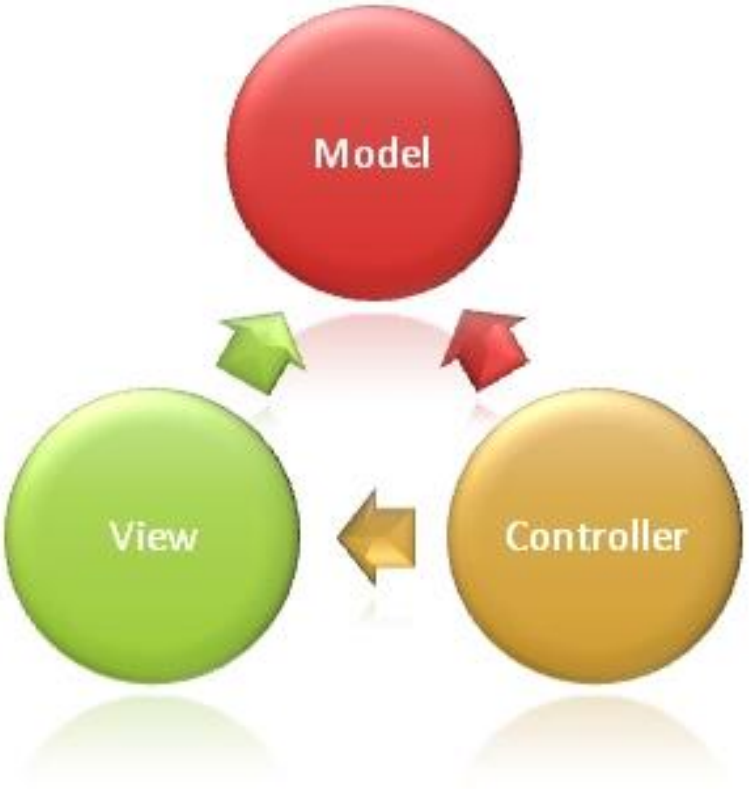
Introduction

- **Model–View–controller** (**MVC**) is a software architecture pattern which separates the representation of information from the user's interaction with it .

- First used in the Smalltalk-80 framework
  - Used in making Apple interfaces (Lisa and Macintosh

Parts of MVC



The MVC model defines web applications with 3 logic layers:

The business layer (Model logic)

The display layer (View logic)

The input control (Controller logic)

Model

- The model is responsible for managing the data of the application.
- It responds to the request from the view and it also responds to instructions from the controller to update itself
- The Model represents the application core (for instance a list of database records).
- It is also called the ***domain layer***

View

- The View displays the data.
- A **view** requests information from the model, that it needs to generate an output representation.
- MVC is often seen in web applications, where the view is the HTML page.

Controller

- **The Controller** is the part of the application that handles user interaction.
- Typically controllers read data from a view, control user input, and send input data to the model.
- It handles the input, typically user actions and may invoke changes on the model and view.

CONTROLLER

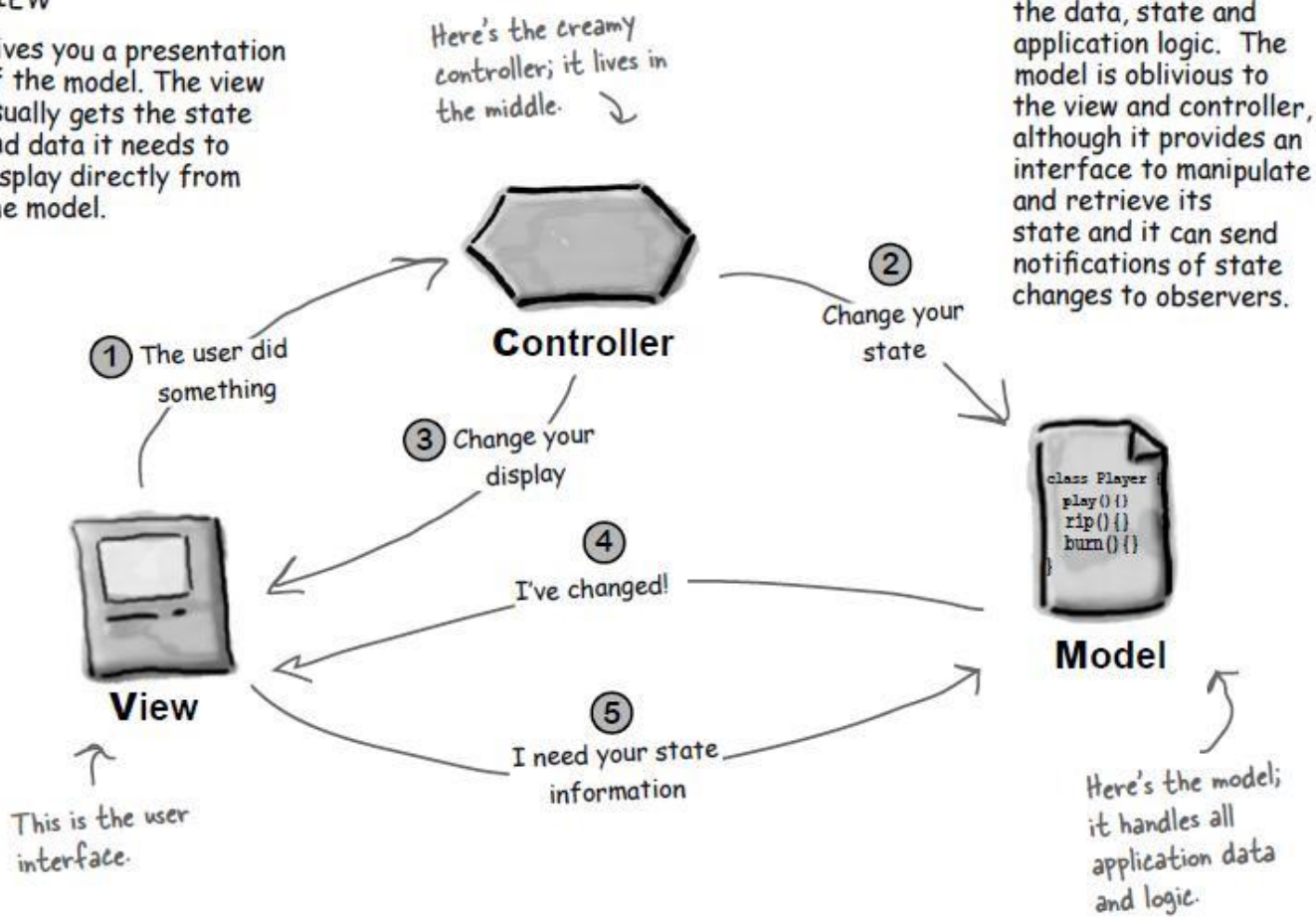
Takes user input and figures out what it means to the model.

MODEL

The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers.

VIEW

Gives you a presentation of the model. The view usually gets the state and data it needs to display directly from the model.



Workflow in MVC

Though MVC comes in different flavours, the control flow generally works as follows:

1. The user interacts with the user interface in some way (e.g., user presses a button)
2. A controller handles the input event from the user interface, often via a registered handler or callback.
3. The controller accesses the model, possibly updating it in a way appropriate to the user's action (e.g., controller updates user's shopping cart).

Workflow in MVC

4. A view uses the model to generate an appropriate user interface (e.g., view produces a screen listing the shopping cart contents).

   The view gets its own data from the model. The model has no direct knowledge of the view.

5. The user interface waits for further user interactions, which begins the cycle anew.

Dependence hierarchy

- There is usually a kind of hierarchy in the MVC pattern.
- The Model knows only about itself.
- That is, the source code of the Model has no references to either the View or Controller.

Dependence hierarchy

- The View however, knows about the Model. It will poll the Model about the state, to know what to display.
- That way, the View can display something that is based on what the Model has done.
- But the View knows nothing about the Controller.
- The Controller knows about both the Model and the View.

Dependence hierarchy

- Take an example from a game: If you click on the "fire" button on the mouse, the Controller knows what fire function in the Model to call.

- If you press the button for switching between first and third person, the Controller knows what function in the View to call to request the display change.

Why a dependence hierarchy?

- The reason to keep it this way is to minimize dependencies.
- No matter how the View class is modified, the Model will still work.
- Even if the system is moved from a desktop operating system to a smart phone, the Model can be moved with no changes.
- But the View probably needs to be updated, as will the Controller.

Use in web applications

- Although originally developed for personal computing, Model View Controller has been widely adapted as an architecture for World Wide Web applications in all major programming languages.
- Several commercial and noncommercial application frameworks have been created that enforce the pattern.
- These frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server