Web Presentation Patterns (controller)

SWEN-343

From Fowler, *Patterns of Enterprise Application Architecture*

Objectives

Look at common patterns for designing Web-based presentation layer behavior

- Model-View-Control
- Handling user input in HTTP request
- Delegating to domain layer for application processing
- Constructing HTML response stream
- Separating and coordinating these concerns in rich client applications with complex behavior

Web Presentation Layer Patterns

Model-View-Controller

Input Controller

- Page Controller
- Front Controller

Application Controller

View

- Template View
- Transform View
- Two Step View

Note: some of the following slides on presentation layer patterns are from a presentation by Martin Fowler. Previously accessible on the web.

Concerns

Handling stateless HTTP requests, network connections, etc.

- Scripts?

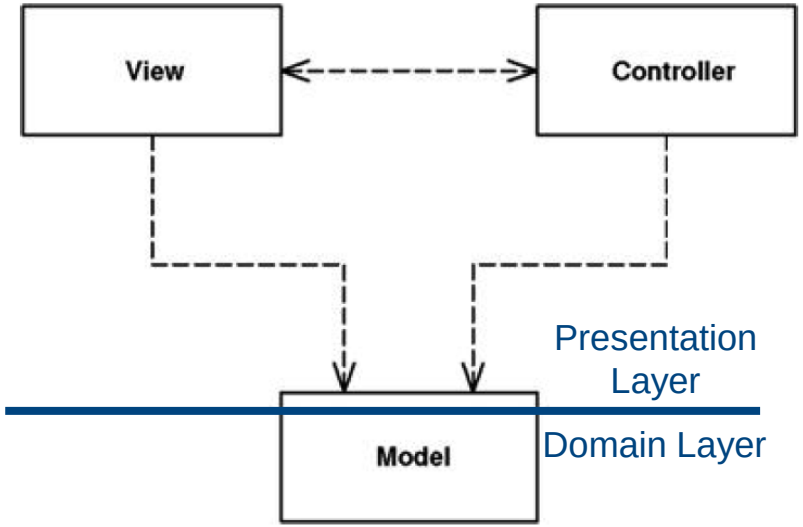
Creating complicated HTML response streams

- Server pages?

When there are complicated decisions based on input (such as display formats and what content to send next) and application state (what is happening in the underlying domain layer?)

Separate user interface from application behavior

Model View Controller



Model

The domain objects

View

Presents information to user

(Input) **Controller**

Handles user input

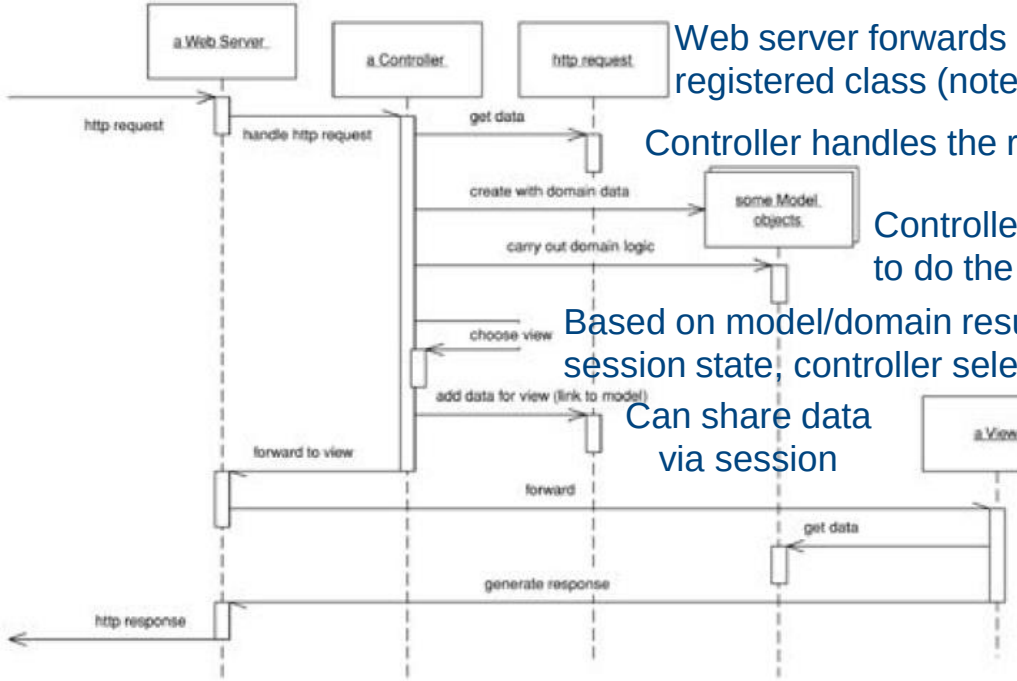
1. *Separation of Presentation (View/Controller) from Domain (Model)*
2. *Separation of View and Controller*
   - *Sometimes not necessary to separate*

MVC in a Web Server

There are many different meanings of “controller” in different contexts

In the context of Web applications, many people get the “Control” of MVC wrong

MVC in a Web Server – Basic Concept



Web server forwards request to registered class (note: asynchronous)

Controller handles the request

Controller gets the model to do the domain logic

Based on model/domain results and user session state, controller selects next view

Can share data via session

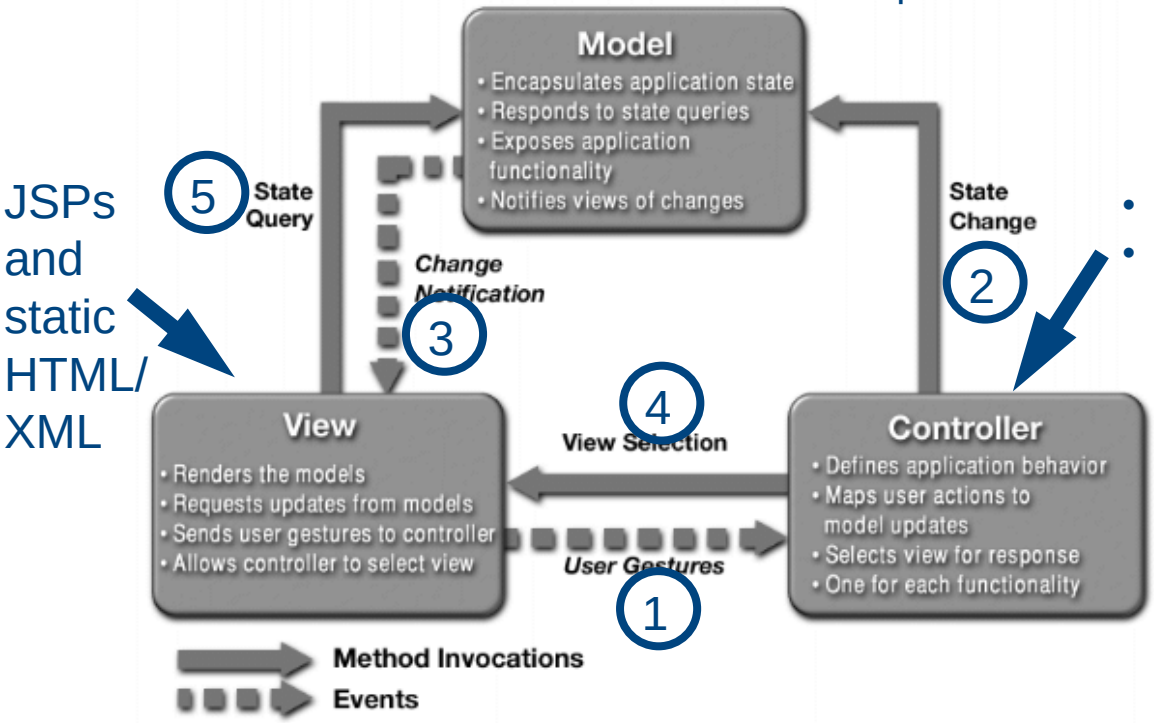
Controller gets the view to create a response based on the model (and data shared in session)

(Input) Controller

- An HTTP request comes into the input controller
- It pulls information off of the request
- It forwards the business logic request to an appropriate model object
- The model object talks to the data source and other domain layer objects and does everything indicated by the request as well as gather information for the response
- When the model object is done, it returns control to the input controller
- The input controller looks at the results and decides which view is needed to display the response
- The input controller passes control, together with the response data, to the view
  - Often the response data is left in an agreed location, such as an HTTP session object

MVC Structure in Java EE

- Application Tier (EJBs, POJOs, etc.)
- The Web Tier will have servlets, Java Beans, POJOs, etc. that are proxies for or adapters to the underlying application tier



JSPs and static HTML/ XML

- Servlets
- Centralized controller object
  - Such as Front Controller for screen flow manager

Input Controllers

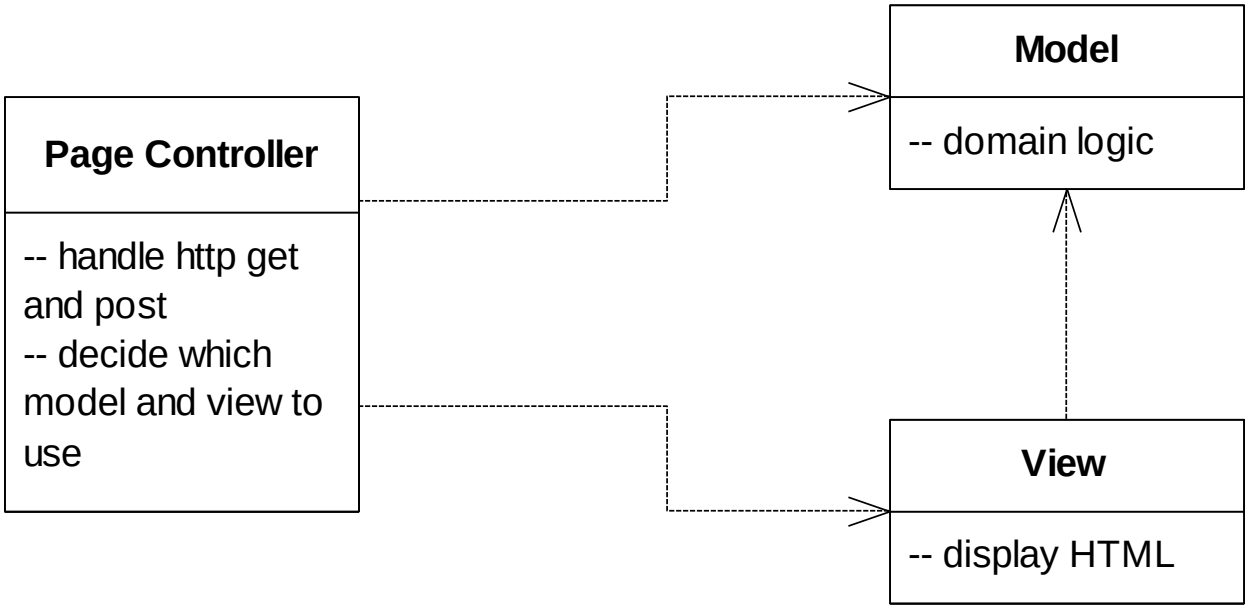
- Input controller responsibilities
  - Handle the HTTP request
  - Decide what to do with it
  - Delegate to model objects to actually do the work
  - Select next view and pass control to view

- Example: Server page handles request, then delegates to a separate helper object to decide what to do with it
- Example: Front controller is a single object that handles all requests
  - Interprets the URL to figure out what kind of request it is dealing with, then creates a separate object to process it

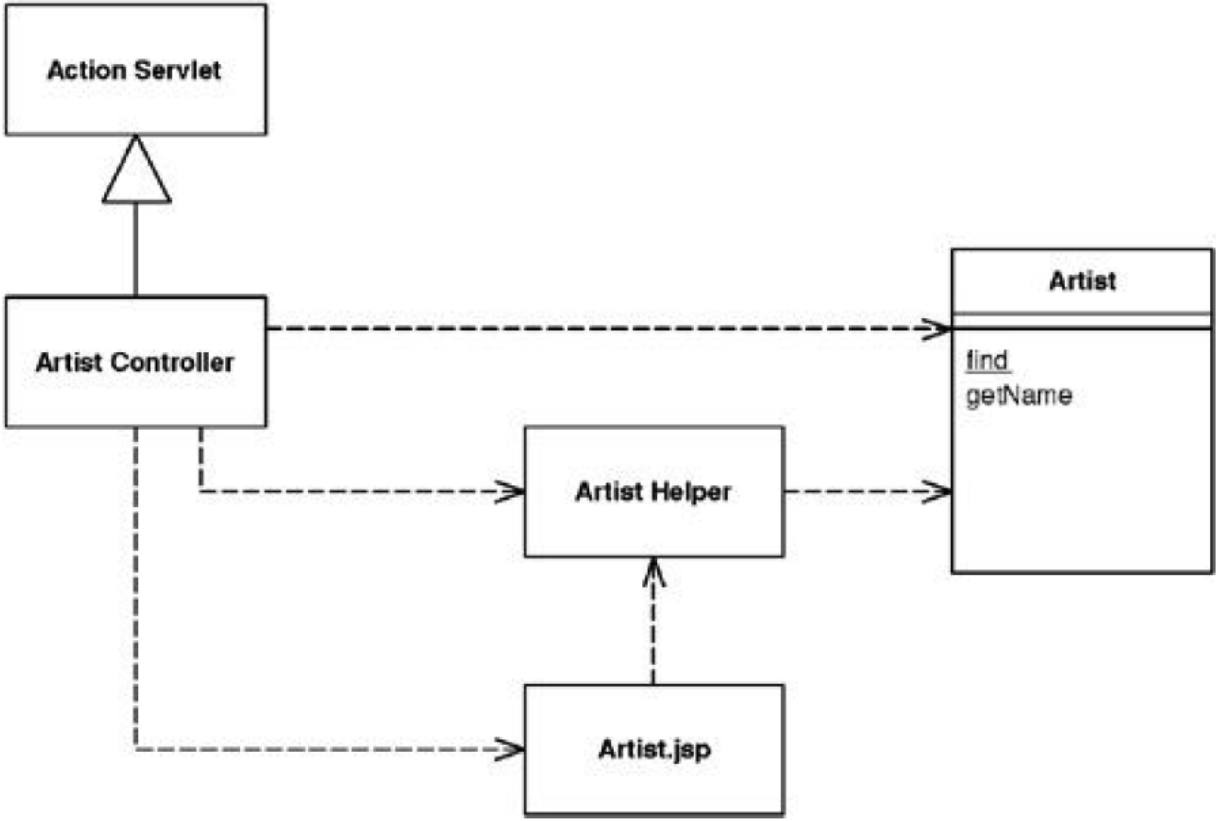
Page Controller Input Controller

- One controller per Web page
  - May be a server page that combines view and control (but should separate view and control in the page code)
  - Input controller creates appropriate models to do the processing, then instantiates a view to return the result
- One controller per action (action: a button or navigation link)

Page Controller Input Controller



Example: Servlet Controller, JSP View



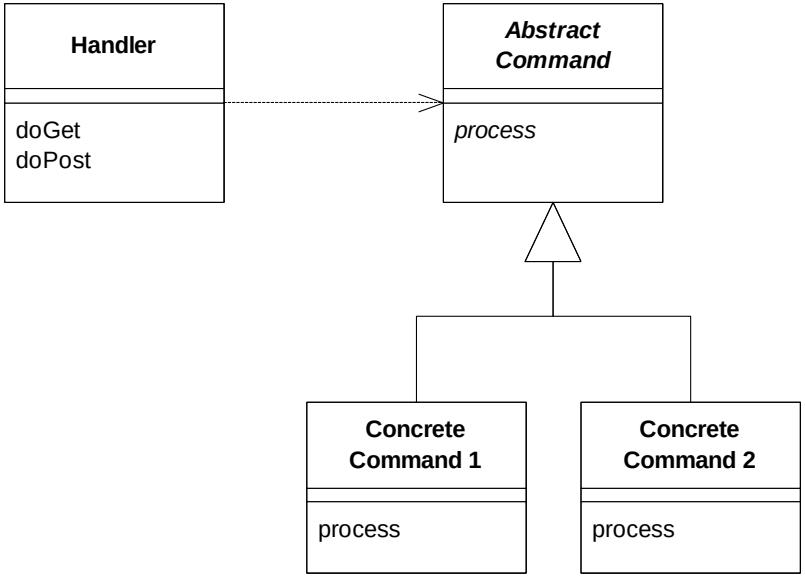
Front Controller Input Controller

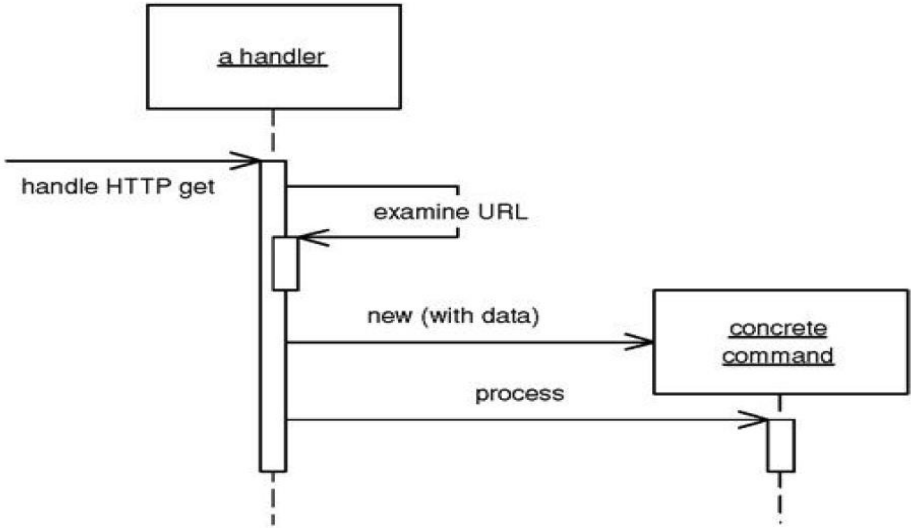
Consolidate control behavior

All request handling is channeled through a single handler object

Removes duplicate behavior, such as security or internationalization, spread across multiple page controllers

Front Controller Input Controller





- Commands can be plain old objects
- Combine with Intercepting Filter pattern (Decorators) to wrap the handler with a filter chain to handle authentication, logging, locale identification, etc.

Front vs. Page Controllers

Page Controller

- Input controller per page is easy to follow
- Don’t put controller logic in scriptlets
- Use separate class

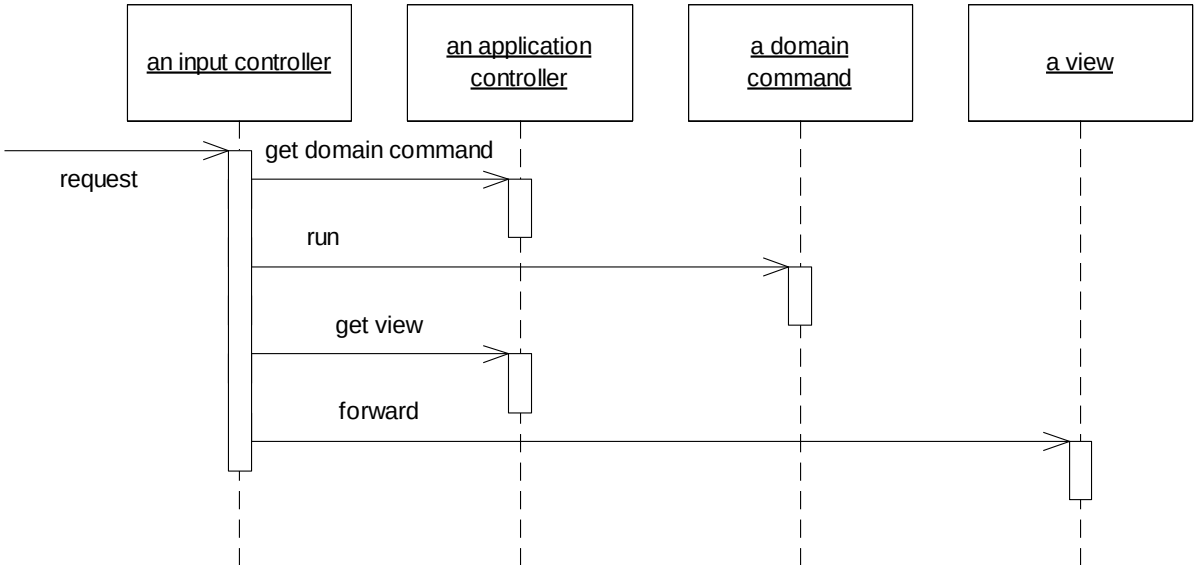
Front Controller

- Single point for adding behavior
- Can add behavior dynamically without changing the Web handler

Application Controllers (Distinct from Input Controllers)

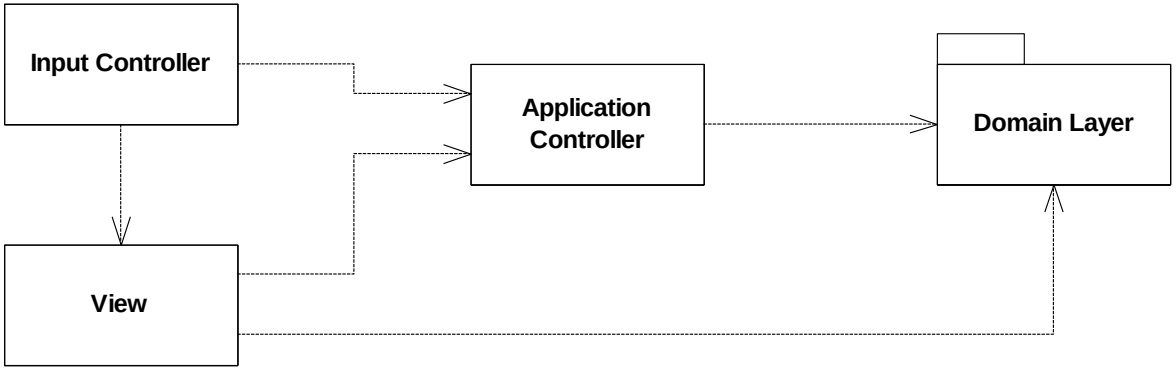
- Application controllers handle the flow of an application
  - Coordinate user conversation in a session with application flow
  - **Input controllers ask the Application Controller** for the appropriate commands for execution against a model and the correct view to use depending on the application context
  - For example, manage complicated logic of screen flow and navigation
  - For example, when there is not a simple mapping between pages and actions in the domain
- Part of presentation layer or a separate layer that mediates between the presentation layer and the domain layer

Application Controller



A centralized point for handling screen navigation and flow of an application

Application Controller: Dependencies



Note that, from the Input Controller's perspective, the Application Controller plays the role of Model in MVC

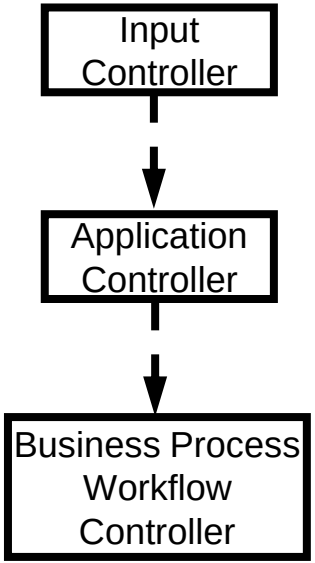
Is an Application Controller Necessary?

A test for needing an application controller:

- If the user is in control of the order of screen navigation, you don’t need an application controller
- If the computer is in control of the screen flow, you need an application controller

“Model” in MVC and Application Controllers and Input Controllers

- The most important reason to use MVC is to completely separate the model (domain behavior) from the Web presentation
- Input controllers receive HTTP requests and decide what to do
- The model behavior (the “what to do”) can also be complicated, requiring Application Controllers to handle the flow of an application and screen navigation
- Handle the flow of user tasks (“dialog controller”), what screens should appear, etc.
- Further, the Domain Layer can have application workflow controllers (use-case controllers, business process choreography, transaction controllers, etc.)



To Control or Not To Control? And Where?

- There will always be an input controller (at a minimum, to handle http request)
  - Even if it is a scriptlet in a server page (which is a bad idea)
- Not all presentation layers need an application controller
- If the user does not need to see the flow of control among business objects and/or external applications, provide a domain-layer control separate from a presentation-layer application control
- Suggestion: Figure out what has to be controlled and provide a separate object to control it
  - Be willing to provide multiple controllers across layers
  - Don't get hung up on type(s) of control and which layer(s) are in control, and don't be overly constrained by the "assumed" model of your development tools and run-time platform/container

Conclusions

- It is imperative to separate user interface (view, control) from application (model)
  - Especially when the application behavior is executed in the domain layer (as it should be!)
- Input Controller patterns and Application Controller pattern help manage complicated user session flow control and selecting and delegating to appropriate model behavior
  - Domain layer may have additional controllers to manage the flow of application logic, but these are independent of any presentation-oriented controllers
- Read the pattern documentation in Fowler for detailed examples with code in Java and C#