

MINISTRY OF EDUCATION AND RESEARCH



TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 4

FOOD DELIVERY MANAGEMENT SYSTEM

1. Requirements

Design and implement a food delivery management system for a catering company. The client can order products from the company's menu. The system should have three types of users that log in using a username and a password: **administrator**, **regular employee**, and **client**.

The **administrator** can:

- Import the initial set of products which will populate the menu from a .csv file.
- Manage the products from the menu: add/delete/modify products and create new products composed of several products from the menu (an example of composed product could be named "daily menu 1" composed of a soup, a steak, a garnish, and a dessert).
- Generate reports about the performed orders considering the following criteria:
 - time interval of the orders – a report should be generated with the orders performed between a given start hour and a given end hour regardless the date.
 - the products ordered more than a specified number of times so far.
 - the clients that have ordered more than a specified number of times so far and the value of the order was higher than a specified amount.
 - the products ordered within a specified day with the number of times they have been ordered.

The **client** can:

- Register and use the registered username and password to log in within the system.
- View the list of products from the menu.
- Search for products based on one or multiple criteria such as keyword (e.g., "soup"), rating, number of calories/proteins/fats/sodium/price.
- Create an order consisting of several products – for each order the date and time will be persisted and a bill will be generated that will list the ordered products and the total price of the order.

The **employee** is notified each time a new order is performed by a client so that it can prepare the delivery of the ordered dishes.

Consider the system of classes in Figure 1 as a starting point for the system design. Other classes and packages can be added to design and implement the full functionality of the application.

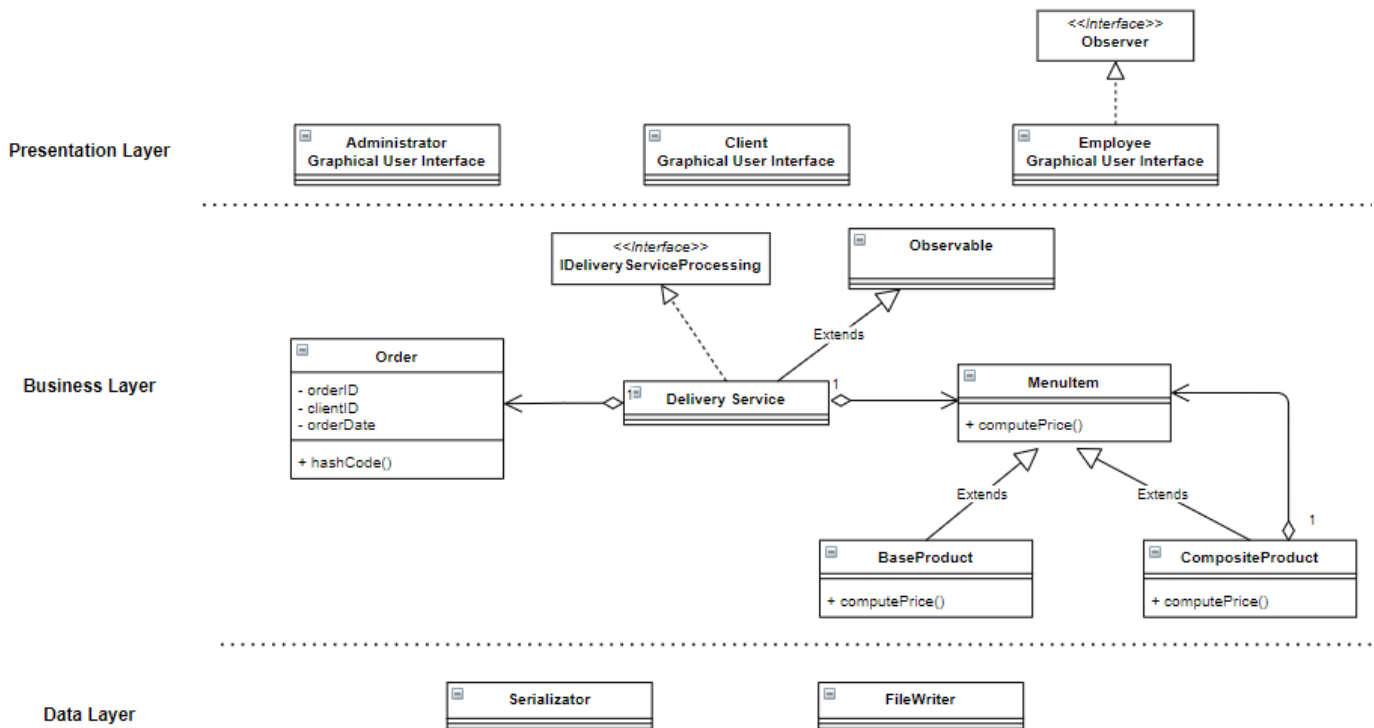


Figure 1: Class diagram to be considered as starting design of the system.

To implement the system, consider the following:

- 1) Define the interface `IDeliveryServiceProcessing` containing the main operations that can be executed by the administrator and client, as follows:
 - Administrator: import products, manage the products from the menu, generate reports
 - Client: create new order which implies computing the price for an order and generating a bill in .txt format, searching for products based on several criteria.
- 2) 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 employee each time a new order is created.
- 3) Implement the class `DeliveryService` using a predefined JCF collection which uses a hashtable data structure. The hashtable key will be generated based on the class `Order`, which can have associated several `MenuItems`.
 - Define a structure of type `Map<Order, Collection<MenuItem>>` for storing the order related information in the `DeliveryService` 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 a structure of type `Collection<MenuItem>` which will save the menu (i.e., all the products) provided by the catering company. Choose the appropriate collection type for your implementation.
 - Define a method of type “well formed” for the class `DeliveryService`.
 - Implement the class using the Design by Contract technique (involving pre, post conditions, invariants, and assertions).
- 4) The base products used initially for populating the `DeliveryService` object can be loaded from the **products.csv** file (adapted from [Link](#)) using lambda expressions and stream processing. *Note: the administrator can manually add other base products as well.*
- 5) The menu items, performed orders and user information will be persisted using serialization so as to be available at future system executions by means of deserialization.

2. Deliverables

- A **solution description document** (minimum 2000 words, Times New Roman, 10pt, Single Spacing) written in the template provided on the laboratory Web site.
- **Source files, JavaDoc files** including the custom tags and descriptions associated to the defined pre, post conditions and invariants – will be uploaded on the personal **gitlab** account created according to the instructions in the **Laboratory Resources** document, and following the steps:
 - Create a repository on **gitlab** named according to the following template *PT2022_Group_FirstName_LastName_Assignment_4* – the repository should be placed in the group named according to the template below: *PT2022_Group_FirstName_LastName*
 - Push the source code and the documentation (**push the code not an archive with the code**)

Make sure that you give access to your group, to the PT lab assistants. On your Group page, go to: Members → Invite Member → and offer Maintainer rights for the gitlab user named **utcn_dsrl**.

3. Evaluation

The assignment will be graded as follows:

Requirement	Grading
Minimum to pass <ul style="list-style-type: none"> • Object-oriented programming design, classes with maximum 300 lines, methods with maximum 30 lines, Java naming conventions • Implement the class diagram from Section 1. Choose appropriate data structures for saving the <i>Orders</i> and the <i>MenuItems</i>. • Define the class <i>BaseProduct</i> with the following fields: title, rating, calories, proteins, fats, sodium, price. Read the data from the file <i>products.csv</i> using streams and split each line in 7 parts: <i>title, rating, calories, protein, fat, sodium, price</i>, and create a list of objects of type <i>BaseProduct</i>. NOTE: the file contains duplicate dishes so make sure you select only one. • Graphical interface: <ul style="list-style-type: none"> ○ Log in window 	5 points

<ul style="list-style-type: none"> ○ Window for Administrator operations (see the requirements in Section 1) ○ Window for Client operations (see the requirements in Section 1) • Use lambda expressions and stream processing for generating the administrator specific reports (see Section 1). • Use lambda expressions and stream processing to implement the search functionalities available to the client (see Section 1). • Good quality documentation covering the sections from the documentation template. 	
Use the Composite Design Pattern for modelling the classes MenuItem, BaseProduct, CompositeProduct.	1 point
Create bill in .TXT format.	0.5 points
Design by contract: preconditions and postconditions in the <i>IDeliveryServiceProcessing</i> interface. Implement them in the <i>DeliveryService</i> class using the assert instruction. Define an invariant for the class <i>DeliveryService</i> . Generate the corresponding Javadoc files which should include the custom tags and descriptions associated to the defined pre, post conditions and invariants.	1.5 points
Window for the employee user: use Observer Design Pattern to notify each time a new Order is added.	1 point
Save the information from the <i>DeliveryService</i> class in a file (i.e., file.txt) using serialization. Load the information when the application starts.	1 point

4. Bibliography

- **Lambda expressions and stream processing**
 - <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
 - <https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html>
 - <https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html>
 - <https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>
 - <https://howtodoinjava.com/java8/java-stream-distINCT-examples/>
- **Java serialization**
 - http://www.tutorialspoint.com/java/java_serialization.htm
 - <https://www.baeldung.com/java-serialization>
 - <https://www.geeksforgeeks.org/serialization-in-java/>
 - <https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html>
- **Java HashMap**
 - <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>
- **Java assert**
 - <https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
 - <http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>
 - <http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>

- <https://intellij-support.jetbrains.com/hc/en-us/community/posts/207014815-How-to-enable-assert>
- **Adding custom tags to javadoc**
 - <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html#tag>