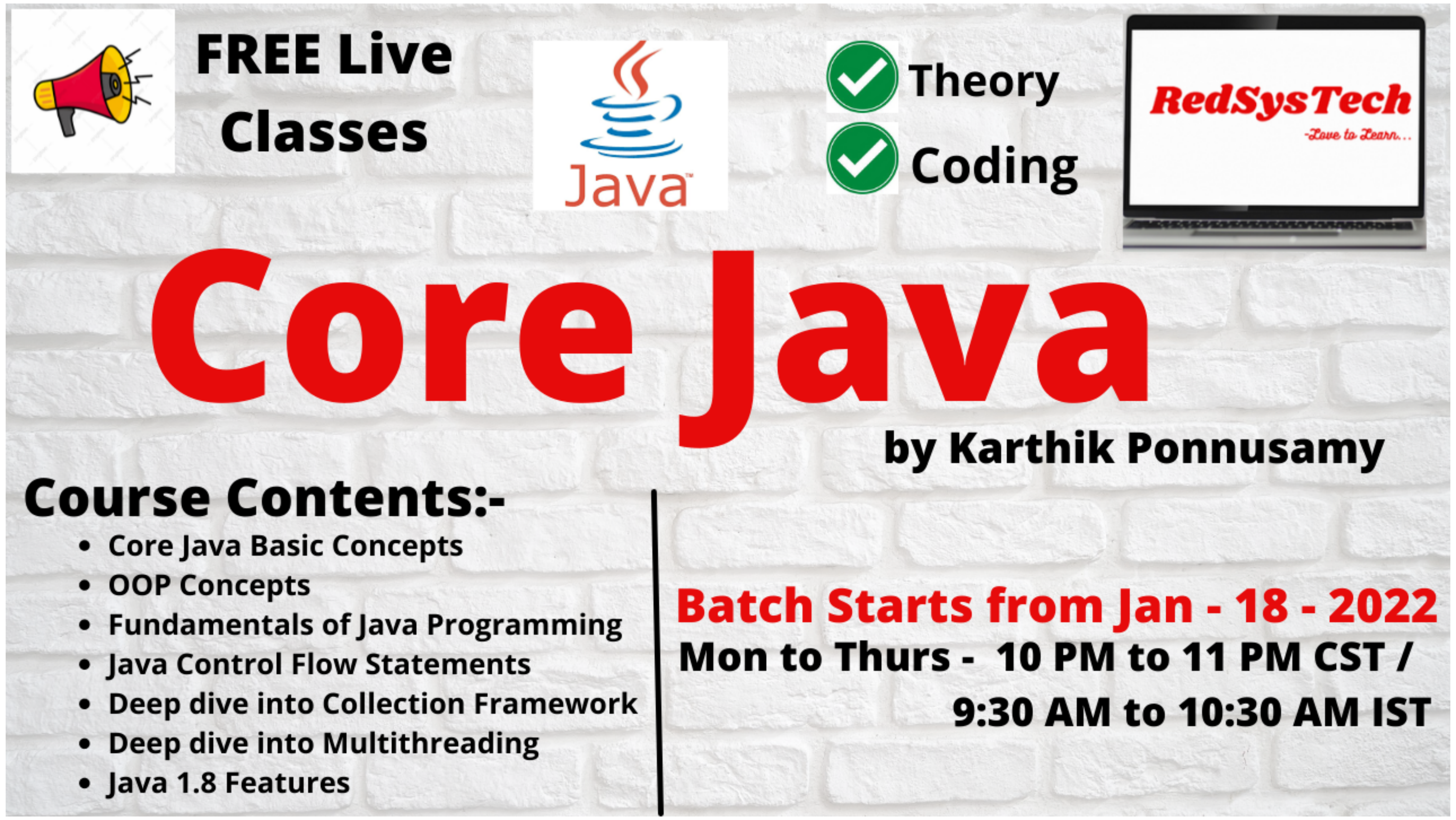


FREE Live Classes

Java

✓ Theory

✓ Coding

RedSysTech -Love to Learn...

Core Java

by Karthik Ponnusamy

Course Contents:-

- Core Java Basic Concepts
- OOP Concepts
- Fundamentals of Java Programming
- Java Control Flow Statements
- Deep dive into Collection Framework
- Deep dive into Multithreading
- Java 1.8 Features

Batch Starts from Jan - 18 - 2022

Mon to Thurs - 10 PM to 11 PM CST / 9:30 AM to 10:30 AM IST

Day 03: Agenda

MVC Architecture in Java

MVC - What is Model, View and Controller?

How MVC Architecture works?

Implementation of MVC using Java

Advantages of MVC Architecture

MVC - What is Model, View and Controller?

MVC stands for Model-View-Controller, which is a design pattern used to separate the application logic into three interconnected components: the Model, the View, and the Controller.

Model: The Model represents the data and the business logic of the application. It is responsible for managing the data, validating it, and providing methods for accessing and manipulating it.

View: The View is responsible for displaying the data to the user in a user-friendly format. It can be any type of output representation, such as a web page, a graphical user interface, or a mobile application.

Controller: The Controller acts as an intermediary between the Model and the View. It receives user input from the View, processes it, and updates the Model as necessary. It also updates the View based on changes in the Model.

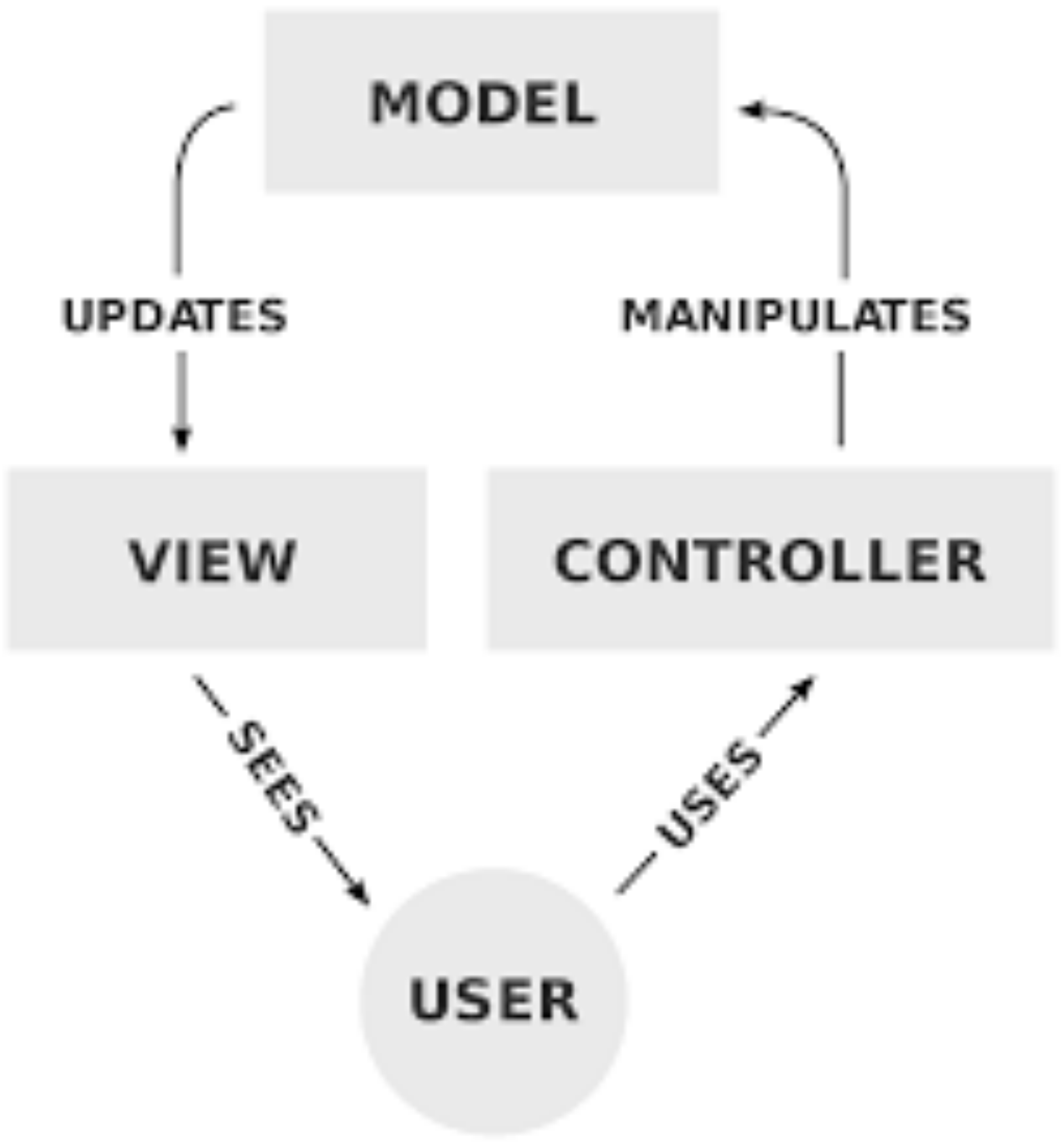
The main idea behind the MVC pattern is to separate the concerns of each component, allowing them to be developed and maintained independently. This results in a more modular and scalable architecture, where changes made to one component do not affect the others.

In a typical MVC application, the user interacts with the View, which sends input data to the Controller. The Controller processes the input data, updates the Model accordingly, and sends the updated data to the View for display. Any changes made to the Model are reflected in the View automatically.

For example, in a web application, the Model might represent the data stored in a database, the View might be a web page that displays the data, and the Controller might be a PHP script that processes the user input and updates the database.

Overall, the MVC pattern is a widely used architecture for building complex applications, providing a structured and organized way to manage the application logic and user interface.

How MVC Architecture works?



Implementation of MVC using Java

Here's a simple example of implementing the MVC pattern using Java:

Model:

```
public class Person {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

Implementation of MVC using Java

View:

```
public class PersonView {
    public void printPersonDetails(String name, int age){
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Controller:

```
public class PersonController {
    private Person model;
    private PersonView view;

    public PersonController(Person model, PersonView view) {
        this.model = model;
        this.view = view;
    }

    public void setPersonName(String name) {
        model.setName(name);
    }

    public void setPersonAge(int age) {
        model.setAge(age);
    }

    public void updateView() {
        view.printPersonDetails(model.getName(), model.getAge());
    }
}
```

Implementation of MVC using Java

Main Class:-

```
public class Main {
    public static void main(String[] args) {
        // Create a new Person object
        Person person = new Person();

        // Create a new PersonView object
        PersonView personView = new PersonView();

        // Create a new PersonController object
        PersonController personController = new PersonController(person, personView);

        // Update the person's details
        personController.setPersonName("John Doe");
        personController.setPersonAge(30);

        // Update the view
        personController.updateView();
    }
}
```

Implementation of MVC using Java

In this example, the Person class represents the Model, the PersonView class represents the View, and the PersonController class represents the Controller. The main class creates a new Person object, a new PersonView object, and a new PersonController object. It then updates the person's details using the PersonController object and updates the view using the updateView() method of the PersonController object.

Overall, this example shows how the MVC pattern can be implemented in Java to separate the application logic and user interface into three distinct components.

Advantages of MVC Architecture

Advantages of the MVC pattern:

Separation of Concerns: The MVC pattern separates the concerns of the application into three distinct components, each with its own responsibilities. This makes the code more organized and easier to maintain.

Reusability: Since each component of the MVC pattern is separate from the others, they can be reused in other parts of the application or even in other applications.

Testability: The separation of concerns makes it easier to write unit tests for each component of the MVC pattern. This helps to ensure that the application is functioning correctly and that changes to one component do not affect the others.

Flexibility: The MVC pattern provides a flexible architecture that can be adapted to a wide range of applications and user interfaces.

Disadvantages of the MVC pattern:

Complexity: The MVC pattern can be more complex than other patterns, especially for small applications. It requires additional code and layers of abstraction, which can be difficult to understand for beginners.

Overhead: The separation of concerns in the MVC pattern can lead to additional overhead in terms of development time and code complexity.

Learning Curve: Developers who are not familiar with the MVC pattern may find it challenging to learn and implement in their applications.

Lack of Transparency: The MVC pattern can be less transparent than other patterns, making it harder to follow the flow of data and logic between the components. This can make it more difficult to debug and maintain the code.

Overall, the MVC pattern provides a well-defined structure for building complex applications, but it may not be suitable for every application. Developers should consider the complexity, flexibility, and overhead of the pattern before deciding to use it in their applications.

WHAT?
WHY?
WHEN?
WHERE?
HOW?
WHO?