**Zoya kayani**

**Fa22-bse-042**

**MVC MODEL**

**CODE**

public class Main {

public static void main(String[] args) {

Model model = new Model();

View view = new View();

Controller controller = new Controller(model, view);

controller.setData("Hello, MVC!");

controller.updateView();

}

}

// Model class

class Model {

private String data;

public String getData() {

return data;

}

public void setData(String data) {

this.data = data;

}

}

// View class

class View {

public void printData(String data) {

System.out.println("Data: " + data);

}

}

// Controller class

class Controller {

private Model model;

private View view;

public Controller(Model model, View view) {

this.model = model;

this.view = view;

}

public void setData(String data) {

model.setData(data);

}

public String getData() {

return model.getData();

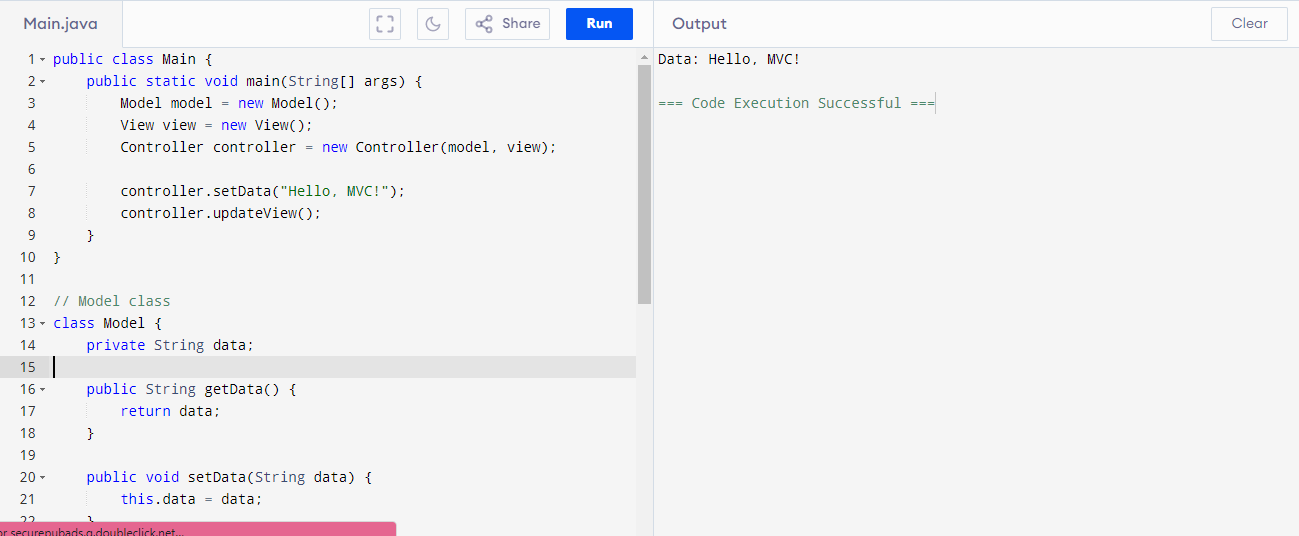
}

public void updateView() {

view.printData(model.getData());

}

}



**Introduction to Technology Stack for MVC Design**

The **MVC (Model-View-Controller)** design pattern is a widely adopted architectural pattern that separates the application into three interconnected components: **Model**, **View**, and **Controller**. This separation improves code maintainability, scalability, and readability.

Below is an overview of the technology stack commonly used to implement the MVC pattern in Java applications.

**Technology Stack Overview**

**1. Programming Language: Java**

* Java is a versatile, platform-independent, and object-oriented programming language.
* Ideal for building scalable and maintainable MVC-based applications.

**2. Build Tools:**

* **Maven**: A build automation tool used for dependency management and project configuration.
* **Gradle**: A flexible build tool often used for complex projects.

**3. Development Environment:**

* **IDE**:
  + Examples: IntelliJ IDEA, Eclipse, or Visual Studio Code.
  + Provides features like code completion, debugging, and project organization.
* **Text Editors**:
  + Examples: Sublime Text, Atom.
  + Lightweight alternatives for small projects.

**4. Runtime Environment:**

* **Java Development Kit (JDK)**: Required to compile and run Java applications.
* **Java Virtual Machine (JVM)**: Executes Java bytecode on different platforms.

**5. Presentation Layer (View):**

* **Java Swing**: Used for desktop applications with GUI.
* **JavaFX**: A modern toolkit for building visually rich desktop applications.
* **HTML, CSS, and JavaScript**: Used for web-based applications when combined with frameworks like Spring MVC.

**6. Controller and Backend Frameworks:**

* **Spring MVC**: A popular framework for building web applications.
* **Jakarta EE** (formerly Java EE): Provides specifications for building enterprise-grade applications.

**7. Database (Model Persistence):**

* **Relational Databases**: MySQL, PostgreSQL, Oracle Database.
* **NoSQL Databases**: MongoDB, Redis.
* **JPA (Java Persistence API)**: Simplifies interaction with databases.

**8. Testing Tools:**

* **JUnit**: A framework for unit testing Java applications.
* **Mockito**: Used for mocking dependencies during testing.

**9. Web Servers (For Deployment):**

* **Apache Tomcat**: A lightweight server for Java-based web applications.
* **Jetty**: Another lightweight server for web applications.
* **WildFly**: An enterprise server for running Jakarta EE applications.

**Real-World Use Case**

* **Desktop Applications**:
  + Using **Swing** or **JavaFX** for GUI (View).
  + Business logic in Model and coordination via Controller.
* **Web Applications**:
  + **Spring MVC** for building the backend (Controller).
  + Dynamic HTML and CSS for the frontend (View).
  + Database interactions via JPA or Hibernate (Model).

This technology stack equips developers to implement small-scale to enterprise-level applications using the MVC design pattern efficiently.

4o

Code example:

public class Main {

public static void main(String[] args) {

Database db = new Database();

db.save("message", "Welcome to MVC with Java Technology Stack!");

Model model = new Model(db);

View view = new View();

Controller controller = new Controller(model, view);

controller.updateData("Updated Message: MVC Technology Stack in Action!");

controller.displayData();

}

}

class Database {

private java.util.HashMap<String, String> dataStore = new java.util.HashMap<>();

public void save(String key, String value) {

dataStore.put(key, value);

}

public String retrieve(String key) {

return dataStore.getOrDefault(key, "No data found for the given key.");}

}

class Model {

private Database database;

private String currentKey = "message"; // Fixed key for simplicity

public Model(Database database) {

this.database = database;

}

public void saveData(String data) {

database.save(currentKey, data);

}

public String getData() {

return database.retrieve(currentKey);

}

}

class View {

public void displayMessage(String message) {

System.out.println("Displaying Data: " + message);

}

}

class Controller {

private Model model;

private View view;

public Controller(Model model, View view) {

this.model = model;

this.view = view;

}

public void updateData(String data) {

model.saveData(data);

}

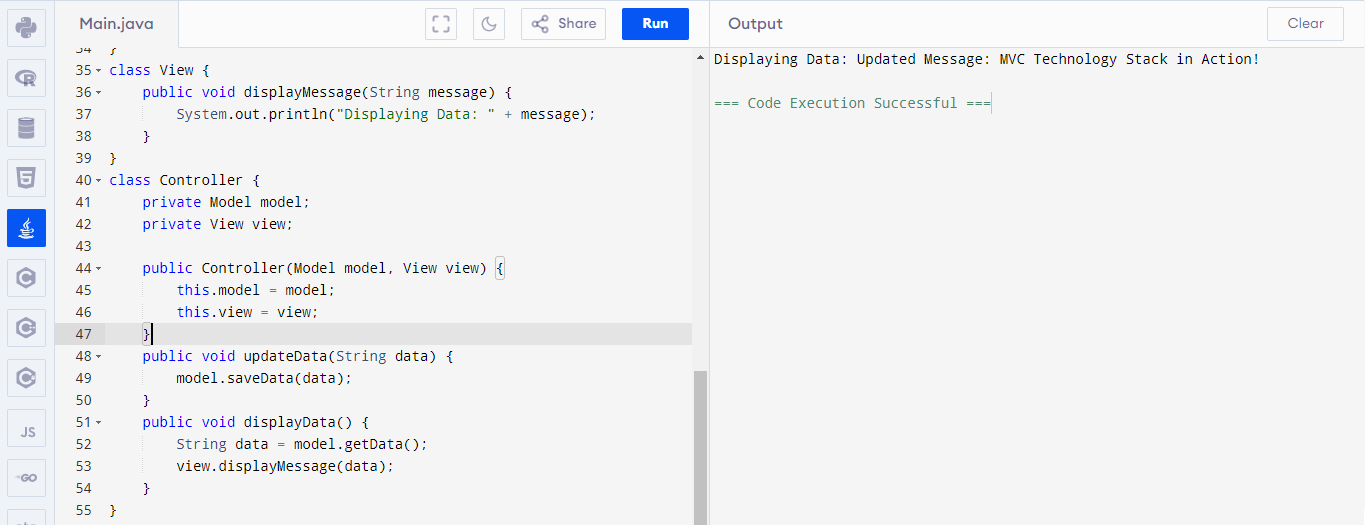
public void displayData() {

String data = model.getData();

view.displayMessage(data);

}

}



**MODIFIED MVC EXAMPLE**

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

Database db = new Database();

Model model = new Model(db);

View view = new View();

Controller controller = new Controller(model, view);

controller.saveData("message", "Welcome to the Enhanced MVC Example!");

controller.saveData("info", "This example supports multiple keys.");

controller.displayData("message");

controller.displayData("info");

controller.updateData("message", "Updated Message: MVC with Flexibility!");

controller.displayData("message");

}

}

class Database {

private HashMap<String, String> dataStore = new HashMap<>();

public void save(String key, String value) {

dataStore.put(key, value);

}

public String retrieve(String key) {

return dataStore.getOrDefault(key, "No data found for the given key.");

}

}

class Model {

private Database database;

public Model(Database database) {

this.database = database;

}

public void saveData(String key, String data) {

database.save(key, data);

}

public String getData(String key) {

return database.retrieve(key);

}

}

class View {

public void displayMessage(String message) {

System.out.println("Displaying Data: " + message);

}

}

class Controller {

private Model model;

private View view;

public Controller(Model model, View view) {

this.model = model;

this.view = view;

}

public void saveData(String key, String data) {

model.saveData(key, data);

}

public void updateData(String key, String data) {

model.saveData(key, data);

}

public void displayData(String key) {

String data = model.getData(key);

view.displayMessage(data);

}

}

