Student Records Management with MVC Pattern

public class MVCPatternExample {

public static void main(String[] args) {

Student model = retrieveStudentFromDatabase();

StudentView view = new StudentView();

StudentController controller = new StudentController(model, view);

controller.updateView();

System.out.println("\nUpdating student information...");

controller.setStudentName("John Smith");

controller.setStudentId("S12345");

controller.setStudentGrade("A-");

controller.updateView();

System.out.println("\nAdding a new student...");

Student model2 = new Student("", "", "");

StudentController controller2 = new StudentController(model2, view);

controller2.setStudentName("Emily Johnson");

controller2.setStudentId("S67890");

controller2.setStudentGrade("B+");

controller2.updateView();

}

// Helper method to create initial student data

private static Student retrieveStudentFromDatabase() {

Student student = new Student();

student.setName("Robert Johnson");

student.setId("S10001");

student.setGrade("B");

return student;

}

// Model

static class Student {

private String name;

private String id;

private String grade;

public Student() {}

public Student(String name, String 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 String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getGrade() {

return grade;

}

public void setGrade(String grade) {

this.grade = grade;

}

}

// View

static class StudentView {

public void displayStudentDetails(String name, String id, String grade) {

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

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

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

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

}

}

// Controller

static 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(String id) {

model.setId(id);

}

public String 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());

}

}

}

Key Features:

1. **Separation of Concerns**: Clear division between data, presentation, and control
2. **Model (Student)**: Contains the data and business logic
3. **View (StudentView)**: Handles display and presentation
4. **Controller (StudentController)**: Manages data flow between model and view
5. **Loose Coupling**: Components interact through well-defined interfaces

MVC Pattern Benefits:

1. **Modularity**: Components can be developed and tested independently
2. **Maintainability**: Easier to modify or extend individual components
3. **Reusability**: Views can be reused with different models
4. **Flexibility**: Multiple views can display the same data differently
5. **Testability**: Components can be unit tested in isolation

Output:

