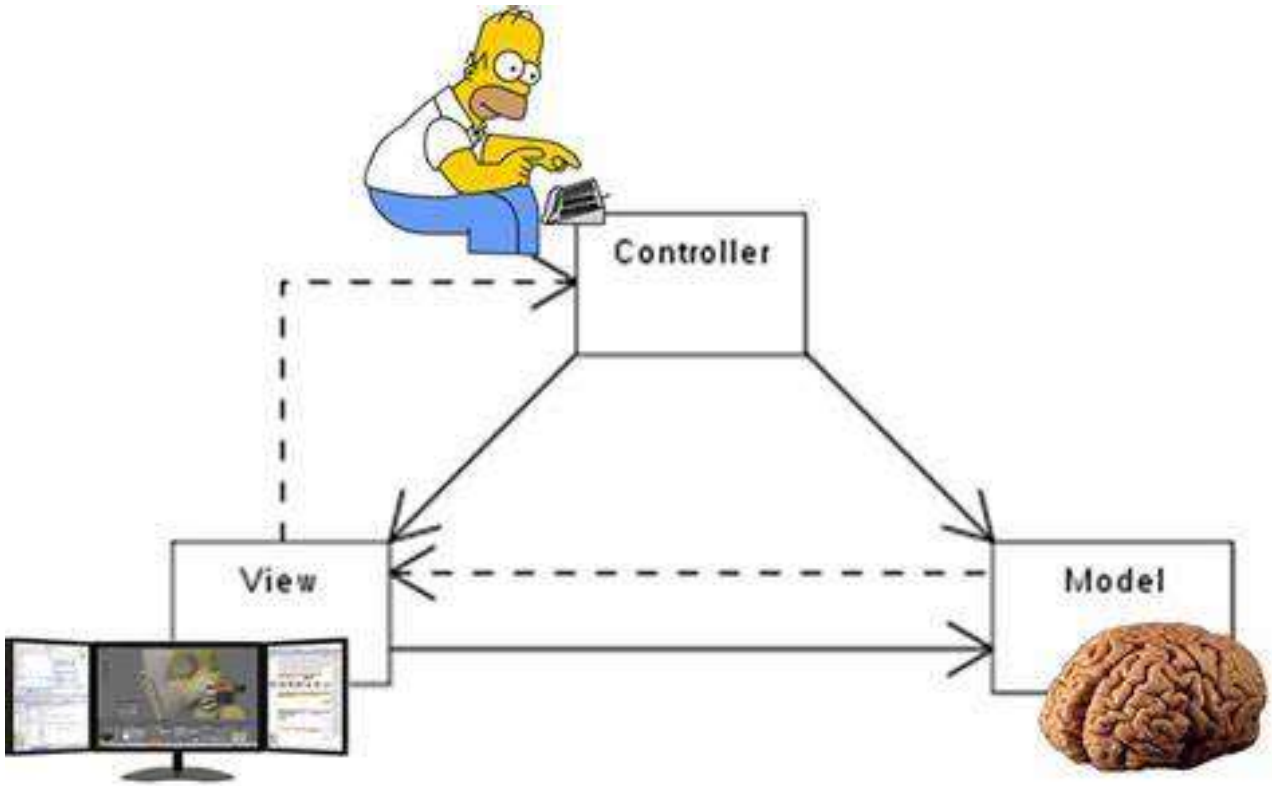
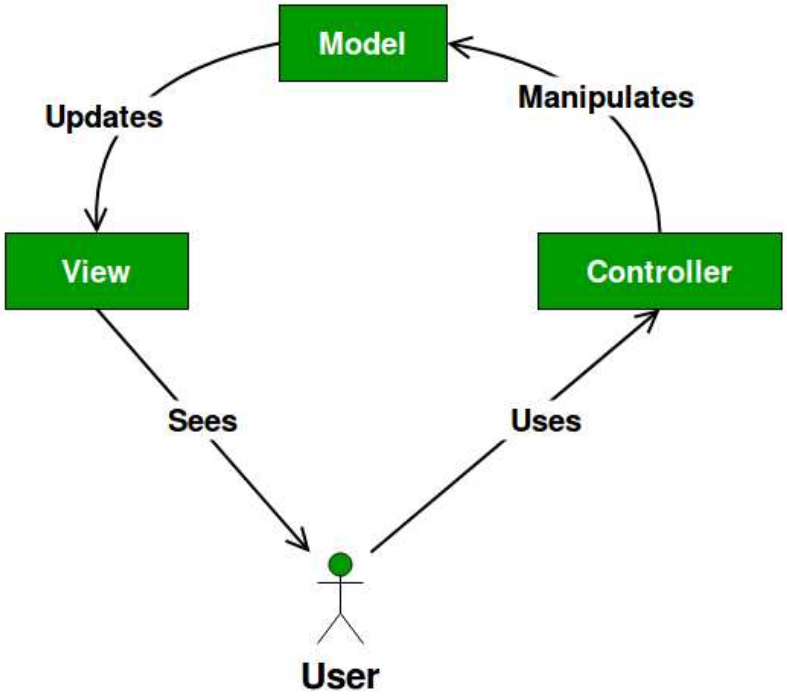
Model View Controller (MVC) Design Pattern



Introduction

- The Model View Controller (MVC) 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 mostly relates to the UI / interaction layer of an application. You're still going to need business logic layer, maybe some service layer and data access layer.


Model
Updates
Manipulates
View
Controller
Sees
Uses
User

Model

- The Model contains only the pure application data, it contains no logic describing how to present the data to a user.
- It is just a data that is shipped across the application like for example from back-end server view and from front-end view to the database.
- In java programming, Model can be represented by the use of Plain-old-java-object (POJO) which is a simple java class.

Model

- The Model contains only the pure application data, it contains no logic describing how to present the data to a user.
- It is just a data that is shipped across the application like for example from back-end server view and from front-end view to the database.
- In java programming, Model can be represented by the use of Plain-old-java-object (POJO) which is a simple java class.

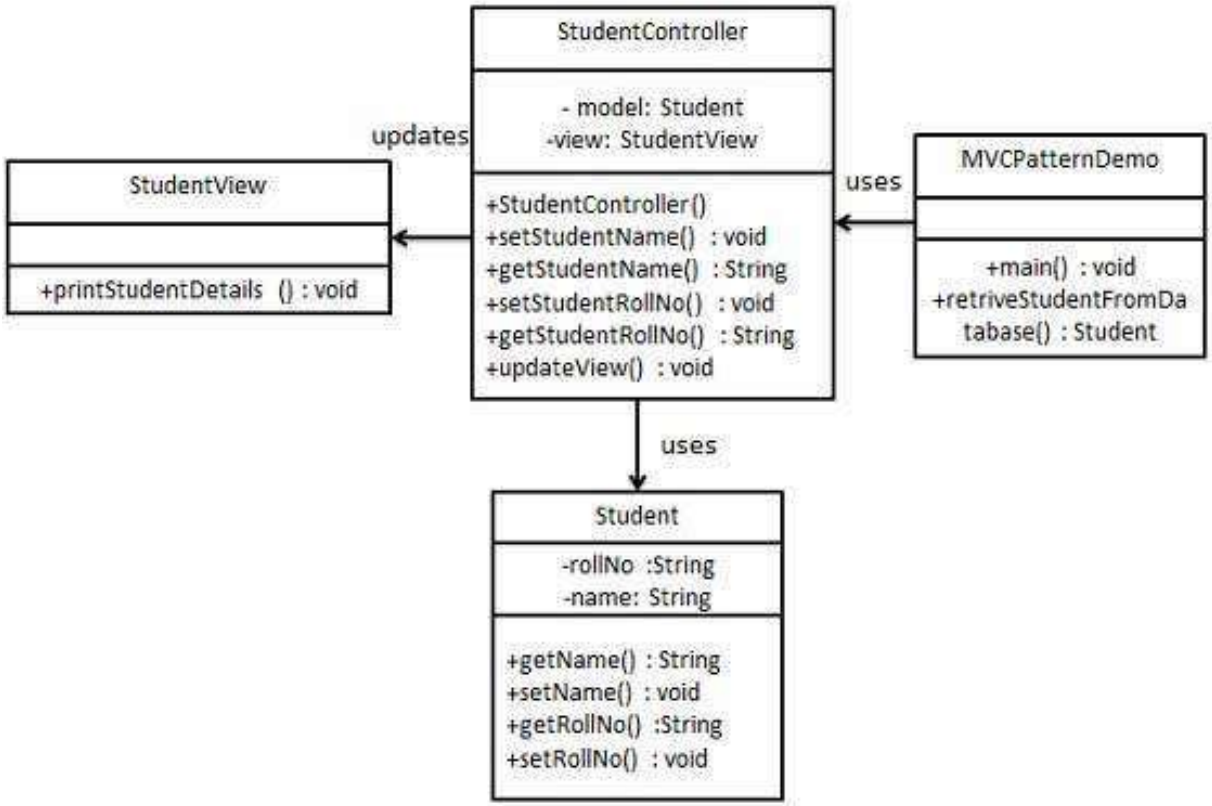
View

- The View presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- View just represent, displays the application's data on screen. View page are generally in the format of .html or .jsp in java programming (which is flexible).

Controller

- The Controller exists between the view and the model. It is where the actual business logic is written.
- It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model.
- Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

Class Diagram



Step 1
Create Model... *Student.java*

```java
public class Student {
   private String rollNo;
   private String name;

   public String getRollNo() {
      return rollNo;
   }

   public void setRollNo(String rollNo) {
      this.rollNo = rollNo;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

Step 2
Create View... *StudentView.java*

```
public class StudentView {
   public void printStudentDetails(String studentName, String studentRollNo){
      System.out.println("Student: ");
      System.out.println("Name: " + studentName);
      System.out.println("Roll No: " + studentRollNo);
   }
}
```

Step 3
Create Controller... *StudentController.java*

```
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 setStudentRollNo(String rollNo){
      model.setRollNo(rollNo);
   }
   public String getStudentRollNo(){
      return model.getRollNo();
   }
   public void updateView(){
      view.printStudentDetails(model.getName(), model.getRollNo());
   }
}
```

Step 4

Use the StudentController methods to demonstrate MVC design pattern usage. MVCPatternDemo.java

```
public class MVCPatternDemo {
   public static void main(String[] args) {

      //fetch student record based on his roll no from the database
      Student model  = retriveStudentFromDatabase();

      //Create a view : to write student details on console
      StudentView view = new StudentView();

      StudentController controller = new StudentController(model, view);

      controller.updateView();

      //update model data
      controller.setStudentName("John");

      controller.updateView();
   }

   private static Student retriveStudentFromDatabase(){
      Student student = new Student();
      student.setName("Robert");
      student.setRollNo("10");
      return student;
   }
}
```

Step 5
Output

```
Student:
Name: Robert
Roll No: 10
Student:
Name: John
Roll No: 10
```