# FUNDAMENTAL PROGRAMMING

# TECHNIQUES

Assignment 4

Restaurant Management

System

## Name: Leonties Dan Alexandru

Group: 30423

Contents:

1. Objective
2. Analysis
3. Design
4. Implementation
   1. BusinessLayer
   2. DataLayer
   3. PresentationLayer
   4. Start
5. Results
6. Conclusions
7. Objective

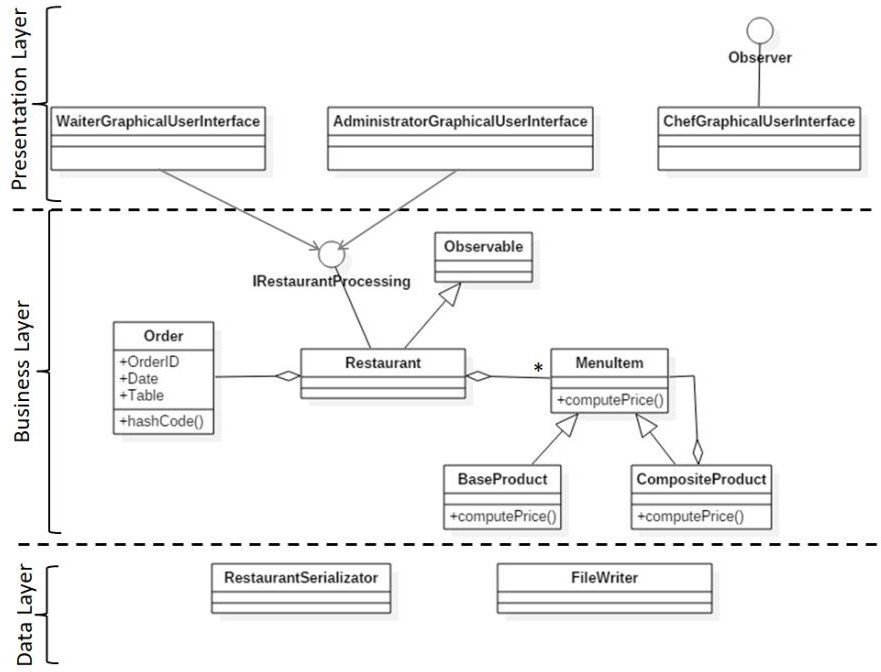
Consider implementing a restaurant management system. The system should have three types of users:

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 that is ordered through a waiter.

Consider the system of classes in the diagram below.



The secondary objectives are:

* Object-oriented programming design
* Classes with maximum 300 lines
* Methods with maximum 30 lines
* Java naming conventions
* Basic documentation
* Implement the class diagram from the homework specification. Choose wisely the appropriate data structures for saving the Orders and the MenuItems.
* Graphical interface :
  + Window for Administrator operations: add new MenuItem, edit MenuItems, delete MenuItems, view all MenuItems in a table (JTable)
  + Window for Waiter operations: add new Order, view all Orders in a table (JTable), compute bill for an Order
* jar file - the application should permit to be run with the following command:

**java -jar PT2020\_Group\_LastName\_FirstName\_Assignment\_4.jar**

* Queues should open/close dynamically. Initially all queues are closed. When clients are distributed to the queues, they become open as needed. When a queue becomes empty, it is closed, and the corresponding thread is paused.
* For the create order command:
  + Create a bill for each order as.pdf file
  + The product stock will be decremented after the order is finalized.
  + In case that there are not enough products, the order will not be created and the PDF document representing the bill will not be generated. In this case, instead of the bill, a PDF document will be generated in which an under-stock message will be written.
* Quality of the Documentation
* Use Composite Design Pattern for modelling the classes MenuItem, BaseProduct, CompositeProduct.
* Create bill in .txt format.
* A Design by contract: preconditions and postconditions in the IRestaurantProcessing interface. Implement them in the Restaurant class using the assert instruction. Define an invariant for the class Restaurant.
* Window for Chef user: use Observer Design Pattern to notify each time a new Order

is added

* Save the information from the Restaurant class in a file (i.e. **restaurant.ser**) using serialization. Load the information when the application starts. Consequently, the application should permit to be run with the following command:

**java -jar PT2020\_Group\_LastName\_FirstName\_Assignment\_4.jar restaurant.ser**

# Analysis

The application should allow the user to select 3 of the following roles : Administrator, Chef or Waiter.

After that, the required operations for each role should be allowed to be performed using the Graphical User

Interface. To simplify the application, it was specified that there may be only one administrator, one waiter

and one chef, and for the moment, there will be no need for a login process.

The following problems have been analyzed and solved, for the implementation of the application :

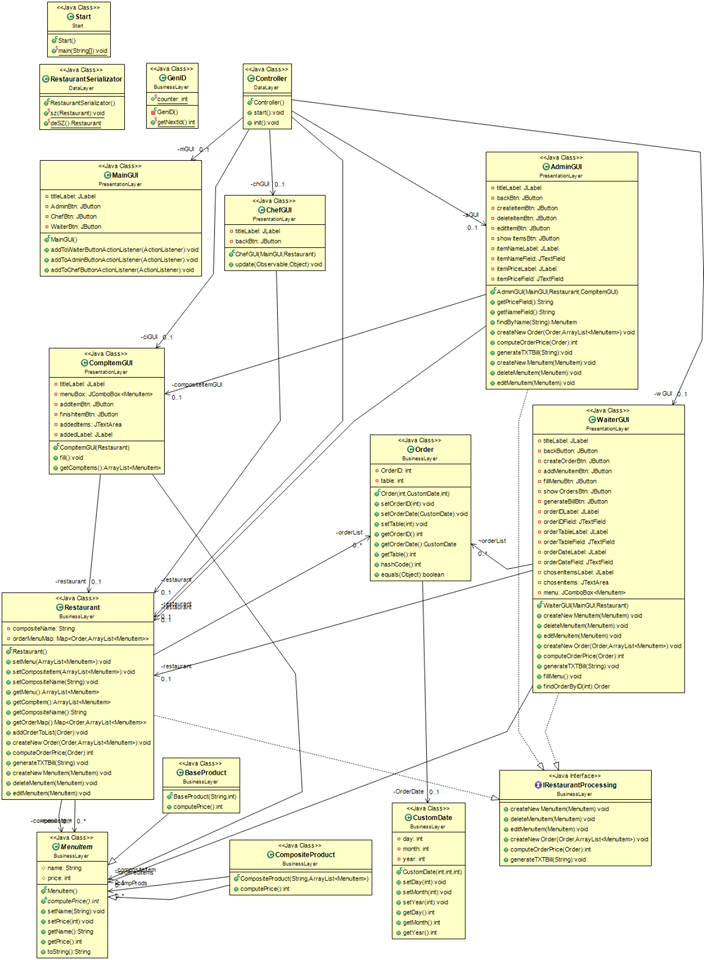
* 1. Define the interface IRestaurantProcessing containing the main operations that can be executed by

the waiter/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 MenuItem, BaseProduct and CompositeProduct
  + - 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 that is based on a hashtable data structure. The hashtable key will be generated based on the class Order, which can have associated several MenuItems. Use JTable to display Restaurant related information.
     + Define a structure of type Map<Order, Collection<MenuItem>> 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 hashCode() method will be overwritten to compute the hash value within the Map from the attributes of the Order (OrderID, date, etc.).
     + Define an appropriate collection consisting of MenuItem objects to store the menu of the restaurant.
     + Define a method of type “well formed” for the class Restaurant.
     + Implement the class Restaurant using Design by Contract method (involving pre, post conditions, invariants, and assertions).
  2. The menu items for populating the Restaurant object will be loaded/saved from/to a file using Serialization.

# Design

The UML diagram of the whole project can be seen here, and next will be described each part of it:



For the design of the project, 4 packages were created :

BusinessLayer – contains the 8 classes needed for the project which contain the data we will be working with

DataLayer – contains the the serialization procedure and the controller for starting up the project

PresentationLayer – contains the interfaces needed for each user

Start – starts the whole project

The BusinessLayer contains 8 classes : BaseProduct, CompositeProduct, CustomDate, GenID,

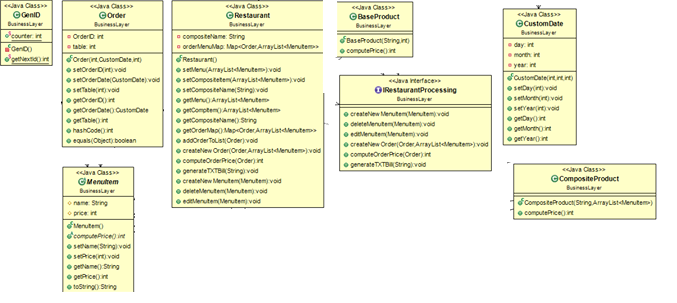
IRestaurantProcessing, MenuItem, Order and Restaurant.

It contains the products used in the menu, a custom data for easier management, an ID Generator for

the orders, the order object, the interface for commands used by the users and the restaurant class.

All of them will be described in the next section.

The UML diagram of this Package can be seen here:

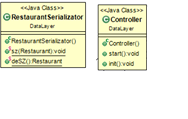


The DataLayer package of the project contains 2 classes : Controller and RestaurantSerializator.

The Controller class is used to start the Graphical User Interface of the project, and RestaurantSerializator is used for the Serialization part of the project.

They will be explained in the next section.

Here is the UML diagram of this part of the project:



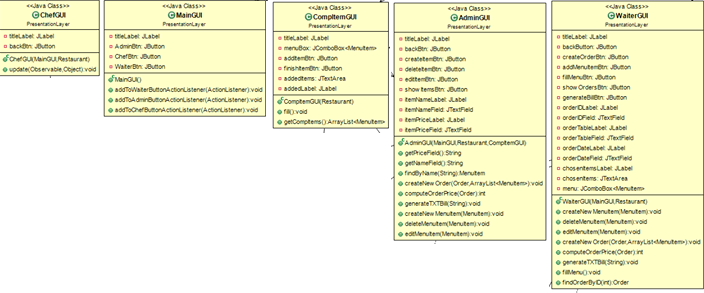
The PresentationLayer package of the project contains 5 classes : AdminGUI, ChefGUI, WaiterGUI,

CompItemGUI and MainGUI.

These Classes Are the GUI part of the project, which help any user have an easier time accessing the

application.

Here is the UML diagram of this part of the project:



Lastly, we have the package Start which contains only the start class, class used to start the whole project.

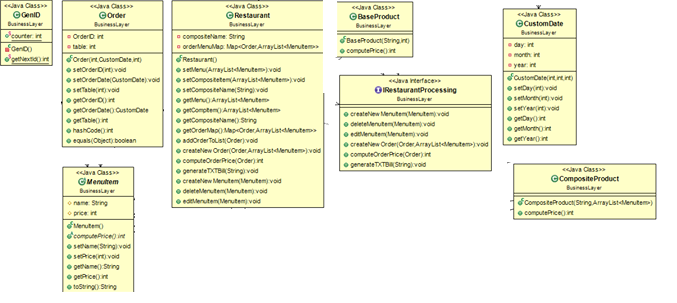
This is the UML for the last package :



Finishing with the Design part of the project, the algorithms were a bit rough to understand at first, but easy to use after I understood their individual concepts. In the next section I will present how I implemented them.

# Implementation

## BusinessLayer



As I have said before, this class is used to help with the creation of data in the project.

First we have the 3 classes : BaseProduct, CompositeProduct and MenuItem which work together, by using

the Composite Design Pattern. MenuItem is an Abstract Class which has 2 attributes, name and price for an

item that will be created and put in the menu. It also has getters and setters for the price and name. The

created Item can either be a BaseProduct or a CompositeProduct. It also has an abstract method called

computePrice() which will be used to compute the price for the item. The BaseProduct class

is used for creating just a simple product to be sold (water ,juice, coffee, etc.), it has a single constructor

used to create it. It then uses the method from MenuItem to compute the price, which is literally the price of

the item. The third class used, CompositeProduct is used to create a complex product built from other

simple products, this is why it contains 1 attribute which is an ArrayList of multiple MenuItem used to

create the respective product. It has 1 constructor which will name the new product, and show the contents

of it. It will use the computePrice() method from MenuItem to create the price, this time, it will iterate

through the ArrayList and sum the price from each item.

Next, we have the Order Class which is used to help the waiter create orders for clients. The class has 3

attributes, an OrderID, an OrderDate and a Table which the order will be delivered to. It also has getters and

setters for the attributes. It also has the hashCode() method used to create a hash value for the Map and the

overriden equals() method. For the Date I have created a CustomDate class for easier management of the

date(I have decided that the format of the date will be “DD-MM-YYYY”, for example “03-12-2020” – the

user is required to type the date as shown in the example which means to put a “0” in front if the number of

the date or month is single digit). For the OrderID a class GenID has been created which generates an ID for

the order(There won’t be any problem with generating an ID since it is done in and ascending order, starting

from 1).

Lastly, the IRestaurantProcessing interface and Restaurant class which implements it. The interface has the

methods which will be used in the Restaurant class and the AdministratorGUI and WaiterGUI. The

Restaurant class 5 attributes, menu which contains all the items available to be ordered, compositeItem

which is an ArrayList of MenuItem, compositeName which will be the name for each product,

orderMenuMap which is the HashMap containing the orders and their specific ordered items, and finally

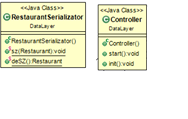
orderList which is an ArrayList of type Order, which contains all the orders. The class has a constructor for

the menu, orderMenuMap and orderList, and getters and setters for every attribute used. The other methods

are the implementations from IRestaurantProcessing interface.

This concludes the BusinessLayer package.

## DataLayer

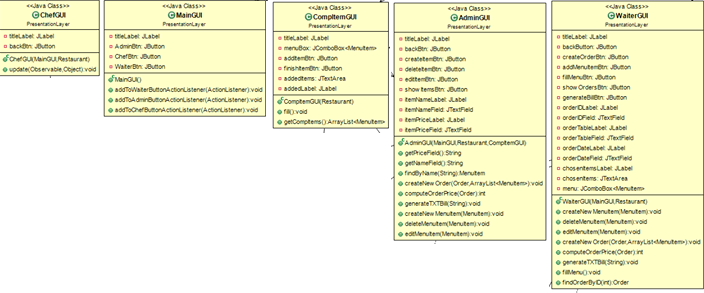


The RestaurantSerializator class contains the two methods used to serialize and deserialize the application, so that the Data which has been already put in the project, doesn’t get deleted when the application is closed. The serialization method, called sz() creates a file name “Restaurant” of type “ .ser ” => “ Restaurant.ser ”, in which the data will be stored. Next the deserialization method called deSZ(), is used to gather the data from “Restaurant.ser”.

Next, the Controller class is used for the GUI part of the project. It contains 6 attributes, 5 of them being all the objects from the PresentationLayer and the Restaurant object which will get the deserialized data from “ Restaurant.ser ”. There are 2 methods, start() and init(), the start() method creates the 5 GUIs needed and gets the data for the Restaurant object, and the init() initializes the buttons which will be used in the MainGUI object.

This concludes the PresentationLayer package.

## PresentationLayer



This package contains the Graphical – User – Interface part of the project.

The first important class is the MainGUI class, which contains a message for the user, which

will tell him to choose one of the 3 other interfaces : AdminGUI, ChefGUI and WaiterGUI. It

functions as a Main Menu.

First, the AdminGUI will let the Administrator create, edit or delete items which will be

added to the menu. The Administrator will be able to write a name and a price for an item,

and then he will be able to create it. Then, he can delete an item just by typing its name and

pressing on the delete button. With the edit item, he just has to type the name and the new

price for the item. Whenever he wants to see if a new item has been added, or an older one

has been deleted or edited, he has to press the “Show items” button which will display all of

the existing items. Also the AdminGUI uses the CompItemGUI, which its only purpose is to

store the created items. It also has a back button which takes the user back to the main menu.

Secondly, the WaiterGUI will let the waiter create a new order, by typing the OrderID, the

Table and the Date (the format of the date is “DD-MM-YYYY”, and required to be 2 digits in

the day and month section). Next he/she will click on the “ Show Menu ” button to fill the

box above with the items in the menu. Then the waiter will select each needed item and press

on the “ Add ” button to put it in the order. All of the selected items will be shown in a box,

under the “ Chosen Items ” title. Next he/she will click on the “ Create Order ” button which

will create a notification and instantly send it to the chef. Then he can generate the bill for the

order by clicking on the corresponding button. He can also list all the orders which have been

created. It also has a button to go back to the main menu.

Lastly, the ChefGUI is pretty simple in design, this is where a notification with all the details of an order will pop up. There is also a button which takes the user back to the main menu.

## Start

STARTUML

In Start we simply run the project. A Controller object is created, and its start method is called to start the

project

# Results

In terms of verifying the correctness of the algorithms, 3 objects have been created by the admin – Potato,

Juice and Water – the waiter created an order with only the first two, and the chef has received the

notification with the order. Also the waiter clicked on the generate bill button and the bill has been created.

All the information has been saved in “ Restaurant.ser ” and the project closed.

Project has been opened up again and all the previous information was available from the start.

# Conclusions

The created application is a very easy to use, fast and correct Restaurant Management System.

In my opinion the most important updates that can be brought onto this projects are the addition of more Quality of Life features, such as an improvement to the Graphical User Interface, for example, the Chef being able to interact with the orders, mark them as finished or deny them.

Or a tip system for the waiter, which will depend on the satisfaction of the customer. Also a customer interface could be implemented which will act as a fast and simple review of the customer (for starters, a simple grading from 1 to 10 or 1 to 5). Maybe also add more than 1 admin, or 1 waiter and 1 chef, create a login system for everyone with customizable menu screens for everyone. Also a shift manager, so that every employee knows when his or her shift is over, or for the administrator to manage and change shifts.

This project help me understand how to create a more complex Graphical – User – Interface and finally how to save the written data by the user, in a way more elegant manner. It also helped me understand concepts of structuring both the project as a whole and the smaller, more organized, mini - projects, the uses of encapsulation, and I also learned how to organize my code, so that when I come back to improve this project, I won’t have a hard time understanding what I did in each part.