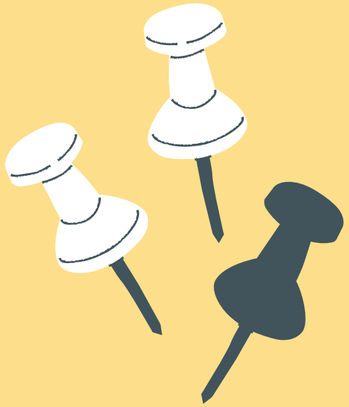

OOP

"Car Rental System"

KEY POINTS:

1-WHAT IS THE CAR RENTAL SYSTEM?
2-CORE FEATURES
3-SYETEM ARCHITECTURE
4-WHY THIS SYSTEM
5-SYSTEM WORKFLOW
6-DEEP IN CODE

WHAT IS THE CAR RENTAL SYSTEM?

- The Car Rental System is designed to streamline the process of renting and managing cars for different types of users (Admin, Employee, Customer).

- It provides an efficient way to handle car availability, rentals, returns, and active contracts.

CORE FEATURES:

- Car Management: Admins can add, remove, and manage cars in the system.
- Rental Process: Customers can rent available cars by selecting from a list of options.
- Return Process: Customers or employees can return rented cars and update their status.
- User Roles: The system supports multiple user roles (Admin, Employee, Customer) with specific functionalities for each role.

SYSTEM ARCHITECTURE:

- The system is divided into Models , Services , and Methods :
- Models: Represent the core entities like Car, Customer, Employee, and RentalContract.
- Services: Handle the main operations such as adding cars, renting, returning, and managing rentals (RentalService).
- Methods: Provide reusable functions for specific tasks like getting available cars or displaying active rentals.

WHY THIS SYSTEM

- Simplifies the car rental process for businesses.

- Ensures proper tracking of cars and rentals.

- Provides a user-friendly interface for different types of users.

SYSTEM WORKFLOW

- Users choose their role (Admin, Employee, Customer).

- Based on the role, they access specific functionalities.

- The system handles all operations through a centralized service (RentalService).

NOW LETS TALK IN DEEP ABOUT "THIS CODE "

1-Methods used

2-Models

3-services

4-Main bage

1- METHODS:

add_car

remove_car

rent_car

return_car

add_customer

remove_customer

manage_customer

add_employee

remove_employee

manage_employee

get_available_cars

show_active_rental

ADD_CAR

This function allows an Admin to add a new car to the rental system.
It collects details about the car (model, plate number, and rental price) from the user, validates the input, and adds the car to the system.

```
You, 6 hours ago | 1 author (You)
import 'dart:io';
import '../models/car.dart';
import '../services/rental_service.dart';

void addCarByAdmin(RentalService rentalService) {
  stdout.write("Enter car model: ");
  String model = stdin.readLineSync() ?? "";

  stdout.write("Enter car plate number: ");
  String plateNumber = stdin.readLineSync() ?? "";

  stdout.write("Enter rental price per day: ");
  double rentalPrice = double.tryParse(stdin.readLineSync() ?? "") ?? 0;

      You, 6 hours ago • intial commit
  if (rentalPrice <= 0) {
    print("Invalid rental price! Please enter a positive value.");
    return;
  }

  Car newCar = Car(model, plateNumber, rentalPrice);
  rentalService.addCar(newCar);
  print("Car added successfully by Admin: ${newCar.model}");
}
```

REMOVE_CAR

This function allows an Admin to remove a car from the rental system by specifying its plate number.

It searches for the car in the list, removes it if found, and provides feedback to the user.

Key Point to Highlight:

Error Handling:

- Handles cases where the car is not found gracefully by providing meaningful error messages using ->(try & catch)

```dart
import 'dart:io';
import '../models/car.dart';

void removeCar(List<Car> cars) {
  stdout.write("Enter car plate number to remove: ");
  String? plateNumber = stdin.readLineSync();

  try {
    Car carToRemove = cars.firstWhere((car) => car.plateNumber == plateNumber);
    cars.remove(carToRemove);
    print("Car removed successfully: ${carToRemove.model}");
  } catch (e) {
    print("Car not found with plate number: $plateNumber");
  }
}
```

RENT_CAR

This function allows a Customer to rent a car by selecting from a list of available cars. It collects necessary details such as the car choice, rental duration, payment method, and customer name, then processes the rental request.

Key Point to Highlight:
Integration with RentalService:
Leverages the RentalService class to manage the rental process, ensuring modularity and reusability.

```dart
import 'dart:io';
import '../models/customer.dart';
import '../models/car.dart';
import '../services/rental_service.dart';

void rentCarByUser(RentalService rentalService, Customer customer) {
  var availableCars = rentalService.getAvailableCars();
  if (availableCars.isEmpty) {
    print("No cars available for rent at the moment");
    return;
  }

  print("Choose a car to rent:");
  for (int i = 0; i < availableCars.length; i++) {
    print("${i + 1}. ${availableCars[i]}");
  }

  stdout.write("Enter car number: ");
  int carIndex = int.parse(stdin.readLineSync()!) - 1;
  Car selectedCar = availableCars[carIndex];

  stdout.write("Enter rental days: ");
  int days = int.parse(stdin.readLineSync()!);

  stdout.write("Enter payment method (Cash/Card): ");
  String? paymentMethod = stdin.readLineSync();


  stdout.write("Enter customer name: ");
  String customerName = stdin.readLineSync()?.trim() ?? "Unknown Customer";


  customer.name = customerName;


  rentalService.rentCar(customer, selectedCar, days, paymentMethod ?? "Cash");
}       You, 7 hours ago • intial commit
```

RETURN_CAR

This function allows a Customer to return a rented car by entering its plate number. It searches for the car in the rental system, verifies its existence, and processes the return if found.

```dart
import'dart:io';
import '../models/car.dart';
import '../services/rental_service.dart';

void returnCarByUser(RentalService rentalService) {
  stdout.write("Enter car plate number to return: ");
  String? plateNumber = stdin.readLineSync();
  Car? carToReturn = rentalService.cars.firstWhere(
      (car) => car.plateNumber == plateNumber, orElse: () => Car("", "", 0));

  if (carToReturn.model.isEmpty) {
    print("Car not found.");
  } else {
    rentalService.returnCar(carToReturn);
  }
}
```

Key Point to Highlight:
Integration with RentalService:
Leverages the RentalService class to manage the return process, ensuring modularity and reusability.

ADD_CUSTOMER

This function allows an Admin or Employee to add a new customer to the system. It collects details about the customer (ID, name, and phone number) from the user, creates a Customer object, and adds it to the list of customers.

Key Point to Highlight:

Global List Management:

Uses a global list (customers) to store all added customers, making it easy to manage and access customer data.

Reusability:

The function can be reused by both admins and employees to add customers.

```dart
import 'dart:io';
import '../models/customer.dart';

List<Customer> customers = [];

void addCustomer() {
  stdout.write("Enter customer ID: ");
  String id = stdin.readLineSync() ?? "";
  stdout.write("Enter customer name: ");
  String name = stdin.readLineSync() ?? "";
  stdout.write("Enter customer phone number: ");
  String phone = stdin.readLineSync() ?? "";

  Customer newCustomer = Customer(id, name, phone);
  customers.add(newCustomer);
  print("Customer added successfully: ${newCustomer.name}");
}
```

REMOVE_CUSTOMER

This function allows an Admin or Employee to remove a customer from the system by specifying their ID.
It searches for the customer in the global list (customers), removes them if found, and provides feedback to the user.

```dart
import'dart:io';
import '../models/customer.dart';
import '../methods/add_customer.dart'; // عشان نستخدم القائمة customers

void removeCustomer() {
  stdout.write("Enter customer ID to remove: ");
  String? id = stdin.readLineSync();

  try {
    Customer customerToRemove = customers.firstWhere((cust) => cust.id == id);
    customers.remove(customerToRemove);
    print("Customer removed successfully: ${customerToRemove.name}");
  } catch (e) {
    print("Customer not found with ID: $id");
  }
}
```

Key Point to Highlight:
Global List Management:
Uses a global list (customers) to remove all added customers, making it easy to manage and access customer data.
Reusability:
The function can be reused by both admins and employees to remove customers.

MANAGE_CUSTOMER

This function provides a menu for managing customers in the system.
It allows an Admin or Employee to add new customers, remove existing ones, or return to the main menu.

Key Point to Highlight:
Modularity:
The function integrates with other methods (addCustomer and removeCustomer) to perform specific actions, ensuring modularity.

```dart
import'dart:io';
import '../methods/add_customer.dart';
import '../methods/remove_customer.dart';

void manageCustomersMenu() {
  while (true) {
    print("\n Manage Customers Menu ");
    print("1- Add Customer");
    print("2- Remove Customer");
    print("3- Back");

    stdout.write("Enter your choice: ");
    String? choice = stdin.readLineSync();

    switch (choice) {
      case "1":
        addCustomer();
        break;
      case "2":
        removeCustomer();          You, 7 hou
        break;
      case "3":
        return;
      default:
        print("Invalid choice!");
    }
  }
}
```

ADD_EMPLOYEE

This function allows an Admin to add a new employee to the system.
It collects details about the employee (ID and name) from the user, creates an Employee object, and adds it to the list of employees.

Key Point to Highlight:
Reusability:
The function can be reused by admins to add multiple employees.
User Feedback:
Provides clear feedback to the admin after successfully adding an employee.

```dart
import'dart:io';
import '../models/employee.dart';


List<Employee> employees = [];

void addEmployee() {
  stdout.write("Enter employee ID: ");
  String id = stdin.readLineSync() ?? "";
  stdout.write("Enter employee name: ");
  String name = stdin.readLineSync() ?? "";

  Employee newEmployee = Employee(id, name);
  employees.add(newEmployee);
  print("Employee added successfully: ${newEmployee.name}");
}       You, 7 hours ago • intial commit …
```

REMOVE_EMPLOYEE

This function allows an Admin to remove an employee from the system by specifying their ID. It searches for the employee in the global list (employees), removes them if found, and provides feedback to the user.

```dart
import'dart:io';
import '../models/employee.dart';
import '../methods/add_employee.dart'; // عشان نستخدم القائمة employees

void removeEmployee() {
  stdout.write("Enter employee ID to remove: ");
  String? id = stdin.readLineSync();

  try {
    Employee employeeToRemove = employees.firstWhere((emp) => emp.id == id);
    employees.remove(employeeToRemove);
    print("Employee removed successfully: ${employeeToRemove.name}");
  } catch (e) {
    print("Employee not found with ID: $id");
  }
}
```

Key Point to Highlight:

Reusability:

The function can be reused by admins to remove multiple employees.

Global List Management:

Operates directly on the global employees list, making it easy to manage employee data.

MANAGE_EMPLOYEE

This function provides a menu for managing employees in the system. It allows an Admin to add new employees, remove existing ones, or return to the main menu.

Key Point to Highlight:

Reusability:

The function can be reused by admins to remove multiple employees.

Modularity:

The function integrates with other methods (addEmployee and removeEmployee) to perform specific actions, ensuring modularity.

```
You, 8 hours ago | 1 author (You)
import'dart:io';
import '../methods/add_employee.dart';
import '../methods/remove_employee.dart';

void manageEmployeesMenu() {
  while (true) {
    print("\n Manage Employees Menu ");
    print("1- Add Employee");
    print("2- Remove Employee");
    print("3- Back");

    stdout.write("Enter your choice: ");
    String? choice = stdin.readLineSync();

    switch (choice) {
      case "1":
        addEmployee();
        break;
      case "2":
        removeEmployee();
        break;
      case "3":
        return;
      default:
        print("Invalid choice!");
    }       You, 8 hours ago • intial commit
  }
}
```

GET_AVAILABLE_CAR

This function filters a list of cars to return only those that are currently available for rent. It is used to simplify the process of retrieving available cars from a larger list of all cars in the system.

```dart
import '../models/car.dart';

List<Car> getAvailableCarsFromList(List<Car> cars) {
  return cars.where((car) => car.isAvailable).toList();
}
```


Key Point to Highlight:

Integration with Models:

Relies on the isAvailable property of the Car class, ensuring consistency across the system.

Efficiency:

The function uses Dart's built-in where method, which is efficient for filtering collections.

SHOW_ACTIVE_RENTALS

This function displays all active rental contracts in the system. It checks if there are any active rentals and, if so, prints their details. If no rentals are active, it informs the user accordingly.

```dart
import '../models/rental_contract.dart';

void showActiveRentals(List<RentalContract> contracts) {
  if (contracts.isEmpty) {
    print("No active rentals.");
    return;
  }

  print("Active Rentals:");
  for (var contract in contracts) {
    print(contract);
  }
}       You, 8 hours ago • intial commit …
```

Key Point to Highlight:

Integration with Models:

Relies on the toString method of the RentalContract class to provide a human-readable output.

Efficiency:

The function first checks if the list is empty to avoid unnecessary iterations.

1- MODELS:

User

admin

employee

customer

car

payment

rental_contrast

USER CLASS

The User class is a base class that represents a generic user in the system. It contains basic information about the user, such as their ID and name. This class can be extended by other classes (e.g., Admin, Employee, Customer) to add more specific functionality.

Key Point to Highlight:

Reusability:
Acts as a base class for other user types, promoting code reuse and modularity

```
class User {
  String id;
  String name;

  User(this.id, this.name);

  @override
  String toString() {
    return "User: $name | ID: $id";
  }
}       You, 8 hours ago • intial com
```

ADMIN CLASS

The Admin class represents an administrator in the system. It provides a menu (startMenu) that allows the admin to perform administrative tasks such as managing cars, employees, and customers, as well as returning to the main menu.

Key Point to Highlight:

Modularity:

Each menu option calls a specific function, ensuring modularity separation of concerns.

Integration with Other Methods:

Relies on external methods like addCarByAdmin, removeCar, manageEmployeesMenu, and manageCustomersMenu to perform tasks.

```dart
import 'dart:io';
import '../services/rental_service.dart';
import '../methods/add_car.dart';
import '../methods/manage_employees.dart';
import '../methods/manage_customers.dart';
import '../methods/remove_car.dart';

You, 8 hours ago | 1 author (You)
class Admin {
  String id;
  String name;

  Admin(this.id, this.name);

  void startMenu(RentalService rentalServic
    while (true) {
      print("\n Admin Menu ");
      print("1- Add a Car");
      print("2- Remove a Car");
      print("3- Manage Employees");
      print("4- Manage Customers");
      print("5- Back to Main Menu");

      stdout.write("Enter your choice: ");
      String? choice = stdin.readLineSync()

      switch (choice) {
        case "1":       You, 8 hours ago • i
          addCarByAdmin(rentalService);
          break;
        case "2":
          removeCar(rentalService.cars);
          break;
        case "3":
          manageEmployeesMenu();
          break;
        case "4":
          manageCustomersMenu();
          break;
        case "5":
          return;
        default:
          print("Invalid choice!");
      }
    }
  }

  @override
  String toString() {
    return "Admin: $name | ID: $id";
  }
```

EMPLOYEE CLASS

The Employee class represents an employee in the system. It provides a menu (startMenu) that allows employees to perform tasks such as renting cars, returning cars, managing customers, and removing cars. Additionally, it includes a helper method (createCustomer) for creating temporary customer objects during car rentals.

Key Point to Highlight:

Validation:

The createCustomer method validates both the customer's name and phone number before creating the object.

```dart
class Employee {
  String id;
  String name;

  Employee(this.id, this.name);

  void startMenu(RentalService rentalService) {
    while (true) {
      print("\n Employee Menu ");
      print("1- Rent a Car");
      print("2- Return a Car");
      print("3- Show Active Rentals");
      print("4- Manage Customers");
      print("5- Remove a Car");
      print("6- Back to Main Menu");

      stdout.write("Enter your choice: ");
      String? choice = stdin.readLineSync();

      switch (choice) {
        case "1":
          rentCarByUser(rentalService, createCustomer()); // تأجير سيارة
          break;
        case "2":
          returnCarByUser(rentalService); // إرجاع سيارة    "إرجاع": Unknown
          break;
        case "3":
          showActiveRentals(rentalService.contracts); // عرض العقود النشطة
          break;
        case "4":
          manageCustomersMenu(); // (جديد)
          break;
        case "5":
          removeCar(rentalService.cars); //
          break;
        case "6":
          return; //رجوع للقائمة الرئيسية
        default:
          print("Invalid choice!");
      }
    }
  }

// سيتم لاحقًا إنشاء عميل جديد بس بشكل مؤقت    "مؤقت": Unknown word.
Customer createCustomer() {
  String name;
  while (true) {
    stdout.write("Enter customer's full name (at least 3 words): ");
    name = stdin.readLineSync()?.trim() ?? "";
    if (name.split(" ").length >= 3) break;
    print("Invalid name! Please enter the first, middle, and last name.");
  }

  String phone;
  while (true) {
    stdout.write("Enter customer's phone number (at least 10 digits): ");
    phone = stdin.readLineSync()?.trim() ?? "";
    if (RegExp(r'^\d{10,}$').hasMatch(phone)) break;
    print("Invalid phone number! It should be at least 10 digits.");
  }

  return Customer("cust123", name, phone);    "cust": Unknown word.
}

@override
String toString() {
  return "Employee: $name | ID: $id";
}
}
```

CUSTOMER CLASS

The Customer class represents a customer in the system. It provides a menu (startMenu) that allows customers to perform tasks such as renting cars, returning cars, and returning to the main menu. Additionally, it includes properties for the customer's ID, name, and phone number.

Key Point to Highlight:
Integration with Other Methods:
Relies on external methods like rentCarByUser and returnCarByUser to perform tasks.

```dart
class Customer {
  String id;
  String name;
  String phone;

  Customer(this.id, this.name, this.phone);

  void startMenu(RentalService rentalService) {
    while (true) {
      print("\n Customer Menu ");
      print("1- Rent a Car");
      print("2- Return a Car");
      print("3- Back to Main Menu");       You, 20 hours

      stdout.write("Enter your choice: ");
      String? choice = stdin.readLineSync();

      switch (choice) {
        case "1":
          rentCarByUser(rentalService, this);
          break;
        case "2":
          returnCarByUser(rentalService);
          break;
        case "3":
          return;
        default:
          print("Invalid choice!");
      }
    }
  }

  @override
  String toString() {
    return "Customer: $name | Phone: $phone | ID: $id";
  }
}
```

CAR CLASS

The Car class represents a car in the system. It contains properties to store details about the car (e.g., model, plate number, rental price) and methods to manage its rental status (e.g., renting and returning the car).

The class also overrides the toString method to provide a readable string representation of the car.

Key Point to Highlight:

State Management:

Tracks the car's availability and booking status using isAvailable and bookedUntil.

```dart
class Car {
  String model;
  String plateNumber;
  double rentalPrice;
  bool isAvailable;
  DateTime? bookedUntil;

  Car(this.model, this.plateNumber, this.rentalPrice,
      {this.isAvailable = true, this.bookedUntil});

  void rentCar(DateTime returnDate) {
    isAvailable = false;
    bookedUntil = returnDate;
  }

  void returnCar() {       You, 2 weeks ago • First commit
    isAvailable = true;
    bookedUntil = null;
  }

  @override
  String toString() {
    return "Car: $model | Plate: $plateNumber | Price per day: \$${rentalPrice} | Available: $isAvailable";
  }
}
```

PAYMENT CLASS

The Payment class represents a payment transaction in the system. It stores details about the payment, such as the amount, payment method, and the date/time when the payment was made.
The class also overrides the toString method to provide a human-readable string representation of the payment.

```dart
class Payment {
  double amount;
  String method;
  DateTime paymentDate;

  Payment(this.amount, this.method) : paymentDate = DateTime.now();

  @override
  String toString() {
    return "Payment: \$$amount via $method on $paymentDate";
  }
}
```

Key Point to Highlight:
Automatic Date/Time Tracking:
The paymentDate property is automatically set to the current date and time using DateTime.now(), ensuring accurate tracking of when the payment was made.

RENTAL CONTRAST CLASS

The RentalContract class represents a rental agreement between a customer and a car in the system. It encapsulates details about the rental, such as the customer, car, rental duration, total cost, rental/return dates, and payment information.
The class also overrides the toString method to provide a human-readable string representation of the rental contract.

```dart
import 'customer.dart';
import 'car.dart';
import 'payment.dart';

You, 20 hours ago | 1 author (You)
class RentalContract {
  Customer customer;
  Car car;
  int rentalDays;
  double totalCost;
  DateTime rentalDate;
  DateTime returnDate;
  Payment payment;

  RentalContract(
      this.customer, this.car, this.rentalDays, this.rentalDate, this.payment)
      : returnDate = rentalDate.add(Duration(days: rentalDays)),
        totalCost = rentalDays * car.rentalPrice;

  @override
  String toString() {
    return "RentalContract: ${customer.name} rented ${car.model} for $rentalDays days from $rentalDate to $returnDate\n$payment";
  }
}
```

Key Point to Highlight:

Automatic Calculations:

The returnDate and totalCost are automatically calculated based on the rental duration and car's rental price.

3- SERVICES

RENTAL SERVICE

RENTAL SERVICE

The RentalService class manages the core functionality of the car rental system.

It handles operations such as adding cars, renting cars, returning cars, and retrieving available cars.

It also maintains two main lists:

- cars: A list of all cars in the system.
- contracts: A list of all active rental contracts

Key Point to Highlight:

Centralized Management:

The RentalService class serves as the central hub for managing cars and rental contracts.

```
class RentalService {
  List<Car> cars = [];

  List<RentalContract> contracts = [];

  void addCar(Car car) {
    cars.add(car);
    print("Car added successfully: ${car.model}");
  }

  List<Car> getAvailableCars() {
    return getAvailableCarsFromList(cars);
  }

  void rentCar(Customer customer, Car car, int days, String paymentMethod) {
    if (!car.isAvailable) {
      print("Car is not available for rent.");
      return;
    }

    double totalAmount = car.rentalPrice * days;

    Payment payment = Payment(totalAmount, paymentMethod);

    DateTime rentalDate = DateTime.now();
    RentalContract contract =
        RentalContract(customer, car, days, rentalDate, payment);
      You, 20 hours ago • intial commit …
    contracts.add(contract);
    car.rentCar(contract.returnDate);

    print("Car rented successfully: ${car.model}");
    print(contract);
  }
  void returnCar(Car car) {
    if (car.isAvailable) {
      print("Car is already available.");
      return;
    }

    try {
      RentalContract contract = contracts.firstWhere((c) => c.car == car);
      car.returnCar();
      contracts.remove(contract);

      print("Car returned successfully: ${car.model}");
    } catch (e) {
      print("No active rental found for this car.");
    }
  }
}
```

4- MAIN

The main function serves as the entry point for the car rental system.
It provides a user-friendly interface for interacting with the system by allowing users to choose their role (Admin, Employee, or Customer) and perform tasks specific to their role.
The function also initializes the RentalService object, which manages the core functionality of the system.

```dart
import 'dart:io';
import 'models/admin.dart';
import 'models/employee.dart';
import 'models/customer.dart';
import 'services/rental_service.dart';

Run | Debug
void main() {
  RentalService rentalService = RentalService();



  while (true) {
    print("\n Car Rental System ");
    print("Choose your user type:");
    print("1- Admin");
    print("2- Employee");
    print("3- Customer");
    print("4- Exit");
    stdout.write("Enter your choice: ");

    String? userTypeChoice = stdin.readLineSync();

    switch (userTypeChoice) {
      case "1":
        Admin admin = Admin("Myrna2", "System Admin");
        admin.startMenu(rentalService);
        break;
      case "2":
        Employee employee = Employee("emp123", "System Employee");
        employee.startMenu(rentalService);
        break;
      case "3":
        Customer customer = Customer("cust123", "Myrn", "1234567890");
        customer.startMenu(rentalService);
        break;
        case "4":
          print("Exiting...");
          return;
        default:

              You, 20 hours ago • fix changes
          print("Invalid choice! Please enter a valid option.");
      }
    }
  }
```

OKAY ..THATS ALL!

Do you have any questions for me before we go?

THANK YOU!

MADE BY
MYRNA NADER