Sure, let's break down the provided Java code step by step:

Import Statements

- The code begins with import statements that bring in necessary classes from the `java.util` package, including `ArrayList`, `List`, and `Scanner`.

Interface Definition: `RentalStructure`

- Defines an interface `RentalStructure` with three abstract methods: `displayVehicles()`, `bookVehicle()`, and `addVehicle()`.

- This interface outlines the structure that rental system classes must follow.

Abstract Class: `Vehicle`

- Defines an abstract class `Vehicle` with common attributes and methods that all vehicles in the rental system will have.

- Attributes include company, model, year, cost per hour, and availability.

- Provides methods to get and set these attributes and declares an abstract method `displayDetails()`

Concrete Classes: `TwoWheeler` and `FourWheeler`

- These classes extend the `Vehicle` abstract class, providing concrete implementations for the `displayDetails()` method.

- They define constructors to initialize the vehicle attributes and call the superclass constructor using `super()`.

Class: `RentalSystem` (Implements `RentalStructure`)

- Implements the `RentalStructure` interface, which mandates the implementation of the methods defined in `RentalStructure`.

- Contains a list `availableVehicles` to store available vehicles.

- Initializes the `availableVehicles` list with some default vehicles in the constructor.

- Implements methods to display available vehicles, book a vehicle, and add a new vehicle.

  - `displayVehicles()`: Displays the details of available vehicles by iterating through the `availableVehicles` list and calling `displayDetails()` for each vehicle.

  - `bookVehicle()`: Allows the user to book a vehicle by selecting from available options and providing necessary booking details.

  - `addVehicle()`: Enables the addition of a new vehicle to the rental system by prompting the user for vehicle details and adding the vehicle to the `availableVehicles` list.

`main` Method in `Main` Class

- The `Main` class contains the `main` method, which serves as the entry point of the program.

- It creates an instance of `RentalSystem` and a `Scanner` object to take user input.

- Utilizes a `do-while` loop to display a menu of options and perform corresponding actions based on user input.

- The loop continues until the user chooses to exit by selecting option 4.

Summary

This code implements a vehicle rental system where users can view available vehicles, book vehicles, and add new vehicles. It follows the structure defined by the `RentalStructure` interface and utilizes inheritance to create specific vehicle types. The `main` method orchestrates the interaction with the user by presenting a menu and executing the selected actions.

DETAILED

Certainly! Let's delve deeper into each part of the provided Java code:

Import Statements

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
```

- These import statements bring in necessary classes from the `java.util` package.
- `ArrayList`, `List`, and `Scanner` classes are imported, which are essential for working with lists, collection classes, and user input, respectively.

Interface Definition: `RentalStructure`

```java
interface RentalStructure {
    void displayVehicles();
    void bookVehicle();
    void addVehicle();
}
```

- This interface `RentalStructure` declares three abstract methods: `displayVehicles()`, `bookVehicle()`, and `addVehicle()`.
- It defines the structure that any rental system class must implement.

Abstract Class: `Vehicle`

```java
abstract class Vehicle {
    // Attributes
    private String company;
    private String model;
    private int year;
    private int costPHour;
    private boolean available; // New field to track availability

    // Constructor
    public Vehicle(String company, String model, int year, int costPHour) {
        // Initialize attributes
        this.company = company;
```

```
        this.model = model;
        this.year = year;
        this.costPHour = costPHour;
        this.available = true; // Initialize availability to true
    }
    // Getter methods
    public String getCompany() { ... }
    public String getModel() { ... }
    public int getYear() { ... }
    public int getCostPHour() { ... }
    public boolean isAvailable() { ... }
    // Setter method
    public void setAvailable(boolean available) { ... }
    // Abstract method to display details
    public abstract void displayDetails();
}
```

- `Vehicle` is an abstract class representing common attributes and methods of all vehicles in the rental system.
- It includes attributes like company, model, year, cost per hour, and availability.
- The constructor initializes these attributes, and getter and setter methods provide access to them.
- An abstract method `displayDetails()` is declared, which must be implemented by concrete subclasses.

Concrete Classes: `TwoWheeler` and `FourWheeler`

```java
class TwoWheeler extends Vehicle {
    public TwoWheeler(String company, String model, int year, int costPHour) {
        super(company, model, year, costPHour); // Call superclass constructor
    }

    public void displayDetails() { ... }
}

class FourWheeler extends Vehicle {
    public FourWheeler(String company, String model, int year, int costPHour) {
        super(company, model, year, costPHour); // Call superclass constructor
    }
```

```
    public void displayDetails() { ... }
}
```

- These are concrete subclasses of the `Vehicle` abstract class, representing specific types of vehicles.
- Each subclass provides a constructor to initialize attributes using the `super()` call to the superclass constructor.
- They override the `displayDetails()` method to provide specific details about two-wheelers and four-wheelers.

Class: `RentalSystem` (Implements `RentalStructure`)

```java
class RentalSystem implements RentalStructure {
    private List<Vehicle> availableVehicles;

    public RentalSystem() {
        availableVehicles = new ArrayList<>();
        // Add default vehicles to the list
    }

    public void displayVehicles() { ... }
    public void bookVehicle() { ... }
    public void addVehicle() { ... }
}
```

- `RentalSystem` class implements the `RentalStructure` interface, ensuring the implementation of its methods.
- It maintains a list of available vehicles (`availableVehicles`) using the `List` interface and `ArrayList` class.
- The constructor initializes the `availableVehicles` list with some default vehicles.
- It implements methods to display available vehicles, book a vehicle, and add a new vehicle to the system.

`main` Method in `Main` Class

```java
public class Main {
    public static void main(String[] args) {
        // Instantiate RentalSystem and Scanner objects
        RentalSystem obj = new RentalSystem();
        Scanner scanner = new Scanner(System.in);

        // Display menu and process user input
        int choice;
```

```
        do {
            // Display menu options
            // Take user input and execute corresponding actions
        } while (choice != 4);

        scanner.close(); // Close Scanner object
    }
}
```

- The `main` method serves as the entry point of the program.

- It instantiates a `RentalSystem` object and a `Scanner` object to handle user input.

- Inside a `do-while` loop, it displays a menu of options and performs actions based on user input until the user chooses to exit.

Summary

This code represents a vehicle rental system implemented in Java. It defines an interface, an abstract class, concrete subclasses, and a main class to orchestrate the system's functionality. The `RentalSystem` class acts as the core component, managing available vehicles, booking, and adding new vehicles. The `main` method handles user interaction through a menu-driven interface.

ALL Concepts involved in this project

Sure, let's go through the major concepts used in the project and their definitions:

1. **Interface (`RentalStructure`)**:

   - An interface in Java is a reference type that can contain only constants, method signatures, default methods, static methods, and nested types.

   - In this project, the `RentalStructure` interface defines the structure that any rental system class must implement. It ensures that any class implementing this interface will provide implementations for the `displayVehicles()`, `bookVehicle()`, and `addVehicle()` methods.

2. **Abstract Class (`Vehicle`)**:

   - An abstract class in Java is a class that cannot be instantiated on its own and can contain both abstract methods (methods without a body) and concrete methods.

   - The `Vehicle` class serves as a blueprint for all types of vehicles in the rental system.

   - It contains common attributes and methods shared by all vehicles, such as `company`, `model`, `year`, `costPHour`, and `available`.

   - The `displayDetails()` method is declared as abstract to be implemented by concrete subclasses, ensuring that each type of vehicle can provide its specific details.

3. **Concrete Classes (`TwoWheeler` and `FourWheeler`)**:

- Concrete classes are classes that provide implementations for all abstract methods declared in their parent classes or interfaces.

- In this project, `TwoWheeler` and `FourWheeler` are concrete subclasses of the `Vehicle` abstract class.

- They provide specific implementations for the `displayDetails()` method to display details about two-wheelers and four-wheelers, respectively.

4. **Inheritance**:

- Inheritance is a mechanism in Java where a new class inherits properties and behaviors (methods) from an existing class.

- In this project, `TwoWheeler` and `FourWheeler` inherit common properties and behaviors from the `Vehicle` class, such as `company`, `model`, `year`, `costPHour`, and `displayDetails()` method.

5. **List (`availableVehicles`)**:

- A `List` in Java is an ordered collection of elements that allows duplicate elements.

- In this project, the `availableVehicles` list is used to store instances of vehicles available for rental.

- It provides dynamic sizing, automatic resizing, and a variety of methods to manipulate the elements it contains.

6. **Scanner**:

- `Scanner` is a class in Java used for obtaining the input of the primitive types like `int`, `double`, etc., and strings.

- In this project, `Scanner` is used to read user input from the console for various operations like booking a vehicle, adding a new vehicle, and selecting menu options.

7. **Constructor**:

- A constructor in Java is a special type of method that is used to initialize objects.

- In this project, constructors are used to initialize objects of the `Vehicle`, `TwoWheeler`, `FourWheeler`, and `RentalSystem` classes with initial values for their attributes.

8. **Method Overriding**:

- Method overriding in Java occurs when a subclass provides a specific implementation of a method that is already provided by one of its parent classes.

- In this project, both `TwoWheeler` and `FourWheeler` classes override the `displayDetails()` method inherited from the `Vehicle` class to provide specific details about each type of vehicle.

These concepts are used to organize the code, define relationships between classes, facilitate code reusability, and provide a structure for building the rental system functionality.