



UML project

APU Campus Wide Drinks and Snack Ordering Kiosk System

Mohamed zammit chatti – Youssef mazouz – rayen haddad

Object Oriented Methods with UML

Lecturer: DR. IMRAN MEDI

Table of Contents

**Introduction2**

**Use Case Diagram3**

**Class Diagram4**

**Sequence Diagrams5**

**About the User Interface6**

**Design Patterns7**

**Class Diagram with Design Patterns8**

**Conclusion9**

# Introduction

We are three students and our mission is to design and develop a Kiosk System for Ordering Drinks and Snacks on the APU Campus. This project will allow us to practice our knowledge of UML by making a use case, a class diagram and a sequence diagram of an imposed scenario. We will then have to implement the design pattern in Java and then refine the class diagram thanks to the result of the implementation of the design pattern. The scenario that has been imposed on us consists in setting up a system of five kiosks allowing students or teachers/staff to order drinks or snacks. Users will be able to customize their drink, they will be able to choose the size or the quantity of sugar, milk... Users will also be able to order snacks and a selection of sandwiches and customize them. As for the payment, it can be done only with the APU card available for students and teachers. When paying, the user will have the cost of their order deducted directly from the AP card. They will have the possibility to recharge their AP card.

# Use Case Diagram

Diagram

Description automatically generated

To start this Use Case Diagram, we put a "*student*" as an actor. We made an inheritance from "*teacher/staff*" to the student since they can also order from the kiosks. We put an association between "*student*" and "*browse menu*", then we made an "*order*" extension of "*browse menu*". When the student orders, he will be able to choose a snack and a drink, for that we made a generalization of "*Snack*" and "*Drink*". We made an inclusion "*Size*" to "*Drink*" so that the customer can choose the size of his drink. When the customer places his order he will have the possibility to customize it, so we made an extension "*customize*" to "*order*". We then made three generalizations of "*Customize*" so that the customer can add toppings, customize the ingredients as well as the quantity. For the customer to cancel his order we put a "*cancel order*" extension. Once the customer has chosen his order he will have to proceed to the payment. The customer will have to make the payment of the order with his AP Card, we have made an inclusion of "m*ake payment*" with an inclusion "*Check Balance*" which a dependency of "*payment*". The payment can be validated or not, so we made two generalizations "*valid payment*" and "*invalid payment*", these two generalizations go to "*payment*". Once the payment is made, the amount will be updated with an inclusion of "*payment*". "*Update Card Balance*" is associated with the "*AP Card System*" actor.

The customer will also have the possibility to have information about the ingredients and an extension will allow to update this information. This extension is directly associated with the "*Kitchen Area*". The student has the possibility to validate his order and an inclusion "*Update all Kiosks*" to update the kiosks after the validation of the order. Once the payment is validated, the order validated and the kiosks updated, the order will be sent. We have therefore included "*send order*" for these three actions. "*send order*" will be linked by an association to the "*notification system*" actor.

# Class Diagram

Diagram

Description automatically generated

# Sequence Diagrams

Diagram

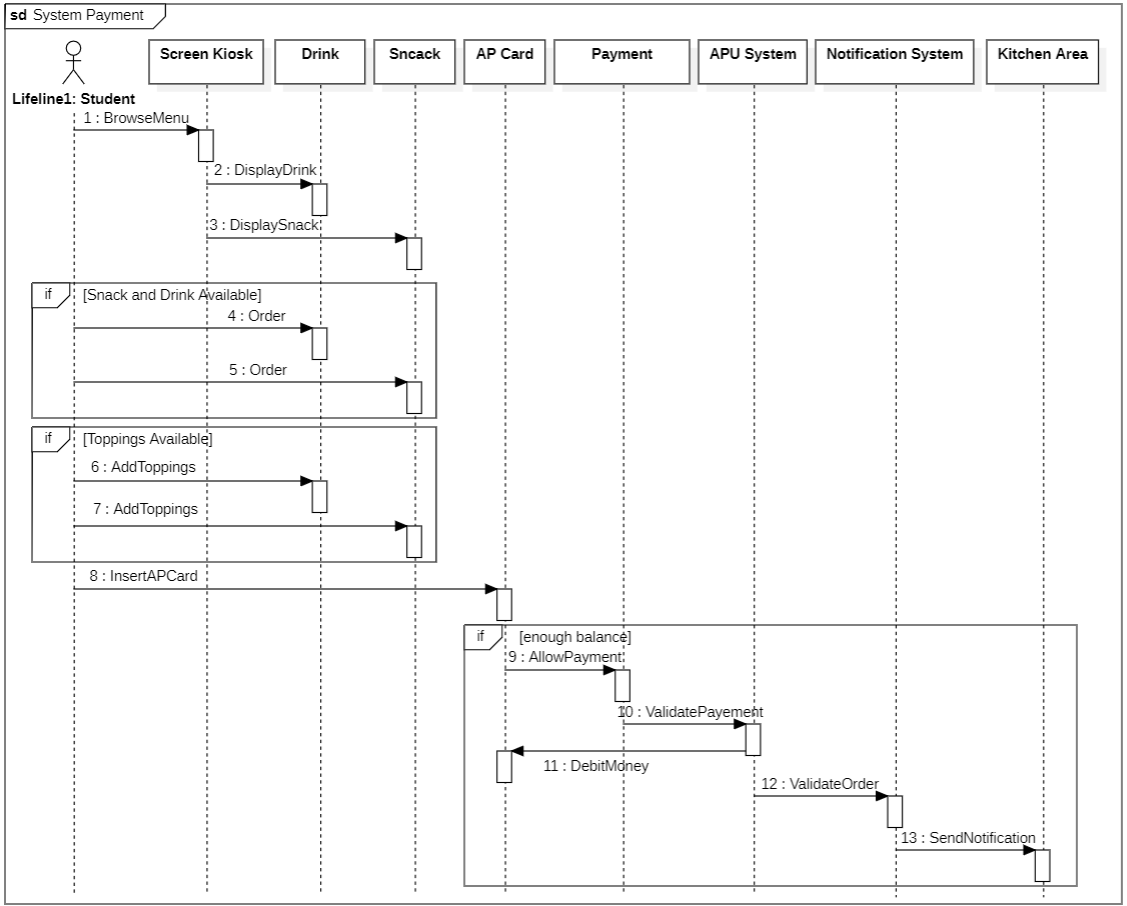
Description automatically generated

In this case we will make a sequence diagram so that customers can choose to add toppings. First the student will browse the menu on the kiosk screen. They will then be able to choose what to eat and drink. Then the toppings will be displayed. We then set a condition, if the student chooses a topping, he can add it to his order. If they don't want it, they can skip this step.

Diagram

Description automatically generated

In this case, we will make a sequence diagram so that customers can order food and drinks from the time they select their order until they pay. First of all the student will have to browse the menu on the kiosk screen again. If there are any snacks or drinks available, the student will be able to choose them. He will be able to see all the products available in the restaurant. The student will then receive a receipt with a summary of the entire order. He will then have to validate his order. Once the order is validated, he will have to proceed to the payment with his AP Card.



In this case, we will make a sequence diagram for customers to pay for their order. First the student will have to browse the menu again on the Kiosk screen. Then the student will have to choose a drink and a snack. If there are drinks and a snack available, the customer will be able to order, if there is the possibility to add a topping, he will be able to add it. Then the customer will insert his AP Card. If the customer has enough balance in his AP Card, the payment will be authorized, then the payment will be validated by the APU system, then the AP Card will be debited. Once the card is debited, the APU system validates the order and sends a notification to the kitchen to start preparing the order.

# About the User Interface

We have decided to exporte our project to java language because, it’s a language we know since the last year.

If we want to talk about the technical aspect of the code (here we talk about class diagram without the design pattern), there are 13 classes with 2 inhertances. The first inheritance involves the class user, there are two kinds of user: staff (lecturer, principal...) and the students. The seconde regarding the class food because te user can order snack or/and drink. All this details are mentionned in the subject.

The normal path is as follows: the user can browses menu if he don not know make him want to eat (with the method browseMenu) and finally order thanks to the method (ValidateOrder). At this time, a new object of type Order is created. A notification sent by the notication system to the preparation area to prepare the order. To prepare order object belonging to the class food is created, here we wanted to represent the creation process.

# Design Patterns

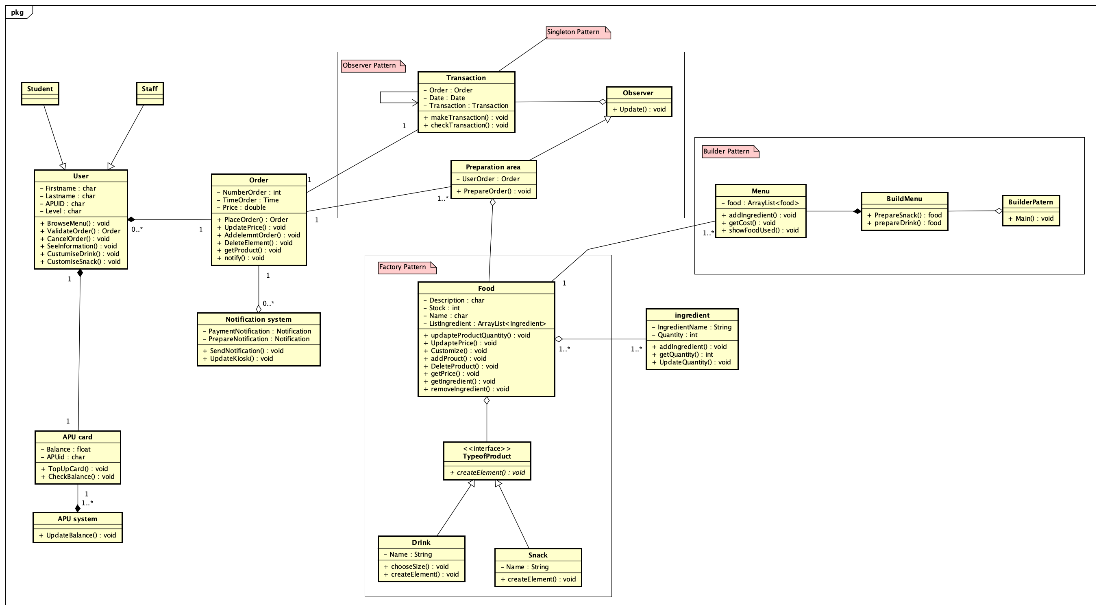
In our project we use four types that we will present you. First of all, the singleton pattern, we apply it to the class transaction. We thought that we need only one instance of transaction is needed to reach the end of the process (one transaction by user). In java we declared all constructors of the class to be private, which prevents it from being instantiated by other objects (represent by reflective relation) and providing a static method that returns a reference to the instance.

The second pattern uses is the factory pattern, the factory design pattern says that define an interface ( here is Typeproduct) for creating object and let the subclasses decide which class to instantiate. If we want snack, drink or both.

The third pattern is the builder pattern, it defines all of the steps that must be taken in order to correctly create a product. This pattern allows us to vary a product's internal representation, that is very interessant if we want to represent menu because the internal representation of a menu often change and it’s very various.

The last pattern is the observer pattern we used here when we want to update the status of a transaction or an order, more precisely of a transaction. Indeed, if something happened to the transaction, the kitchen area will be warning, and the change will be considering.

# Class Diagram with Design Patterns



# Conclusion

This project was very interessant and was a direct application of what we did during class session, very close of the assessments. We developed the understanding of design pattern. The main difficulty of this project was the use of the pattern, to know which one we want to use to aim, focus on and solve which problem. We try to keep in mind the fact that the system will change and grow that is why we try to reduce the dependance between object thanks to the pattern like Singleton or Observer.

The perpective of evolution are numerous, for example, we know that if a person forgets to to up his APcard, it’s will not be able to pay. The think about creates a system that store the order and authorize the payment later.