**Software Engineering Assignment 2 – Designing a Restaurant Ordering System: A UML Modeling Challenge**

**Description**

You work for a new start-up that wants to develop and sell software that allows restaurants to digitalize their menus and create an order process for their customers. The goal is to streamline the logistics for a food business. We need to track orders going through three stages: customer, kitchen, and waiter/delivery person.

The software should be designed such that it can be used for any type of restaurant, from pizzeria to bistro, from lunch café to snack bar. Both for in-restaurant dining as well as take-away and delivery. Your goal at the start of the development is to create a comprehensive design of all the classes involved.

**Users and Stakeholders:**

The users of the software will be restaurant staff and customers, while the stakeholders will be the restaurant owners and the software development team.

**Features and Requirements:**

The software should have the following features and requirements:

• A restaurant menu presenting all dishes with their ingredients, drinks, and desserts.

• An order-entry module where customers, servers, or other systems can enter orders:

• Through a web-interface for home delivery.

• Through a web-interface for in-restaurant ordering by scanning a QR code, browsing through the menu and ordering.

• By a server/worker in the restaurant who takes and enters the customer’s order.

• By offering their menu through an API to Uber-eats / Thuisbezorgd / Takeaway / Deliveroo etc.

• A delivery module that manages deliveries to customers.

• The ability to calculate prices using custom-made classes that take into account different pricing models, such as discounts and happy hours.

• Compatibility with different types of restaurants and the ability to integrate with third-party delivery services.

**Pricing:**

Different restaurants use different ways to calculate the prices of their dishes. Therefore, you’ll design the software such that it is possible to use on or more custom-made classes that calculate prices. For instance:

• Restaurant A wants their dishes to be 10% more expensive if eaten in.

• Restaurant B offers a discount of 5% for lunch orders placed between 10.00 and 11.00 a.m.

• Restaurant C has a happy hour between 16:00 and 18:00, two cocktails for the price of 1!

• Restaurant C also wants their dishes to be 5% more expensive if eaten in.

Our company will offer this customization as a bespoke service when the restaurant purchases the software. This means that, our company will implement the pricing strategy in Java for the customer. However, the design must allow for uniform implementation, addition, and modification of the pricing strategy. Ensuring that restaurants can easily change their pricing strategy. Therefore, in the design an interface should be included that allows the implementation of arbitrary pricing strategies.

**To achieve this, the system will consist of several modules:**

1. The menu module which presents all dishes with their ingredients, drinks, and desserts. Each restaurant has a menu with different menu items. These items can consist of drinks, main dishes, side dishes, desserts, etc. In some cases, such as main dishes, the price of the dish will depend on its ingredients.
2. The order-entry module allows customers, servers, or other systems to enter orders. Customers can place orders through a web interface for home delivery, through a web interface for in restaurant ordering by scanning a QR code and browsing through the menu, or by a server/worker in the restaurant who takes and enters the customer’s order. The restaurant can also offer their menu through an API to third-party delivery services such as Uber Eats, Takeaway, and Deliveroo.
3. The pricing module allows for custom pricing for each restaurant. Different restaurants use different ways to calculate the prices of their dishes. Therefore, the software should be designed such that it is possible to use one or more custom-made classes that calculate prices.
4. The delivery module manages deliveries to customers. If customers order through the web interface, a delivery must be made. The restaurant has several deliverers who deliver orders. When an order is placed, it takes some time to prepare it in the kitchen. After this, the order should be delivered as soon as possible by an available deliverer.

The goal of this project is to streamline the logistics of a food business by tracking orders going through three stages: customer, kitchen, and waiter/delivery person. To achieve this, we need a comprehensive design of all the classes involved in the software.

**Summary of requested deliverables:**

• **UML class diagram**: This should include important constructors, public methods, and properties of the classes that will implement the software's features. Methods and properties that are not important to understanding how the software functions may be omitted. Ensure that you include them where they matter.

• **Three Sequence diagrams**: Draw separate sequence diagrams for all three ordering-entry methods described in the assignment.

• **All class and interface definitions**: Write the classes, interfaces, fields, and important methods that define your class model. Implementation of methods is not necessary at this stage.

• **Package structure**: Place your classes and interfaces in appropriate packages (for Java, this means using an appropriate directory structure). Ensure that classes that belong together are in the same package (i.e. in the same directory), and create loosely coupled dependencies between packages to avoid circular dependencies and too many dependencies between classes that should not have dependencies. Ensure that your class diagram matches your package structure.

• **Sample instances**: Create a few sample instances of whatever is aggregated by the relations in your class diagram (e.g. starters, main dishes, drinks, desserts for the menu). Create a few orders and deliveries as well. Every relation between two classes in your class diagram must be shown using examples in your code.

**Some remarks and tips:**

• In terms of class design. You do not have to model the graphical user interface (GUI), only the back-end of the software. Meaning all classes that form the structure (the models, controllers and interfaces) of the software and how they relate.

• Make sure to include the user and GUI as actor and lifeline in the sequence diagram.

• You should however, include a class that acts as the API to the external ordering websites (Uber eats etc.). Think about how you could generalize this, internally you have one menu, but you must “present” this menu to different external services. Similarly, external services must be able to connect to your software to place orders. If you have no knowledge of how API’s work, you can either base the class on your own assumptions or first do some research online (search “REST API”) for examples and approaches.

• A good starting point may be to first define the classes needed to model the software. Next, see which classes aggregate others. Can you make the design as compact as possible?

• Once you think you have a correct diagram, write out the classes, make “sample” objects of your classes and see if your model still makes sense. Adjust if needed.

• Consider that the class diagram and your “empty” classes should be sufficient to start implementing the project without having to change the structure later on. Multiple teams will start working on implementing the software and they must all be able to trust the design.

• Check if your design can be structured into cohesive packages that have clear dependencies on other packages. At most one or two classes in a package should depend on another package.

• The pricing strategy should be able to take into account any type of pricing that the customer wants. The three examples in the assignment are nothing more than examples. It's up to you to design a facility in your class diagram that allows programmers to create pricing strategies as "plug-in" classes. Think about what we talked about in the OOD lecture that would allow for such a mechanism

• In your class diagram you may treat the web interface and API as "black boxes" you do not have to model them in detail. So just include one or two classes as you see fit that "create" orders for your system.

• Consider that, in your class diagram you are modeling everything you need to consider to represent the state space of your program. You are not modelling the flow of your program, but how to represent all the possible states your system can be in.

• Please don't include any extra functionality in your design, only what is described in the assignment.

• Is the kitchen an entity that you need to model in your class design?

• Make sure you identify where instances, abstract classes and inheritance are needed

1st step

**Step 1**

Based on the requirements provided, here is a suggested UML class diagram for the Restaurant Ordering System:

+-------------------+

| <<interface>> |

| PricingStrategy |

+-------------------+

| |

| +calculatePrice()|

+-------------------+

^

|

|

+---------------------|---------------------+

| | |

| | |

| +-----------------+ |

| | MenuItem | |

| +-----------------+ |

| | -name: String | |

| | -price: double | |

| +-----------------+ |

| ^ |

| | |

| | |

| +----------------------+ |

| | MainCourse | |

| +----------------------| |

| | -ingredients: String[]| |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Drink | |

| +----------------------| |

| | | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Dessert | |

| +----------------------| |

| | | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Menu | |

| +----------------------| |

| | -items: MenuItem[] | |

| +----------------------+ |

| | -addMenuItem() | |

| | -removeMenuItem() | |

| | -getMenuItem() | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Order | |

| +----------------------| |

| | -items: MenuItem[] | |

| | -status: OrderStatus | |

| +----------------------+ |

| | -addItem() | |

| | -removeItem() | |

| | -calculateTotalPrice()| |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Delivery | |

| +----------------------| |

| | -order: Order | |

| | -status: DeliveryStatus| |

| | -deliverer: Deliverer| |

| +----------------------+ |

| | -assignDeliverer() | |

| | -updateStatus() | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Restaurant | |

| +----------------------| |

| | -name: String | |

| | -menu: Menu | |

| | -deliveryService: DeliveryService| |

| +----------------------+ |

| | -receiveOrder() | |

| | -updateOrderStatus() | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | DeliveryService | |

| +----------------------| |

| | | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Deliverer | |

| +----------------------| |

|

| +----------------------| |

| | -name: String | |

| | -available: boolean | |

| +----------------------+ |

| | -markAvailable() | |

| | -markUnavailable() | |

+----------------------| |

| |

| |

+----------------------|---------------------+

| | |

| | |

| +---------------------------------+

| | <<interface>> OrderEntryMethod |

| +---------------------------------+

| | |

| | +enterOrder() |

| +---------------------------------+

| ^

| |

| |

| +-----------------|-----------------+

| | WebInterface | |

| +-----------------|-----------------+

| | -restaurant: Restaurant |

| +-----------------|-----------------+

| |

| |

| +-----------------|-----------------+

| | Server | |

| +-----------------|-----------------+

| | -restaurant: Restaurant |

| +-----------------|-----------------+

| |

| |

| +-----------------|-----------------+

| | ThirdPartyAPI | |

| +-----------------|-----------------+

| | |

| +---------------------------------+

| |

| |

| +-----------------|-----------------+

| | Kitchen | |

| +-----------------|-----------------+

| | -order: Order |

| +-----------------|-----------------+

| |

| |

| +-----------------|-----------------+

| | Customer | |

| +-----------------|-----------------+

| | -name: String |

| +-----------------|-----------------+

|  |  |
| --- | --- |
|  |  |

Note: The above class diagram provides an overview of the class structure and their relationships based on the given requirements. It is not a complete and final design, but rather a starting point for further refinement and implementation.

I will now provide sequence diagrams for each of the three order-entry methods described in the assignment.

**Explanation:**

PricingStrategy (Interface):

This interface represents the pricing strategy that can be implemented by custom-made classes for calculating prices.

It includes a method called **calculatePrice()** that will be implemented by the specific pricing strategy classes.

MenuItem:

Represents a generic menu item that can include drinks, main courses, side dishes, desserts, etc.

It has properties such as **name** and **price**.

MainCourse, Drink, Dessert:

These are specific types of menu items that inherit from the MenuItem class.

Each type of menu item can have its own additional properties or behaviors specific to its category.

Menu:

Represents the restaurant's menu, which is a collection of menu items.

It has an array or list of MenuItem objects.

It provides methods to add, remove, and retrieve menu items.

Order:

Represents an order placed by a customer.

It contains an array or list of MenuItem objects.

The order also has a status (OrderStatus) indicating its progress.

Delivery:

Represents a delivery made to a customer.

It is associated with an Order and a Deliverer.

The delivery has a status (DeliveryStatus) indicating its progress.

Restaurant:

Represents a restaurant.

It has properties like **name**, **menu** (an instance of Menu), and **deliveryService** (an instance of DeliveryService).

The restaurant can receive orders and update their status.

DeliveryService:

Represents a delivery service that handles the delivery process.

It acts as an external entity interacting with the system.

The specific implementation details of the delivery service can be encapsulated within this class.

Deliverer:

Represents a person responsible for delivering orders.

It has properties like **name** and **available**.

The deliverer can be marked as available or unavailable for deliveries.

WebInterface, Server, ThirdPartyAPI:

These classes represent different order entry methods.

WebInterface allows customers to place orders through a web interface.

Server represents the staff in the restaurant who takes and enters customer orders.

ThirdPartyAPI represents an external API through which orders can be placed.

Kitchen:

Represents the kitchen where orders are prepared.

It is associated with an Order and manages the order preparation process.

Customer:

Represents a customer who places an order.

It has a property **name** to identify the customer.

The class diagram shows the relationships between the classes and their properties and methods. It provides an overview of the system's structure and entities involved in the restaurant ordering process.

Next, let's move to the sequence diagrams for the three order-entry methods.

complete UML class diagram for the Restaurant Ordering System:

+-------------------+

| PricingStrategy |

+-------------------+

| |

| +calculatePrice() |

+-------------------+

^

|

|

+---------------------|---------------------+

| | |

| | |

| +-----------------+ |

| | MenuItem | |

| +-----------------+ |

| | -name: String | |

| | -price: double | |

| +-----------------+ |

| ^ |

| | |

| | |

| +----------------------+ |

| | MainCourse | |

| +----------------------| |

| | -ingredients: String[]| |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Drink | |

| +----------------------| |

| | | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Dessert | |

| +----------------------| |

| | | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Menu | |

| +----------------------| |

| | -items: MenuItem[] | |

| +----------------------+ |

| | -addMenuItem() | |

| | -removeMenuItem() | |

| | -getMenuItem() | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Order | |

| +----------------------| |

| | -items: MenuItem[] | |

| | -status: OrderStatus | |

| +----------------------+ |

| | -addItem() | |

| | -removeItem() | |

| | -calculateTotalPrice()| |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Delivery | |

| +----------------------| |

| | -order: Order | |

| | -status: DeliveryStatus| |

| | -deliverer: Deliverer| |

| +----------------------+ |

| | -assignDeliverer() | |

| | -updateStatus() | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Restaurant | |

| +----------------------| |

| | -name: String | |

| | -menu: Menu | |

| | -deliveryService: DeliveryService| |

| +----------------------+ |

| | -receiveOrder() | |

| | -updateOrderStatus() | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | DeliveryService | |

| +----------------------| |

| | | |

| +----------------------+ |

| |

| |

| +----------------------| |

| | Deliverer | |

| +----------------------| |

| | | |

+----------------------|-----------------+

|

| +----------------------| |

| | -name: String | |

| | -available: boolean | |

| +----------------------+ |

| | -markAvailable() | |

| | -markUnavailable() | |

+----------------------|-----------------+

|

|

+----------------------|---------------------+

| | |

| | |

| +---------------------------------+

| | <<interface>> OrderEntryMethod |

| +---------------------------------+

| | |

| | +enterOrder() |

| +---------------------------------+

|

|

+-----------------|-----------------+

| WebInterface | |

+-----------------|-----------------+

| -restaurant: Restaurant |

+-----------------|-----------------+

|

|

+-----------------|-----------------+

| Server | |

+-----------------|-----------------+

| -restaurant: Restaurant |

+-----------------|-----------------+

|

|

+-----------------|-----------------+

| ThirdPartyAPI | |

+-----------------|-----------------+

| |

+---------------------------------+

|

|

+-----------------|-----------------+

| Kitchen | |

+-----------------|-----------------+

| -order: Order |

+-----------------|-----------------+

|

|

+-----------------|-----------------+

| Customer | |

+-----------------|-----------------+

| -name: String |

+-----------------|-----------------+

Please note that the class diagram provided represents the structure and relationships of the classes involved in the Restaurant Ordering System based on the given requirements. It serves as a foundation for further implementation and refinement of the system.

**Software Engineering Assignment 2 Report: Designing a Restaurant Ordering System**

Introduction:

The aim of this report is to present the design of a Restaurant Ordering System, which is intended to streamline the logistics for food businesses by digitalizing menus and creating an efficient order process for customers. This report will include a UML class diagram, sequence diagrams for different order entry methods, class and interface definitions, package structure, and sample instances.

UML Class Diagram:

The provided UML class diagram outlines the structure of the Restaurant Ordering System. It includes classes such as MenuItem, Menu, Order, Delivery, Restaurant, DeliveryService, Deliverer, PricingStrategy, and various interfaces like OrderEntryMethod. The relationships between these classes depict how they interact within the system to facilitate the ordering process efficiently.

Sequence Diagrams:

Three separate sequence diagrams illustrate the order-entry process through different methods: WebInterface, Server, and ThirdPartyAPI. These diagrams depict the interactions between actors and the system components, showcasing the flow of events during order placement.

Class and Interface Definitions:

Each class and interface in the system is defined with its properties, methods, and relationships. For instance, the MenuItem class represents generic menu items, while subclasses like MainCourse, Drink, and Dessert inherit from it. Similarly, the Order class encapsulates order details, including menu items and status.

Package Structure:

The system's classes and interfaces are organized into packages to manage dependencies and maintain a modular structure. The package structure ensures loose coupling between components and facilitates scalability and maintenance.

Sample Instances:

Sample instances of classes like Menu, Order, Delivery, Restaurant, and Deliverer are provided to demonstrate how the system entities interact and relate to each other. These instances help visualize the practical implementation of the class diagram.

Conclusion:

The design presented in this report lays the foundation for the development of a comprehensive Restaurant Ordering System. It encompasses essential features like menu management, order processing, delivery management, and pricing customization. Further refinement and implementation based on this design will lead to the creation of a robust and user-friendly software solution for restaurants to streamline their operations.

Overall, the design adheres to the specified requirements and provides a clear understanding of the system's architecture and functionality.