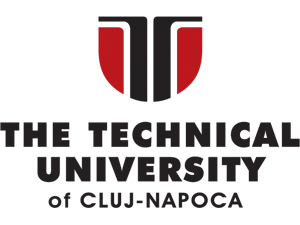
FUNDAMENTAL PROGRAMMING TECHNIQUES

RESTAURANT MANAGEMENT SYSTEM

ASSIGNMENT 4



Teacher: prof. Ioan Salomie

Teacher Assistant: Cristina Pop

Student: Suster Elena Roxana

Group: e\_30421

1. OBJECTIVE

Consider implementing a restaurant management system.

The system should have three types of users: administrator, waiter and chef. The administrator can add, delete and modify existing products from the menu; The waiter can create a new order for a table, add elements from the menu, and compute the bill for an order.

The chef is notified each time it must cook food ordered through a waiter.

Consider the system of classes in the diagram below. To simplify the aplication you may assume that the system is used by only one administrator, one waiter and one chef, and there is no need of a login process.

1. Define the interface Restaurant Processing containing the main operations that can be executed by the waiter or the administrator, as follows:

• Administrator: create new menu item, delete menu item, edit menu item

• Waiter: create new order; compute price for an order; generate bill in .txt format.

1. Define and implement the classes from the class diagram shown above:

• Use the Composite Design Pattern for defining the classes Menu Item, Base Product and Composite Product

• Use the Observer Design Pattern to notify the chef each time a new order containing a composite product is added.

1. Implement the class Restaurant using a predefined JCF collection which uses a hash table data structure. The hash table key will be generated based on the class Order, which can have associated several Menu Items. Use J Table to display Restaurant related information.

• Define a structure of type Map> for storing the order related information in the Restaurant class. The key of the Map will be formed of objects of type Order, for which the hash Code() method will be overwritten to compute the hash value within the Map from the attributes of the Order (Order ID, date, etc.)

• Define a structure of type Collection which will save the menu of the restaurant. Choose the appropriate collection type for your implementation.

• Define a method of type “well formed” for the class Restaurant.

• Implement the class using Design by Contract method (involving pre, post conditions, invariants, and assertions).

1. The menu items for populating the Restaurant object will be loaded/saved from/to a file using Serialization.
2. PROBLEM ANALYSIS, SCENARIOS, USE CASES

General overview

This application should resemble a restaurant with 3 main components, namely:

\* the Administrator, who is able to: create new menu items, to edit the price of the existing ones and to delete menu items

\* the Waiter who is able to create an order taken from the clients at a certain table, he is able to create the order and to deliver it to the chef in order to be cooked or prepared, to compute the total price of the order and to generate a bill in txt format and in the we have the chef whose only job is to get notified by the waiter and to cook the food .

Input and Output

Starting with the input in the application, the user is able to choose to manage the 2 situations: being the waiter or being the administrator. As the administrator he/ she should create menu items, simple ones or composite one.

For the simple ones the user should input the name of the item and the price and for a composite one just the name of the item, because a new frame will be opened and he or she will be able to choose from the already existing items and build up a composite one. He/ she can also delete or edit menu items in the same manner, also displaying the menu, all items with their prices.

When acting as the waiter, the user should introduce the date of the order, the table, also he/ she should display the menu and add items from the menu to the client’s order.

When he/ she finishes to add all the products, the chef is instantly notified and the waiter can display all the orders in a table, this is just to visualize all the existing orders. The waiter is also in charge with the bills and creating them in TXT format.

Use cases

In software and systems engineering, a use case is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal. The actor can be a human, an external system, or time. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. Another way to look at it is a use case describes a way in which a real-world actor interacts with the system. In a system use case you include high-level implementation decisions. System use cases can be written in both an informal manner and a formal manner.

Following example will illustrate on how to plan use cases:

**Use Case:** What is the main objective of this use case. For eg. Adding a software component, adding certain functionality etc.

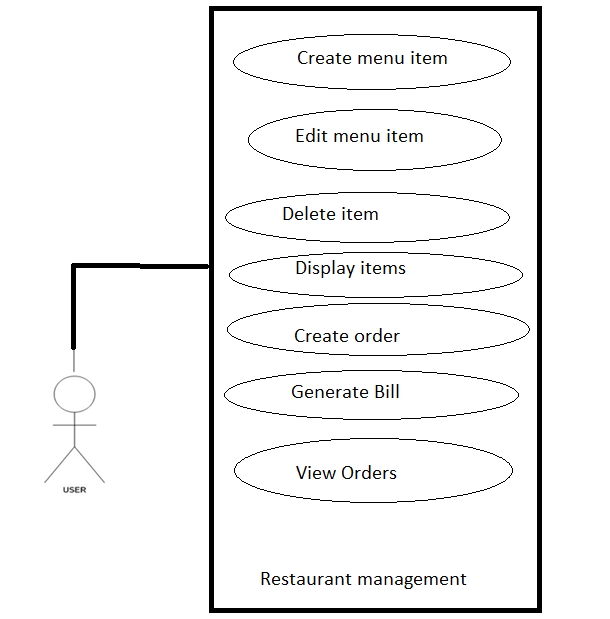
**Primary Actor:** Who will have the access to this use case. In the above examples, administrators will have the access.

**Scope:** Scope of the use case

**Level:** At what level the implementation of the use case be.

**Flow:** What will be the flow of the functionality that needs to be there. More precisely, the work flow of the use case.

Some other things that can be included in the use cases are:preconditions, postconditions, brief course of action and time period.



1. DESIGN

The implementation is done using the following:

\*Data Structures:

ArrayList – I chose to use this structure to keep my queues and to temporary hold my clients, because this kind of lists is useful for storing and accessing data

ArrayList internally uses **dynamic array** to store the elements. Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. This class can **act as a list** only because it implements List only.

\*Packages:

Java packages help in organizing multiple modules and group together related classes and interfaces.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, Smalltalk, C, and C++.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

- *Model*, which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.

- *View*, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)

- *Controller,* which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

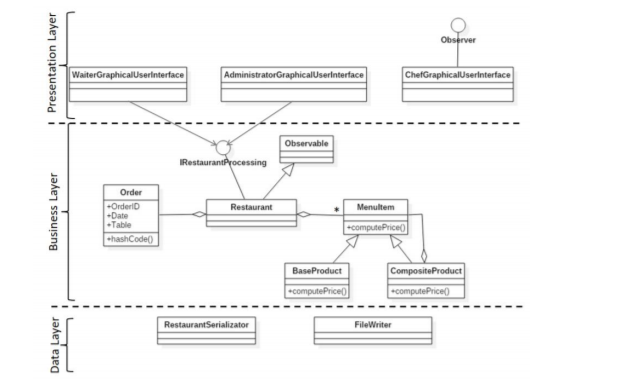
\*Class Design

The whole idea of splitting our program into classes is based on a general rule named divide and conquer. This paradigm can be used almost everywhere: we divide a problem into smaller problems and then we will solve these little, simple and well-known problems .

Dividing our program into classes is one of the types of division which started to become common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects.

I have also used the Observer and the Composite design patterns with respect to the restaurant + waiter + chef and the Menu Item + Base Product + Composite Product respectively.

Now, I will briefly explain my packages and classes that I have used in the presented project in the next subchapter.



1. IMPLEMENTATION
   * app
     + app: – starts the application
   * Controller
     + Main Controller- Initializes the User Interface buttons and sets up the environment of the application.
     + IRestaurantProcessing- Interface that yields the main methods to be implemented in the restaurant class and the Waiter GUI and Administrator GUI classes;
     + Restaurant Serializator- Is responsible with serialization and for the deserialization of the restaurant, namely the Menu, when the application starts.

The whole restaurant is serialized when creating, editing, or deleting a menu item by the administrator also.

* + Model
    - Base Product – Class that models a basic product in the restaurant, for example French fries, cabbage etc.
    - Composite Product- Class that models a more complex product built from other basic or composite products, it basically is a list of other menu items. (it’s price is calculated as being the sum of the smaller components ).
    - Id Generator – Class that uniquely generates the id of the next order. The ids are taken as natural numbers in ascending order starting with 1.
    - Menu Item – Abstract class that resembles a product from the restaurant’s menu. It can be a Base Product or a Composite Product, for those options, the price is computed accordingly.
    - My Date – Class designed to replace the Date class from java for easier work and assignments
    - Order – Class that governs the waiter activities, it is the central part of the waiter’s job. It contains the order id, the date and the table where the products have to be served.
    - Restaurant – The class that is the center of this application, every action must at least pass through the restaurant if not executed by it. It implements the IRestaurantProcessing interface and all the methods present there. It is also built using Design by Contract, testing the preconditions and postconditions with assert instructions in each implemented method.
  + View
    - User Interface – Is the main graphical interface that allows the user to access the other 3 interfaces, namely the Administrator GUI , the Waiter GUI and the Chef GUI. It is initialized in the Main Controller.
    - Administrator GUI – is the place where the administrator does his / her job, adding products in the menu, deleting existing ones and editing prices of the products already present in the menu. The administrator can also view the Menu and decide if he / she should add or not more products.
    - Waiter GUI – is the place where the waiter does his / her job, creating new orders from people on tables, adding products to the orders and notifying the chef to cook those products. The waiter also generates the bill in txt format where the details such as the order id, the date , the table, the ordered products and the final price are written.
    - Chef GUI – the place where a pop-up appears each time an order is created.

1. CONCLUSIONS

This project was a good exercise in remembering the OOP concepts learned in the first semester, but also learning new ones, which I found it very useful and challenging at the beginning. I will present next the things I have learnt during doing this assignment:

First of all , time management is very, crucial, because a good organizational spirit helps you see things gradually and making things from time helps you a lot.

Secondly, modelling the problem in a right way from the beginning helps you to implement it faster.

Thirdly, I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through the mean of research, has the benefit of learning new concepts and a better use of the known ones. In the end, one of the most important things that I have learnt is to make my interfaces from code, because last semester I used Window Builder, and I realised that creating an interface is not so easy as it seems using Window Builder. After this project I can say that my interface building skills are up to date.

By the means of this project I managed to improve my knowledge about the Composite and Observer design patters, as well as the Serialization and Deserialization mechanisms. I must admit that they are pretty useful and show that the person using them really knows what he/ she is doing.

1. FURTHER IMPROVEMENT

Some improvements that I would like to make to this application would be the fact that the chef should be active in the application.

The chef should prepare all the ingredients, cook the food, which should take a while and then the waiter should deliver the food to the clients.

The Chef should signal the administrator when he is nearly out of base products and the administrator should go “shopping” and get all the needed products.

Also the waiter can receive tips and he should have a salary. The tips are given only if the order was delivered in a predefined amount of time.

1. BIBLIOGRAPHY

[www.stackoverflow.com](http://www.stackoverflow.com)

[www.wikipedia.org](http://www.wikipedia.org)

<http://whatis.techtarget.com/definition/model-view-controller-MVC>

Programming Techniques – Lectures of prof. Ioan SALOMIE