1. Name : Jayadharshini Jaiganesh

Title: Pizza Point of Sale

2. Project description: This project models a pizzeria, with admin and customers as actors. The objective is to get customer preference of toppings and all other raw materials, place the order, validate his payment information and notify the customer when the order gets ready. The admin can view the overall orders placed and can also add new items to the menu.

3. Implemented features:

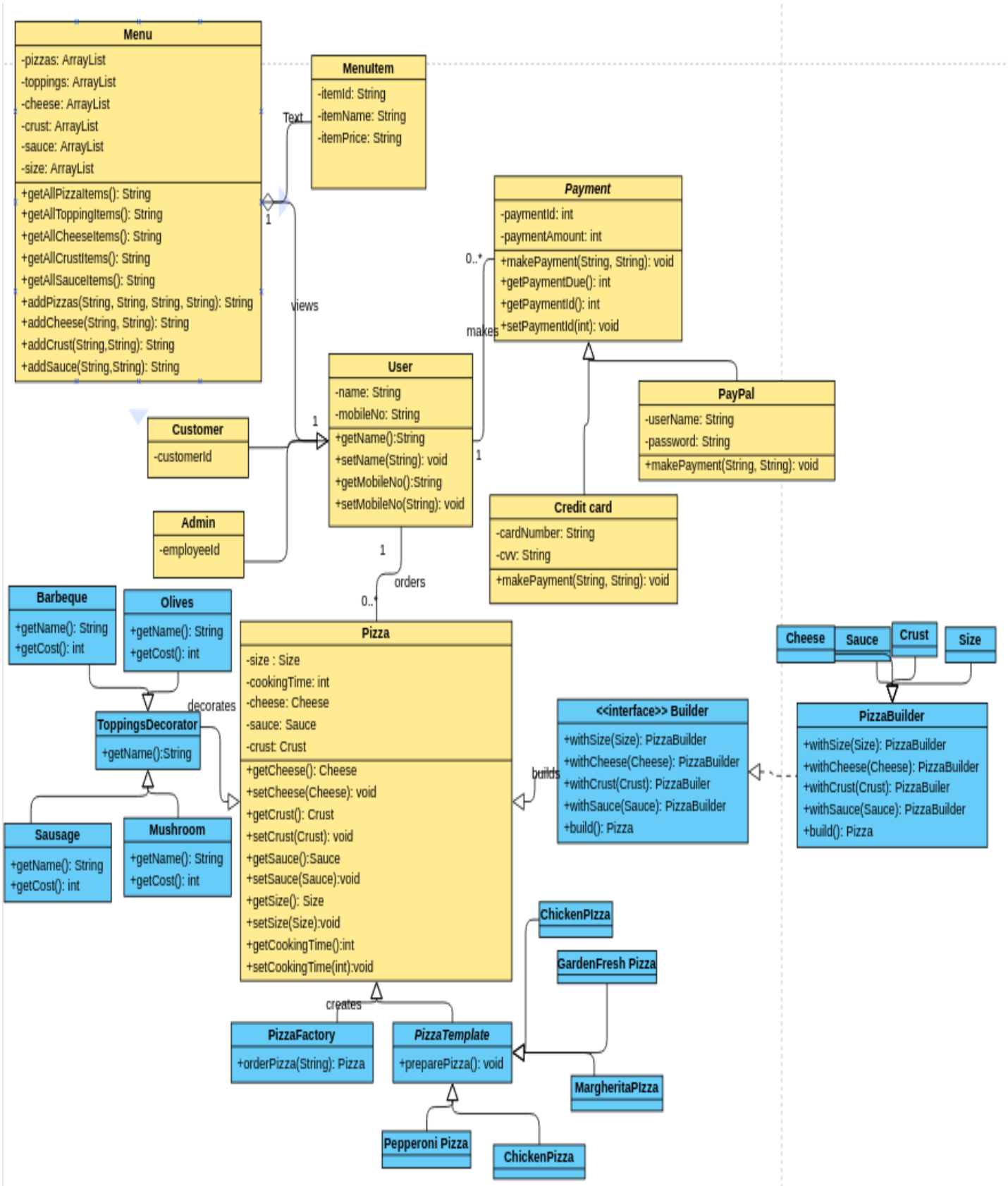
Id	Title
F-01	User can view the menu and customize the order.
F-02	User must be able to view the total price of the order and can edit/delete the order
F-04	Admin must be able to change daily menu
F-05	Admin can view daily orders
F-07	Payment must be done through either credit card or Paypal
F-08	User and Admin can login

4. Features that were not implemented:

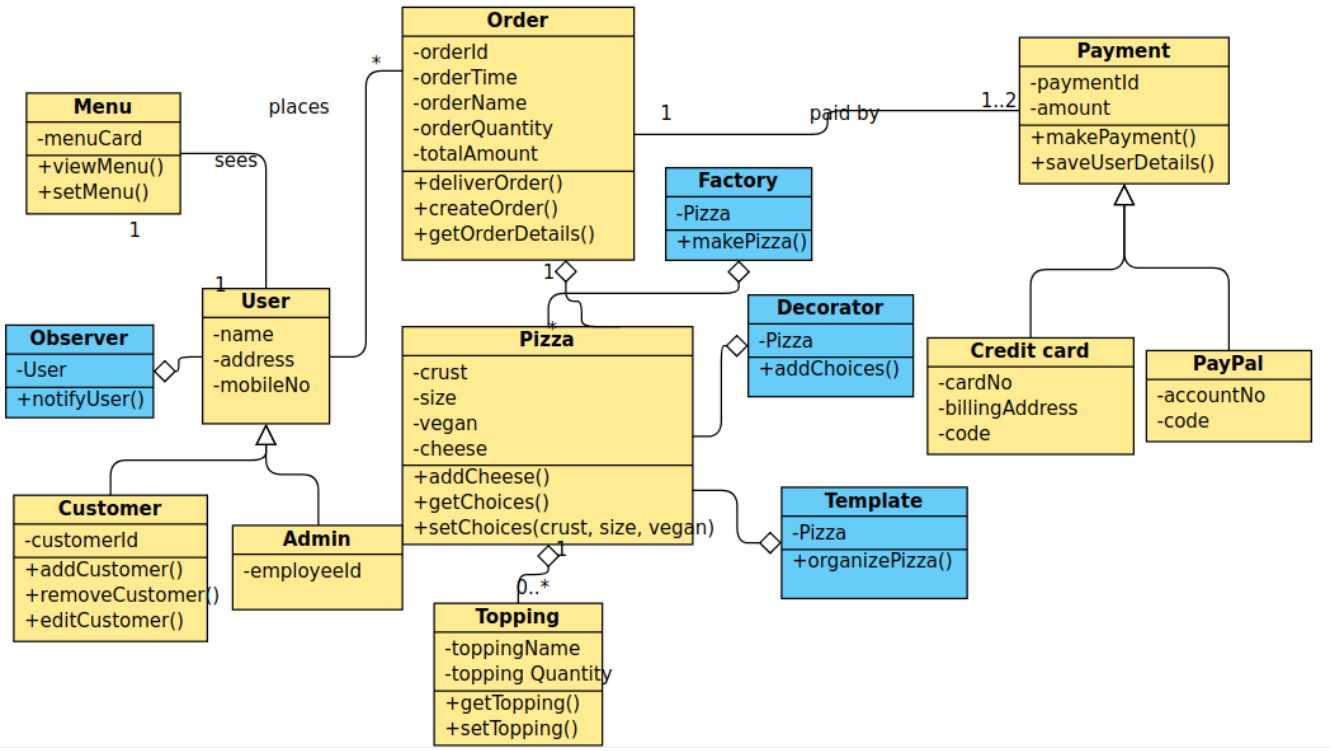
Id	Title
F-04	Admin must be able to stock groceries
F-03, F-06	Home delivery and Admin can track pizza delivery (Stretch features)

5. Changes in class diagram (between old and new):

- Used Buider design pattern for adding extras like cheese, crust.. in the Pizza instead of just specifying them in the Pizza class, so as to reduce it’s complexity.
- A new MenuItem class is introduced, which has all menu item details, which is then used in Menu class as an ArrayList, instead of just having a single Menu class.
- Order class is removed, as all the details involving the Payment is handled in the Payment class.
- Observer design pattern is not implemented, as I found it not as suitable to the project as Builder.



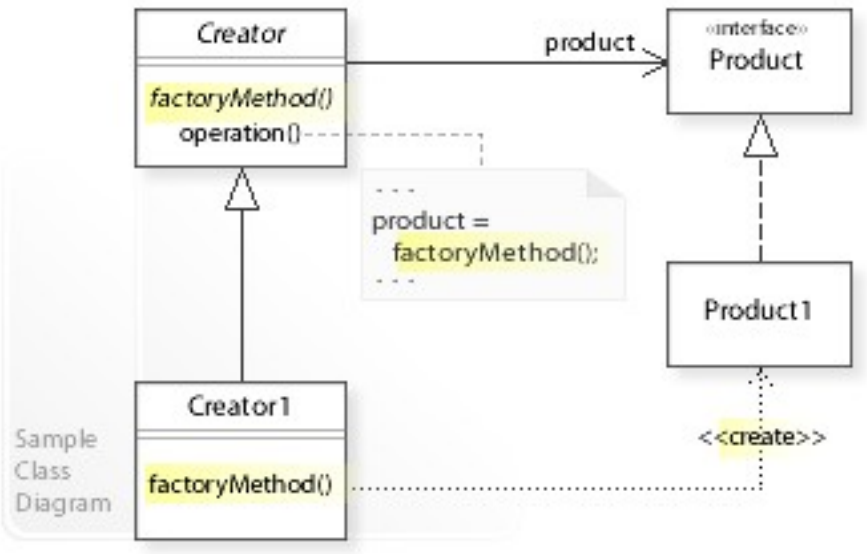
Pizza Point of Sale – New Class Diagram



Pizza Point of Sale – Old Class Diagram

6. Design Patterns implemented:

1. Factory – class diagram:

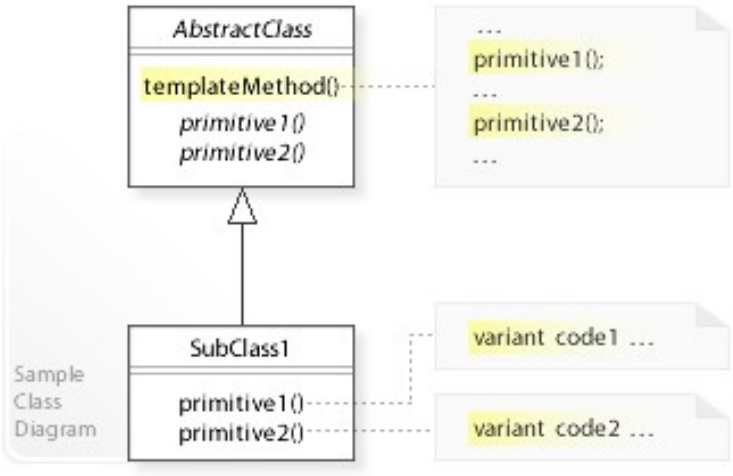


In this application *PizzaFactory* is used by the Pizza class for creating pizza objects based on the type of pizza specified by the customer. Here,

Creator - PizzaFactory class which creates pizza objects.
factoryMethod() – orderPizza(String) is the method used by PizzaFactory to create the pizza objects

Reason: Since Factory pattern enables object without exposing the creation logic to the client, it is used to create different type of pizzas here.

2. Template – class diagram:

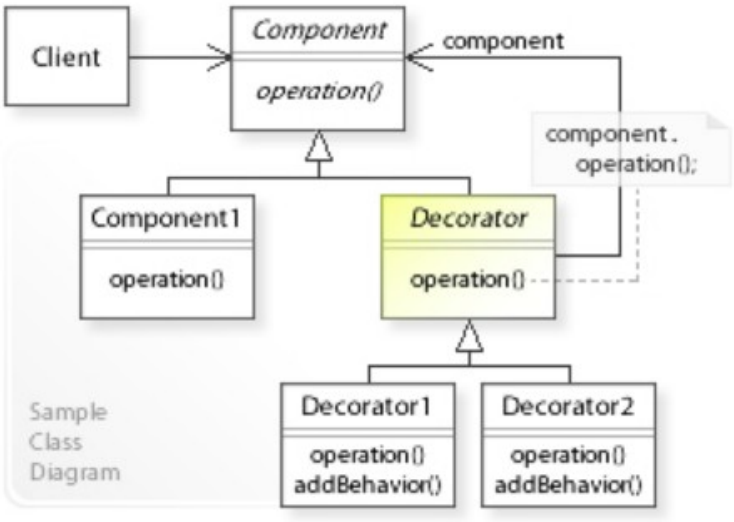


In this application, *PizzaTemplate* is used by the Pizza class for defining the steps involved in creating Pizza. Here,

AbstractClass - PizzaTemplate class which defines steps involved in creating pizza objects.
templateMethod() – preparePizza() method that defines the skeleton for Pizza creation by including methods like prepare(), bake(), cut(), box().
SubClass1..n – CheesePizza, PepperoniPizza, Margherita Pizza, GardenFresh Pizza and ChickenPizza classes which extends the PizzaTemplate class.

Reason: As Template pattern uses an abstract class which defines ways to execute its methods, it is used to depict the steps involved in creating a Pizza.

3. Decorator – Class Diagram:

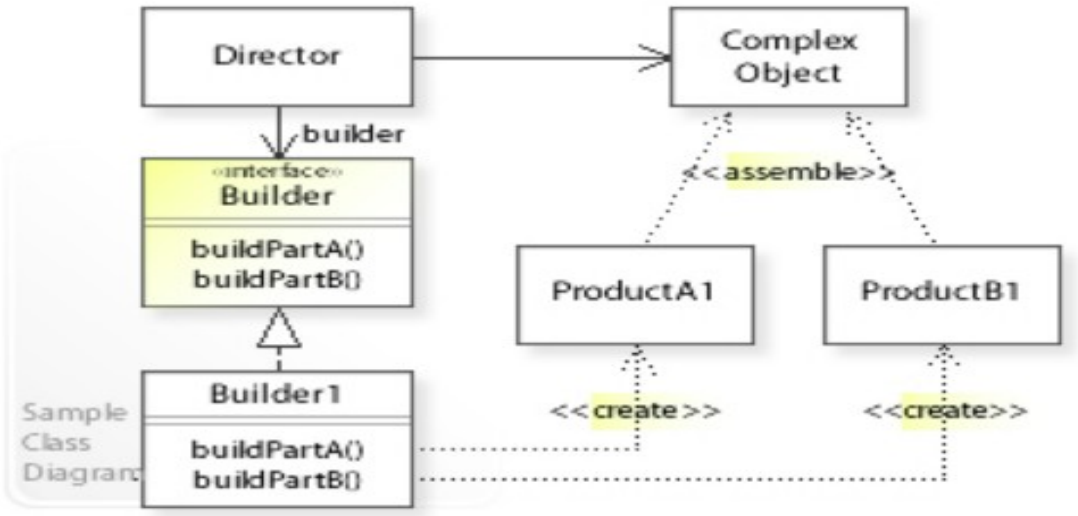


In this application, *ToppingsDecorator* allows adding toppings to an existing pizza object without altering its structure. Here,

Decorator – Toppings Decorator which adds toppings.
Operation() - getName(), which adds the topping to an existing pizza.
Component – Pizza class
Decorator 1..n – Barbeque, Sausage, Mushroom and Olives classes that extend ToppingsDecorator.

Reason: As Decorator pattern acts as a wrapper to existing class, it is used to add toppings to existing Pizza class.

4. Builder – class diagram:



Here, *PizzaBuilder* is used by the Pizza class for building complex Pizza objects i.e, when crust, cheese, sauce and crust of the Pizza need to be specified. Here,

Director - Pizza class which uses PizzaBuilder for creating complex pizza objects.
Builder – An interface implemented by Pizza Builder that has the abstract methods to be implemented.
Builder 1,...n- Cheese, Crust, Size and Sauce classes respectively.

Reason: As Builder pattern builds a complex object using simple objects step by step, all extras in a Pizza like Cheese, Crust etc.. can be added through it.

7. Reflection:

By doing this project, I have learnt the following,

- Constructing a clear class diagram with data types and relationship between class specified will save a lot of time and effort in later stages.
- A design pattern must be implemented only when it suits the goal of the project.
- Integrating the project finally in the end consumes a lot of time. Integrating it as and when new classes are added would be advisable.