**MVC Design Pattern**

**Assignment - 1**

Name - Heet Dobariya Roll No. - 22BCP177 Group - G5

* **MVC Design Pattern :**

The MVC (Model-View-Controller) design pattern is a software architectural pattern commonly used in developing user interfaces. It divides an application into three interconnected components to separate the internal representations of information from the ways that information is presented and accepted by the user.

Here's a brief overview of each component in the MVC pattern:

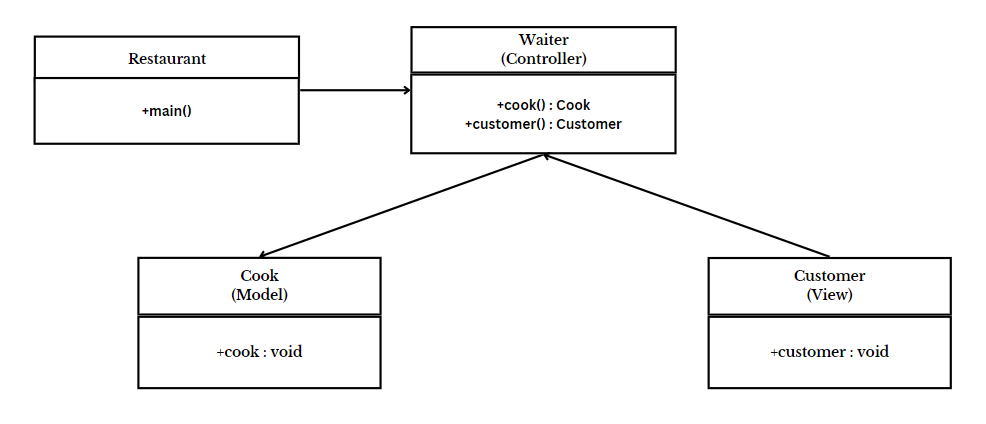
1. Model: The Model represents the application's data and core business logic. It is responsible for managing the data, processing it, and responding to requests for information about its state. The Model component does not depend on the user interface or how the data is presented. Instead, it focuses on the application's behavior and state.

2. View: The View is responsible for the presentation of the data to the user. It displays the data from the Model in a format suitable for interaction, such as a graphical interface or a web page. The View component is designed to be as lightweight as possible, often retrieving data from the Model and rendering it to the user.

1. Controller:
2. The Controller acts as an intermediary between the Model and the View. It receives user input from the View, processes that input (such as handling button clicks or form submissions), and updates the Model accordingly. The Controller is responsible for updating the View to reflect changes in the Model's state. It essentially controls the flow of the application, deciding which actions to take in response to user input.

The MVC pattern promotes a clean separation of concerns, making applications easier to develop, test, and maintain. It's widely used in various software development frameworks and environments, including web development, desktop applications, and mobile apps.

* **Program :** Implement MVC design pattern for restaurant example.
* **UML Diagram :**



* **Code :**

import java.util.\*;

class Cook

{

public void prepareFood(String dish)

{

System.out.println("Preparing " + dish + "...");

System.out.println(dish + " is ready!");

}

}

class Customer

{

public void placeOrder(String dish)

{

System.out.println("Placing order for " + dish);

}

}

class Waiter

{

private Cook cook;

private Customer customer;

public Waiter()

{

this.cook = new Cook();

this.customer = new Customer();

}

public void takeOrder(String dish)

{

customer.placeOrder(dish);

cook.prepareFood(dish);

System.out.println(dish + " is served!");

}

}

class RestaurantExample

{

public static void main(String[] args)

{

String dish;

Scanner sc = new Scanner(System.in);

System.out.println("Enter the dish you want to order: ");

dish = sc.nextLine();

Waiter waiter = new Waiter();

waiter.takeOrder(dish);

}

}

* **Output :**

