**Exercise 10: Implementing the MVC Pattern**

**Scenario:**

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

1. Create a new Java Project:

* Create a new Java Project named MVCPatternExample.
* A Java project named MVCPatternExample is created in eclipse IDE.

2. Define Model class:

* Create a class **Student** with attributes like **name, id, and grade**.

public class Student {

private String name;

private int id;

private String grade;

public Student(String name, int id, String grade) {

this.name = name;

this.id = id;

this.grade = grade;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getGrade() {

return grade;

}

public void setGrade(String grade) {

this.grade = grade;

}

}

* Holds data and provides methods to get and set student details.

3. Define View Class:

* Create a class **StudentView** with a method **displayStudentDetails()**.

public class StudentView {

public void displayStudentDetails(String studentName, int studentId, String studentGrade) {

System.out.println("Student Details:");

System.out.println("Name: " + studentName);

System.out.println("ID: " + studentId);

System.out.println("Grade: " + studentGrade);

}

}

* Responsible for displaying student details.

4. Define Controller Class:

* Create a class **StudentController** that handles the communication between the model and the view.

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void setStudentName(String name) {

model.setName(name);

}

public String getStudentName() {

return model.getName();

}

public void setStudentId(int id) {

model.setId(id);

}

public int getStudentId() {

return model.getId();

}

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

public String getStudentGrade() {

return model.getGrade();

}

public void updateView() {

view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());

}

}

* Manages updates between the model and the view, and updates the view with the model’s data.

5. Test the MVC Implementation:

* Create a main class to demonstrate creating a **Student**, updating its details using **StudentController**, and displaying them using **StudentView**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter initial student details:");

System.out.print("Name: ");

String name = scanner.nextLine();

System.out.print("ID: ");

int id = scanner.nextInt();

scanner.nextLine();

System.out.print("Grade: ");

String grade = scanner.nextLine();

Student model = new Student(name, id, grade);

StudentView view = new StudentView();

StudentController controller = new StudentController(model, view);

controller.updateView();

System.out.println("\nUpdate student details:");

System.out.print("New Name: ");

name = scanner.nextLine();

controller.setStudentName(name);

System.out.print("New ID: ");

id = scanner.nextInt();

scanner.nextLine();

controller.setStudentId(id);

System.out.print("New Grade: ");

grade = scanner.nextLine();

controller.setStudentGrade(grade);

controller.updateView();

scanner.close();

}

}

* Demonstrates user input for creating and updating a student, and interacts with the controller and view.

Output:

