**Exercise 10: Implementing the MVC Pattern**

**Scenario:**

You are developing a simple web application for managing student records using the MVC pattern.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **MVCPatternExample**.
2. **Define Model Class:**
   * Create a class **Student** with attributes like **name, id, and grade**.
3. **Define View Class:**
   * Create a class **StudentView** with a method **displayStudentDetails()**.
4. **Define Controller Class:**
   * Create a class **StudentController** that handles the communication between the model and the view.
5. **Test the MVC Implementation:**
   * Create a main class to demonstrate creating a **Student**, updating its details using **StudentController**, and displaying them using **StudentView**.

**Solution:**

Implement the **Model-View-Controller (MVC)** pattern to separate:

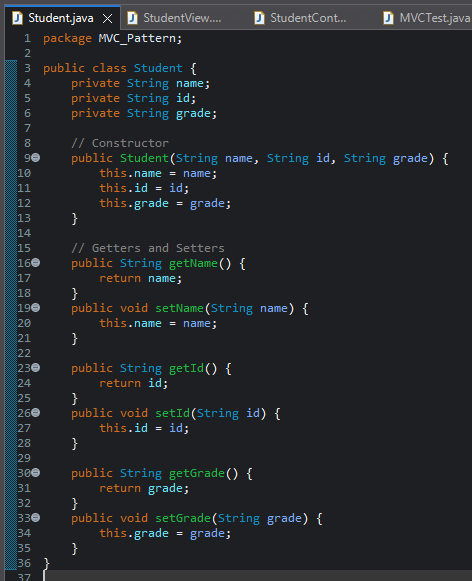
* **Model**: Data (Student)
* **View**: Presentation (StudentView)
* **Controller**: Logic to connect Model and View (StudentController)

**Code:**

1. **Model Class: Student**

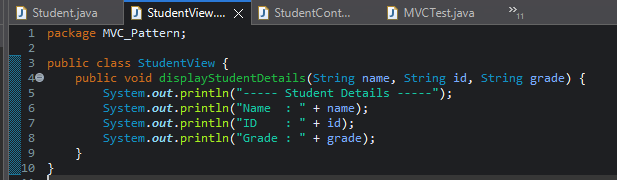
This is the **data layer**.

Holds all student-related information with getter and setter methods.



**2. View Class: StudentView.java**

* This is the presentation layer.
* Displays the student’s information.

****

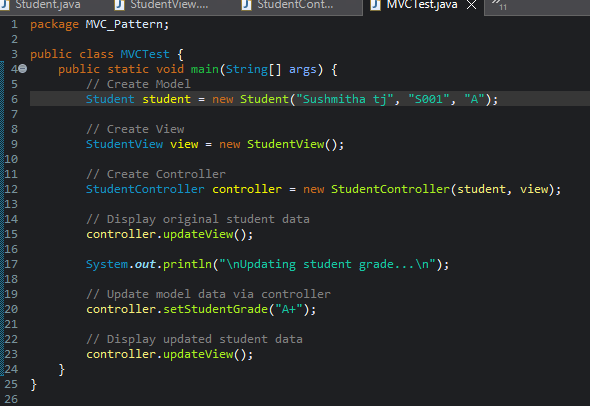
**3. Controller Class: StudentController.java**

* This is the controller that connects the model and the view.
* It updates the model and triggers the view update.

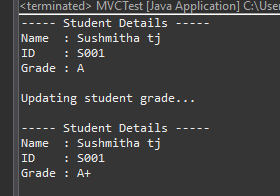
****

**4. Test Class: MVCTest.java (Main Class)**

* Demonstrates how you can **create, update, and view** student data.
* All components (Model, View, Controller) are **loosely coupled**.



**Output:**

****

**Benefits of MVC Pattern**

* ✔ Separation of concerns: Business logic, UI, and data are modular.
* ✔ Easy to maintain: Changes in one component don’t affect others directly.
* ✔ Scalable: Easy to adapt for larger applications and frameworks (like Spring MVC).