> allTransactions = halcionMotors.getAllTransactions();

if (allTransactions.isEmpty()) {

System.out.println("No transactions recorded.");

} else {

allTransactions.forEach(System.out::println);

}

// 8. Return Cars

System.out.println("\n--- Returning Cars ---");

halcionMotors.returnCar("KDA-235U"); // Successfully return car1

halcionMotors.returnCar("KDA-235U"); // Attempt to return already available car1

halcionMotors.returnCar("KCB-234R"); // Attempt to return non-existent car

halcionMotors.returnCar("KDS-333D"); // Attempt to return a car that was never rented

// 9. Display Available Cars after returns

System.out.println("\n--- Available Cars After Returns ---");

availableCars = halcionMotors.getAvailableCars();

if (availableCars.isEmpty()) {

System.out.println("No cars currently available.");

} else {

availableCars.forEach(car -> System.out.println("- " + car.getMake() + " " + car.getModel() + " (" + car.getLicensePlate() + ")"));

}

// 10. Demonstrate car removal

System.out.println("\n--- Demonstrating Car Removal ---");

halcionMotors.removeCar("KDS-333D"); // Remove an available car

halcionMotors.removeCar("KDA-235U"); // Remove a rented car

halcionMotors.removeCar("NON-EXISTENT"); // Remove non-existent car

System.out.println("\n--- Final All Cars List ---");

halcionMotors.getAllCars().forEach(System.out::println);

System.out.println("\n--- End of Halcion Motors Car Rental System Demo ---");

}

}