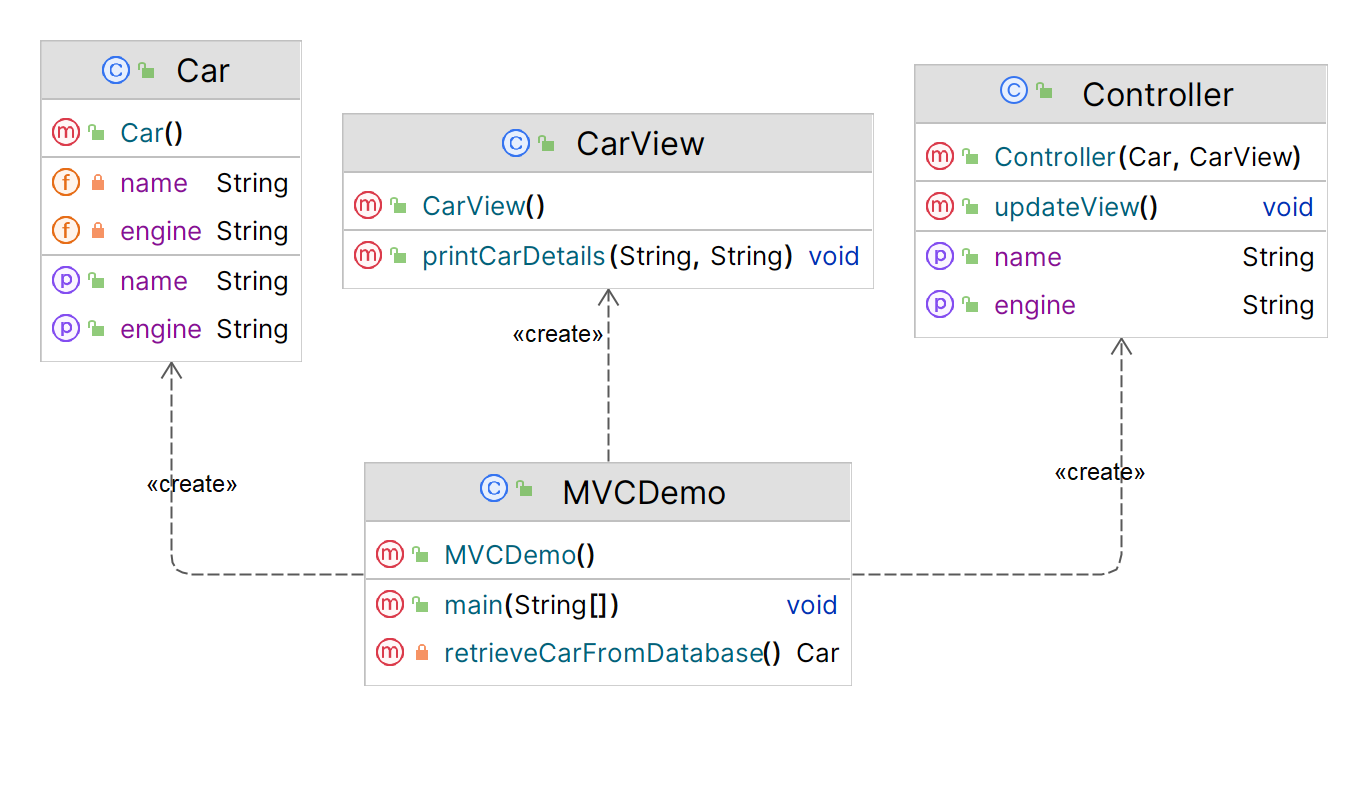
**Assignment 16: MVC Design Pattern**

**What is MVC Design Pattern?**

The **Model View Controller** design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

MVC is more of an **architectural pattern**, but not for complete application. MVC is mostly **UI / UX** of an application. It still needs business logic layer, maybe some service layer and data access layer.

**Structure (Class Diagram)**



**Implementation (Code)**

|  |
| --- |
| import java.sql.\*;  ***// Model***  public class Car {  private String name;  private String engine;   public String getEngine() {  return engine;  }  public void setEngine(String engine) {  this.engine = engine;  }   public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  } }  ***// View***  public class CarView {  public void printCarDetails(String carName, String carEngine) {  System.*out*.println("\nCar: ");  System.*out*.println("Name: " + carName);  System.*out*.println("Engine: " + carEngine);  } }  ***// Controller***  public class Controller {  private Car model;  private CarView view;   public Controller(Car model, CarView view) {  this.model = model;  this.view = view;  }  public void setName(String name) {  model.setName(name);  }  public String getName() {  return model.getName();  }  public void setEngine(String engine) {  model.setEngine(engine);  }  public String getEngine() {  return model.getEngine();  }   public void updateView(){  view.printCarDetails(model.getName(), model.getEngine());  } }  ***// Main Class*** public class MVCDemo {  public static void main(String[] args) {  *// Fetch student record based on his roll no from the database* Car model = *retrieveCarFromDatabase*();   *// Create a view to write student details on console* CarView view = new CarView();  Controller controller = new Controller(model, view);  controller.updateView();   *// Update model data* controller.setName("Tata Nexon EV");  controller.setEngine("1.5L");  controller.updateView();  }  // Database Connection (SQL)  private static Car retrieveCarFromDatabase() {  Car car = new Car();   String url = "jdbc:mysql://localhost:3306/DP\_LAB?useSSL=false";  String username = "user";  String password = "\*\*\*";   try (Connection conn = DriverManager.*getConnection*(url, username, password)) {  System.*out*.println("Connected to database!");   Statement statement = conn.createStatement();  ResultSet resultSet = statement.executeQuery("SELECT \* FROM CARS");   int count = 1;  while (resultSet.next()) {  String name = resultSet.getString("CAR\_NAME");  String engine = resultSet.getString("CAR\_ENGINE");  car.setName(name);  car.setEngine(engine);  count++;  }  } catch (SQLException ex) {  System.*err*.println("Error connecting to database: " + ex.getMessage());  }  return car;  } }  **Output**    **Database Design (SQL)**  CREATE TABLE CARS (  CAR\_NAME VARCHAR(40),  CAR\_ENGINE VARCHAR(30) );  INSERT INTO CARS (CAR\_NAME, CAR\_ENGINE) VALUES ('Honda Civic', '1.5L'),  ('Ford Mustang', 'V8'),  ('Chevrolet Corvette', 'Supercharged V8'); |
|  |

**Applicability**

1. **MVC** design pattern separates the application's concerns into **three distinct components**, making it easier to maintain and modify each component **independently**.
2. **MVC** promotes **code reusability** and **scalability**, as each component can be developed and tested separately before being **integrated** into the overall application.
3. **MVC** helps in designing **user-friendly interfaces** by providing a **clear separation** between the presentation layer and the underlying data and logic.