# Pola Desain Perangkat Lunak

[Week] 12 – Architectural Pattern, Model View Controller (MVC)

Prepared by: Tifanny Nabarian

# Review - Chain of Responsibility

- In this pattern, you **form a chain of objects** where each object in the chain **handles a particular kind of request**.
- If **an object cannot handle** the request fully, **it passes the request to the next object** in the chain. The same process may follow until the end of a chain is reached.

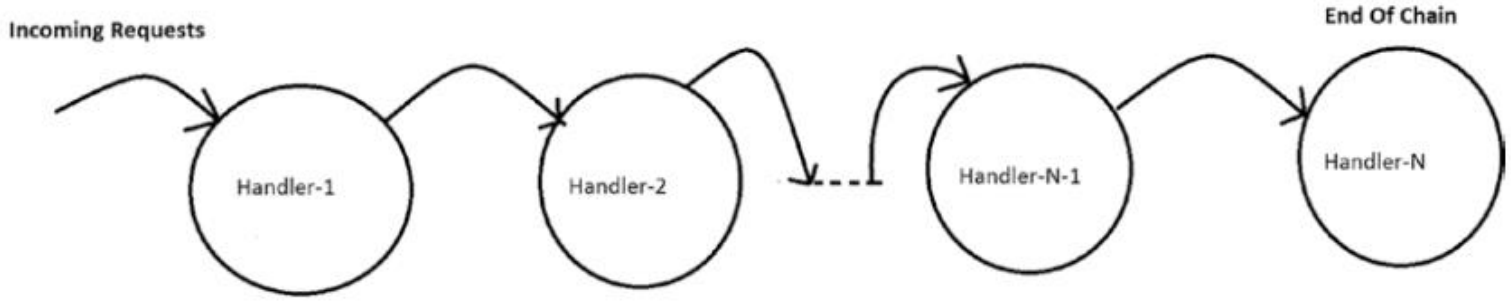
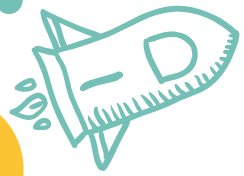



***Figure 22-1.*** *The concept of a chain-of-responsibility pattern*



# Architectural Pattern

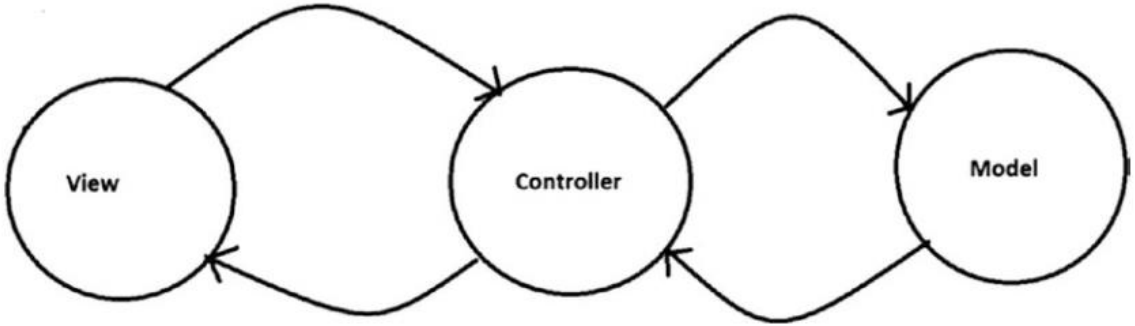
## Model View Controller

# Design Patterns Category

- Model-View-Controller (MVC) is an **architectural pattern**.
- The use of this pattern is commonly seen in web applications or when we develop powerful user interfaces. But it is important to note that Trygve Reenskaug first described MVC in 1979 in a paper titled, "*Applications Programming in Smalltalk-80TM: How to Use Model-View-Controller*," which was before the World Wide Web era. At that time, there was no concept of web applications.
- But modern-day applications can be seen as an adaptation of that original concept.
- It is important to note that some developers believe that it is not a true design pattern, instead, they prefer to call it **"MVC architecture."**

# MVC Concept

- From this introduction, it is apparent that the pattern consists of the **three major components: Model, View, and Controller**.
- **Controller** is **placed between View and Model** in such a way that Model and View can communicate to each other only through Controller.
- Here you **separate** the mechanism of **how data is displayed** from the mechanism of **how the data is manipulated**.



# MVC Concept (cont'..)

- **View represents the output. It is a presentation layer**. Think of it as a user interface/GUI. You can design it with various technologies. For example, in a .NET application, you can use HTML, CSS, WPF, and so forth.
- **Model is the brain of your application**. It manages the data and the business logic. It knows how to store and manage (or manipulate) the data, and how to handle the requests that come from controller. But this component is separated from the View component. A typical example is database, a file system, or similar kinds of storage. It can be designed with JavaBeans, Oracle, SQL Server, DB2, Hadoop, MySQL, and so forth.
- **Controller is the intermediary that accepts users input** from the View component and passes the request to the model. When it gets a response from the model, it passes the data to the view. It can be designed with C#.NET, ASP.NET, VB.NET, Core Java, JSP, servlets, PHP, Ruby, Python, and so forth.

# MVC Concept

## Implementation Variation

- ✓ You can have multiple views.
- ✓ Views can pass runtime values (e.g., using JavaScript) to controllers.
- ✓ Your controller can validate the user's input
- ✓ Your controller can receive input in various ways. For example, it can get input from a web request via a URL, or you can pass the input by pressing a Submit button on a form.
- ✓ In some applications, Model components can update the View component.

# MVC Concept - Variation

## Variation 1

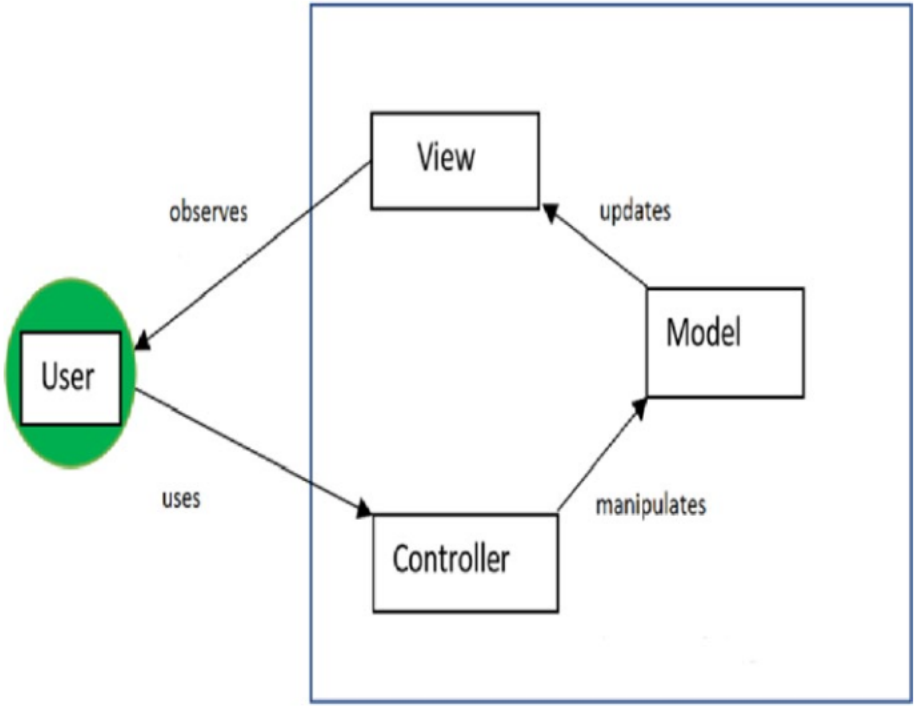


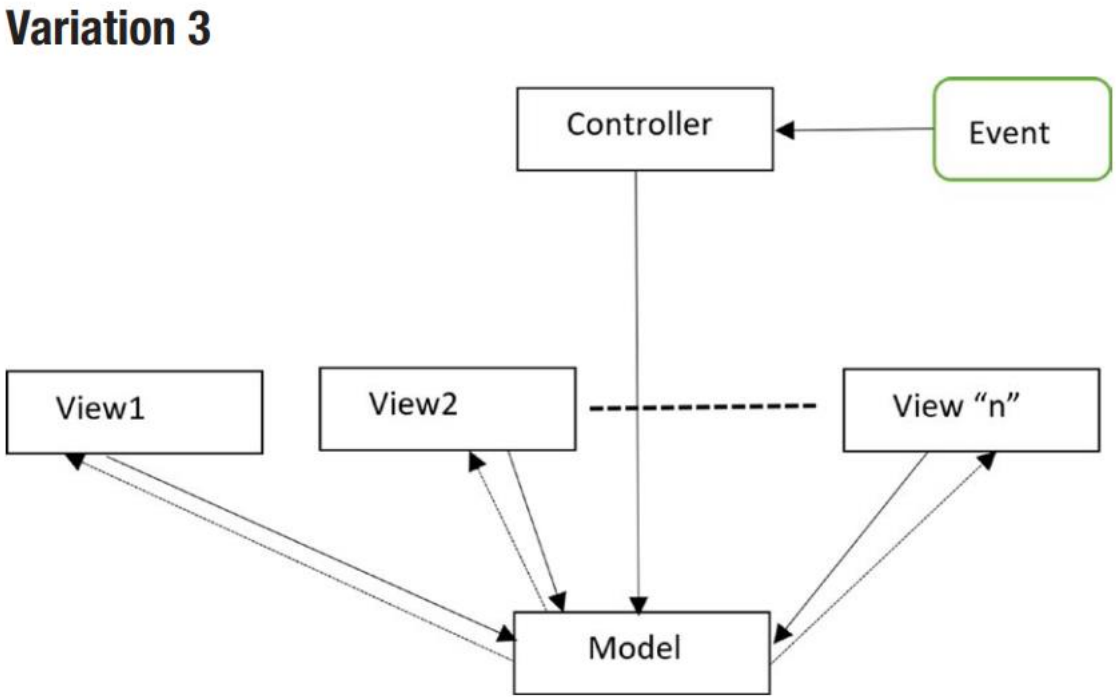

***Figure 26-2.*** *A typical MVC framework*

# MVC Concept - Variation

## Variation 2


***Figure 26-3.*** *A MVC framework with multiple views*

# MVC Concept - Variation



# MVC Concept

## Real World Example

- I said that in a restaurant, based on customer input, a chef can vary the taste and make the final products. **The customers do not place their orders directly to the chef**.
- **The customers see the menu card (view), may consult with the waiter/waitress**, and place their order**. The waiter/waitress passes the order slip to the chef**, who gathers the required materials from the restaurant's kitchen (similar to storehouses/computer databases). Once prepared, the waiter/waitress carries the plate to the customer's table.
- So, you can consider **the role of the waiter/waitress as the controller, and the chef with their kitchen as the model (and the food preparation materials as data).**

# MVC Concept

## Computer World Example

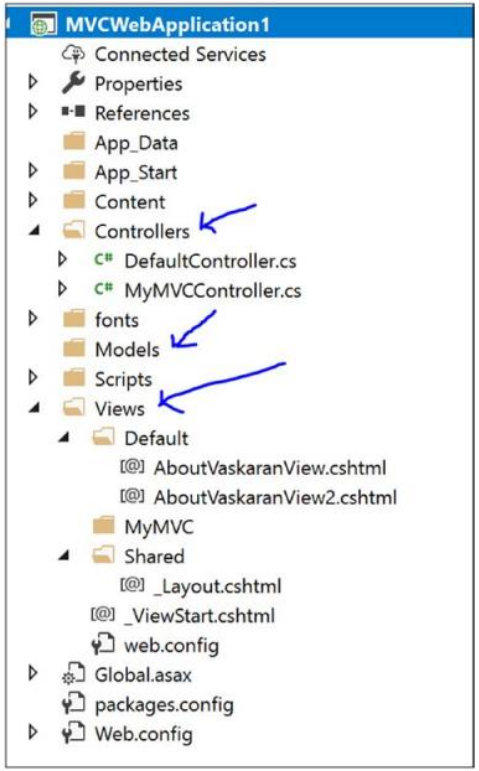



***Figure 26-5.*** *A typical MVC structure in a ASP.NET project*

# Let's Practice!

# Illustration

- In this application, the requirement is very simple. There are employees who need to register themselves in an application/ system. Initially, the application starts with three different registered employees: Amit, Jon, and Sam. At any time, you should be able to see the enrolled employees in the system.
- You can add a new employee or delete an employee from the registered employees list.
- A simple check is added in the Employee class to ensure that you are not adding an employee repeatedly in the application.
- To delete an employee from the registered list, you need to pass the employee ID in the client code, but the application will do nothing if an employee ID is not found in the registered list.

# Class Diagram

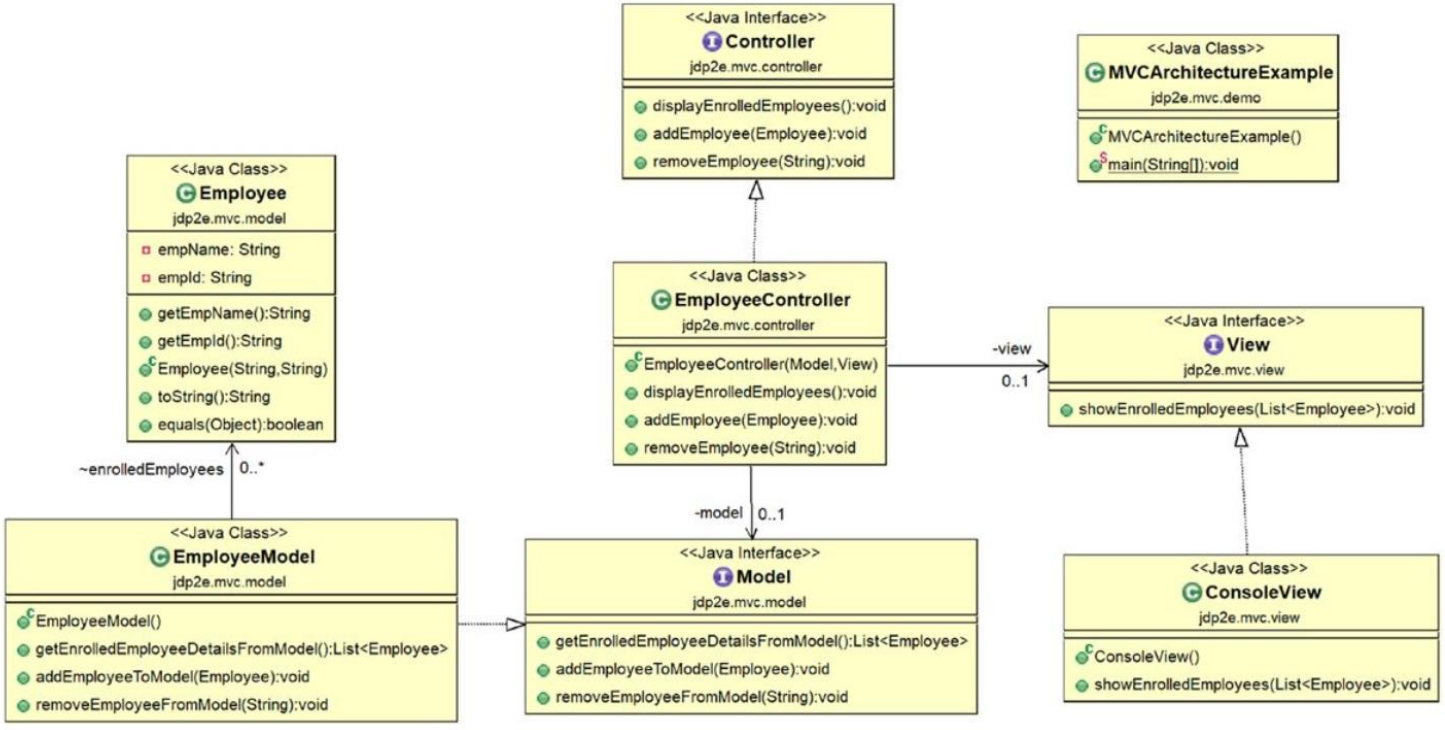
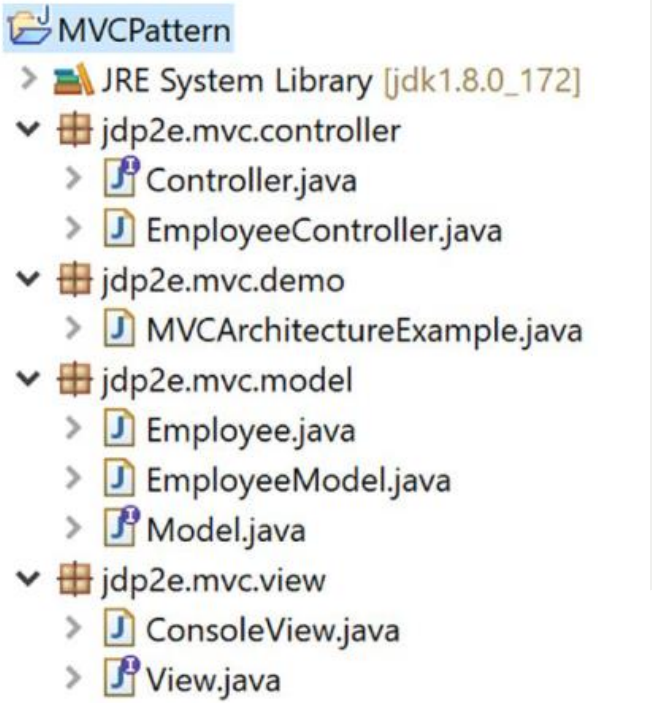



***Figure 26-6.*** *Class diagram*

# MVC Example (Package Explorer)



**Silahkan Kerjakan**
**Tugas Praktikkum..**