MVC Pattern

514770-1
Fall 2021
11/29/2021
Kyoung Shin Park
Computer Engineering
Dankook University

MVC(Model-View-Controller) Pattern

- The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller.
- Separate business logic (Controller) and data representation (Model), and presentation part (View)
- Eliminate Model and View dependencies
  - Model can change regardless of View
  - View can also change regardless of model
- MVC is originally originated from the Smalltalk language, but is currently widely used in GUI applications and in web frameworks.
- MVC pattern is one of the most-used patterns from J2EE design pattern category.

MVC Pattern

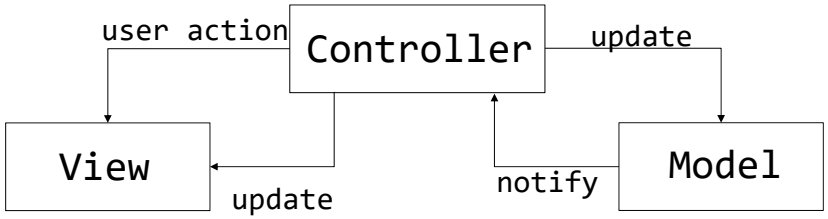
	Description
Pattern	MVC (Model-View-Controller)
Problem	Data and view code are mixed
Solution	Separate data and view and add a controller to link them
Result	Loose coupling, Reusability

Design

Role	Design
Model	Model is the part or logic that manages the application's data (there is only one model in the view)
View	View manages the part displayed on the screen that the user sees (there can be multiple views for the model)
Controller	Controller handles user input and supports interaction between Model and View (there may be more than one)

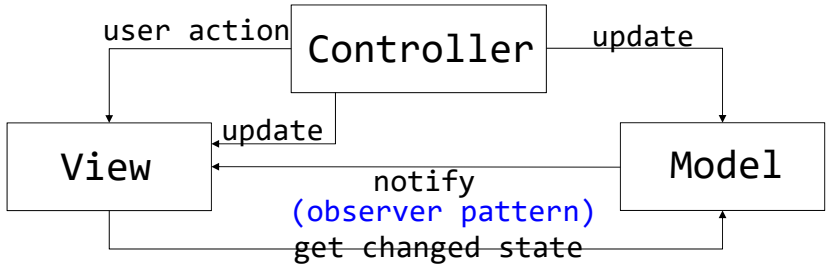
MVC Pattern

- Passive Model
  - Only the **controller** manipulates the **model**.
  - The controller modifies the model as the user types or makes input.
  - After the model has been modified, the controller asks to update the view.
  - The view receives the modified model data (after request) and updates it in the screen.



MVC Pattern

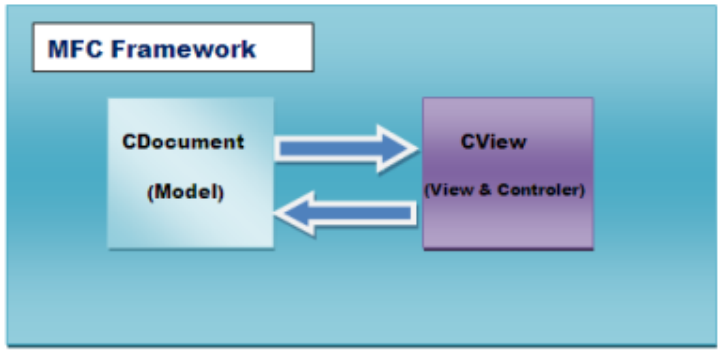
- Active Model
  - Controllers aren't the only way to modify the model.
  - The model can request to update the view.
  - The model provides the **subject** interface and registers as an **observer** in the view.
  - The view receives data from model (after request) and updates it in the screen.



MVC Pattern History

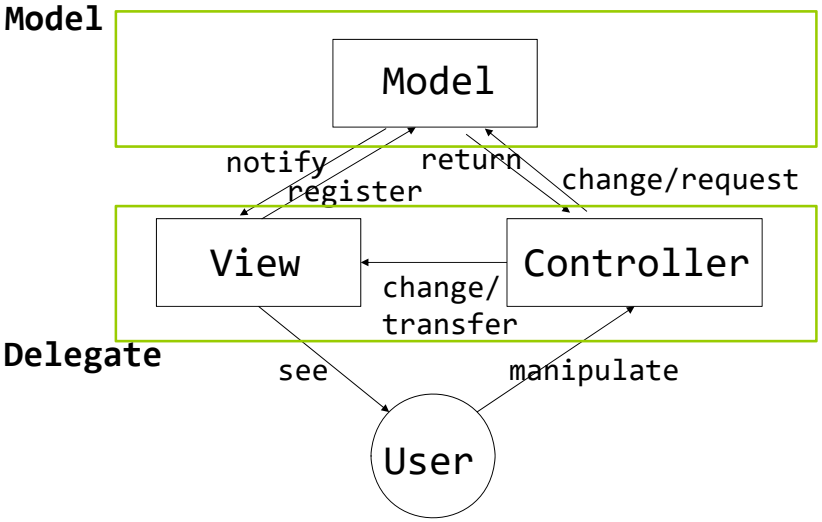
- Microsoft MFC
  - Use the Document/View structure
  - Document represents Model, Controller is processed by Windows messaging system.
- Java Swing
  - Use Model/Delegate structure
  - Delegate can be though of as Controller + View.
  - Java Swing components consist of Model and Delegate.
    - Each component provides a default model and delegate providing the basic functions
    - Model or delegate can be changed using setModel() and setUI() methods.

MVC Pattern (MFC)



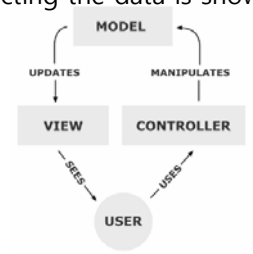
Document(Model) **View(View & Controller)**

MVC Pattern (Swing)

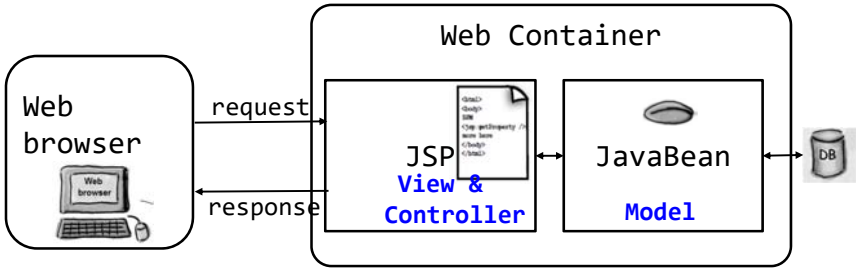


MVC Pattern (Web)

- Quotes from https://opentutorials.org/course/697/3828
  1. User accesses the website. (use)
  2. The Controller calls the Model to service the web page requested by the user. (manipulate)
  3. The Model controls a data source such as a database or file and then returns the results.
  4. The Controller reflects the results returned by the Model to the View. (update)
  5. The View reflecting the data is shown to the user. (see)

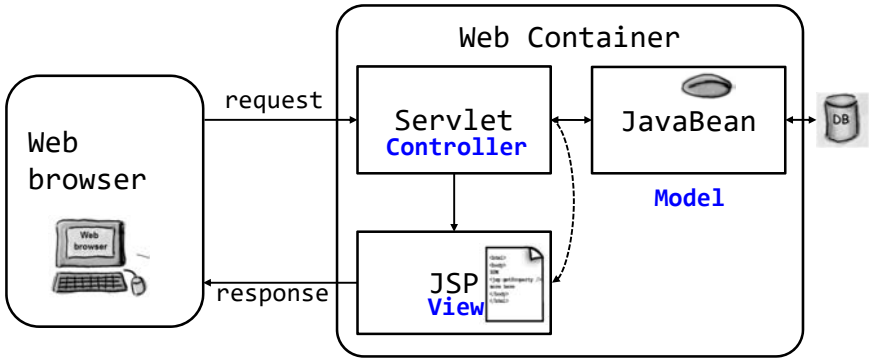


MVC Pattern (Model1)



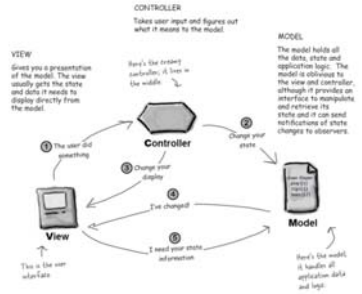
MVC Pattern (Model2)

- Model2 is an adaptation of MVC to the Web

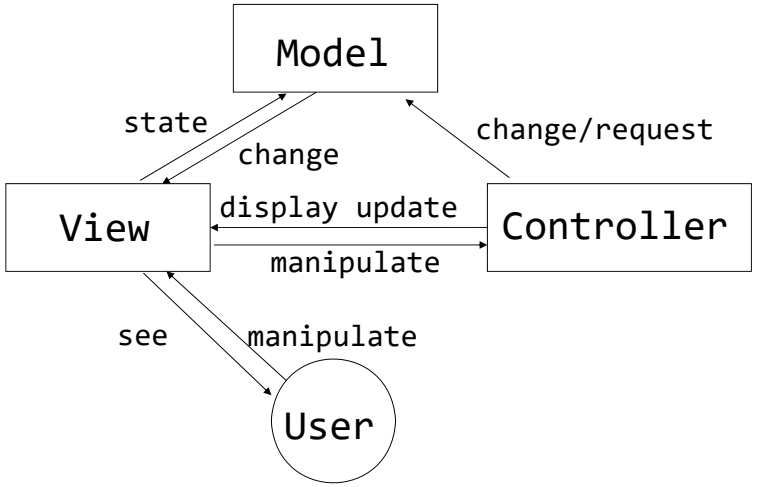


MVC Pattern (HFDP Ch12)

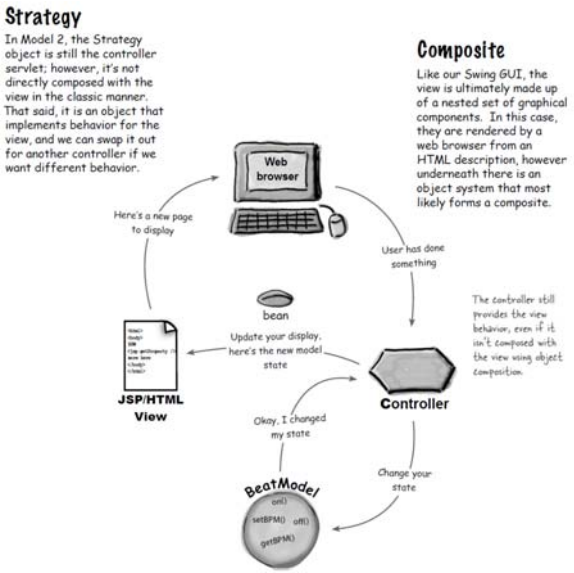
1. User interact with the View
2. The Controller asks the Model to change its state.
3. The Controller may also ask the View to change.
4. The Model notifies the View when its state has changed.
5. The View asks the Model for state.



MVC Pattern (HFDP Ch12)

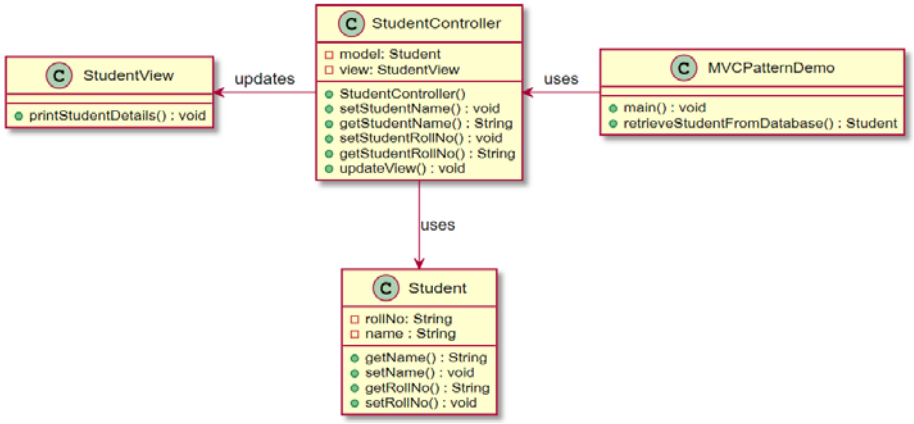


MVC Pattern (Model2 - HFDP Ch12)



Example

- https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm



Example

```
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;
   }
}
```

Example

```
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);
  }
}
```

Example

```
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();
  }
```

Example

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

Example

```
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();
 }
```

Example

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

MVC Pattern Advantage and Disadvantage

- Advantage
  - Object-oriented structure that minimizes information sharing between classes
    - Model and View don't have to know each other well.
  - Multiple Views can be supported in the same model
- Disadvantage
  - May be inefficient
    - When the View is notified that it should be updated, the View receives information from the Model and updates it.
    - It is more efficient for the Model to directly communicate what the View needs, but it is not object-oriented.
    - The role of the Controller can be too large.