

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 4

RESTAURANT MANEGEMENT SYSTEM

Principal Objective

Implementation of 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.

Secondary Objectives

Create Restaurant class that englobes Arrays that describe the Orders and Menu items also a hashtable that bind together the Orders taken and the menu items...........................................14

Create BaseProduct class that abstractize a basic product in a restaurant and a CompositeProduct class that represent a product that has one or more base products objects in their composition.............................................................................12

Create graphical user interface for administrators, chefs and waiters.....................................................................................17

Create Observer and Subject interfaces for implementing the Observer Design Pattern where the graphical user interfaces implement the Observer interface and where the subject is the Restaurant object used...........................................................13

Serialize and deserialize the Restaurant object used – by creating a separate Class RestaurantSerializator responsable for deserialization of the object from a file and serialization of the object in the same file...........................................................................................16

Problem analysis

1. Understantding the problem

Restaurant management involves a lot of processes that, if done right, would ensure the smooth flow of and enterprise and even increased profits. It allows for the improvement of worker efficiencies and customer experience while reducing costs.

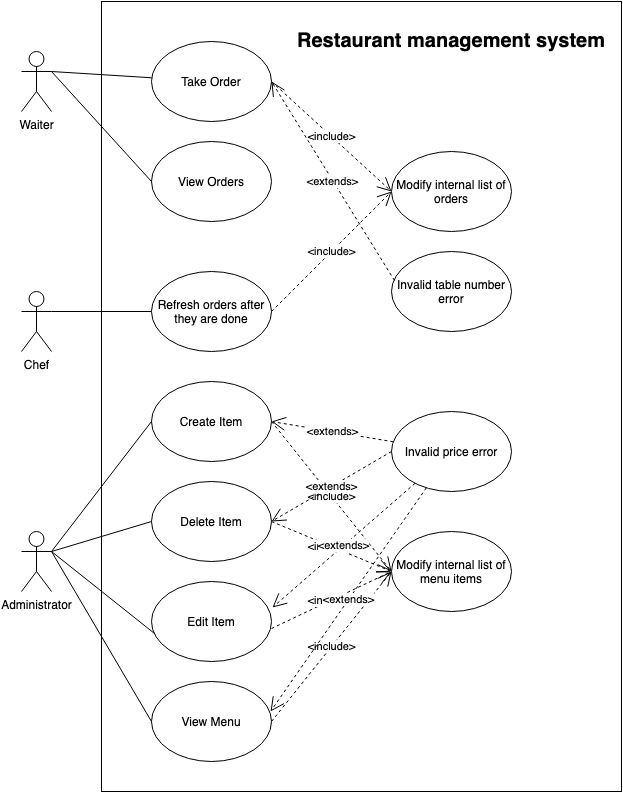
1.1 Approach

This problem is approached with a model view controller design method where the view, or better say the views, are the graphical user interfaces for the employees: administrator,chef and waiter and the model is represented by a Restaurant class Object used to hold the information regarding the menu items and taken orders. The observer design patters is used for notifying all the graphical interfaces when an update (new menu item or item) occurs for keeping the views synchronized.

1.2 Potential risks

The most obvious problem that can occur is the unsynchronization of the three views, of administrator, chef and waiter. To avoid this the Observer design pattern mush be well implemented.

Scenarios and use-cases



Main Success Scenarios:

1. Create Order use-case: In the waiter graphical user interface, the waiter must completed the table text field that refers to the table that made the order and a number of menu items must be selected that will be included in the order.

After all the desired information was inserted the waiter can press the create button.

1. Create Bill use-case: After the order was taken, the waiter can press the Bill button to generate a bill in txt format that will contain the products recently ordered with their prices and the total price.
2. View Orders use-case : This use-case is used to show in a JTable the Orders taken.
3. Refresh orders use-case : This use-case is present in the chef graphical user interface. There the orders are presented with the requested products that the chefs should prepare. After they are done preparing an order they can check is as “Done” then press refresh button so the list of orders can be refreshed.
4. Create Item use-case: This use-case can be accessed via the administrator graphical user interface. The administrator can insert the name of the new menu item in the specific text field. If the product is a basic one then the price text filed must be completed and no other product should be selected to be a part of the new product composition. If the product is a composite product, then all the component products must be selected and the price will be autommatically computed as the sum of prices of component products.
5. Delete Item use-case: This use-case can be accessed via the administrator graphical user interface. The administrator can delete a number of menu items form the menu by selecting them and pressing the “delete" button.
6. Edit Item use-case: This use-case can be accessed via the administrator graphical user interface. In the “Edit Item” tab the item that must be edited is selected and the name price and component products are selected as desired.
7. View Items use-case : This use-case is used to show in a JTable the items in the menu.

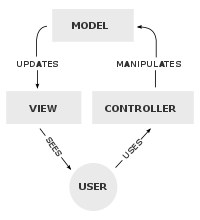
Alternative failure Scenarios:

1. Invalid number error – This is an error case where the table number was not correctly inserted or not inserted at all and a GUI error is displayed.
2. Invalid price error – This is an error case where the price of the inserted item is not correctly inserted.

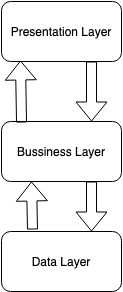
Design

Using MVC model

Model–view–controller[[1]](https://en.wikipedia.org/wiki/Model–view–controller) (usually known as MVC) is a [software design pattern](https://en.wikipedia.org/wiki/Software_design) commonly used for developing [user interfaces](https://en.wikipedia.org/wiki/User_interface) which divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. This kind of pattern is used for designing the layout of the page.

The Model View Controller is suitable for this restaurant management system. The graphical used interfaces for admin, chef and waiter is representing the view that is controlled by a controller that keep it in sync with the Model represented by the Restaurant object used.

Layered Architecture

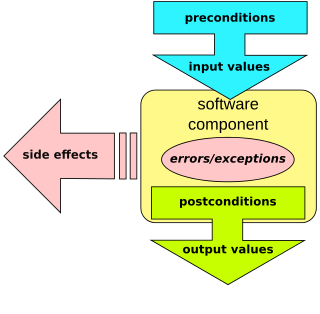
The application is builded around a layered architecture. There is a business Layer that resolve the back end logic of the application, the presentation layer that is responsable for the graphical user interface of the application, and the data layer that si responsible, for this application, for serialization and deserialization upgradable or “in constant change” objects in business layer.

Observer Pattern[[2]](https://en.wikipedia.org/wiki/Observer_pattern)

The observer pattern is a [software design pattern](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) in which an [object](https://en.wikipedia.org/wiki/Object_(computer_science)" \l "Objects_in_object-oriented_programming), called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their [methods](https://en.wikipedia.org/wiki/Method_(computer_science)).

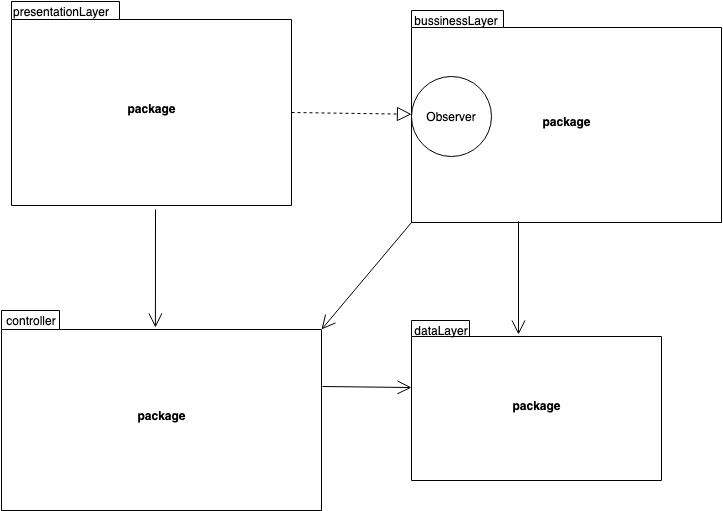
In this application the Observers are represented by the GUI classes that must be updated every time is necessary. The menu or the list or orders can be updated by every graphical user interface in a specific way and every other graphical user interface must be updated as a result of that. The updatable content – the subject of the Observer design Pattern – resides in the Restaurant Class – that is also serialized at the exit of the applicatin for consistency.

Design by contract [[3]](https://en.wikipedia.org/wiki/Design_by_contract)

Design by contract (DbC), also known as contract programming, programming by contract and design-by-contract programming, is an approach for [designing software](https://en.wikipedia.org/wiki/Software_design).

Since the Restaurant class is so important, containing all the sensitive information that is the base for the aplication like the list of menu items, list of orders, hash table with the ordered products. This class is implemented using design by contract where the preconditions and postconditions are verified using java assertions. Also an invariant[[4]](https://stackoverflow.com/questions/8902331/what-is-a-class-invariant-in-java) method called “wellFormed()" is used to check must always return the boolean value true in order or the program to continue.

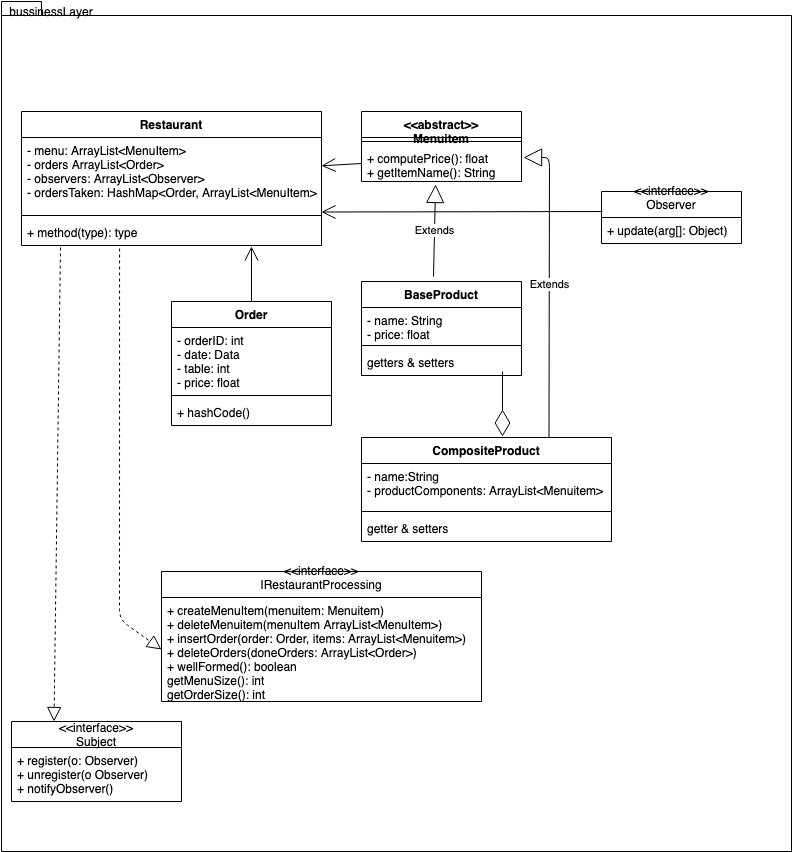
UML Diagram



The packages used for this application are:

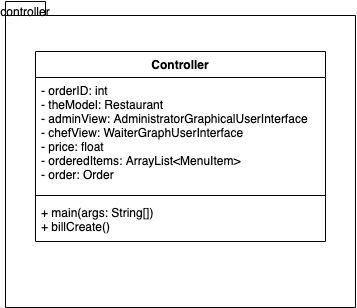
* PreentationLayer: representing the graphical user interface – all of the representative classes are implementing the Observer interface and representing the observers in the Observer design patter.
* BusinessLayer: this is the package that contains the logic for creating and sustaing the Restaurant instantiation.
* DataLayer: this is the package that contains the logic for serialization and deserialization of the Restaurant object
* Controller: This package represent the Controller in the MVC design.

BusinessLayer package



In the businessLayer package resides the Restaurant class that implements the IRestaurantProcessing interface so the implementation can be equiped with method that for using design by contract and also Subject interface to notify, register and unregister observers. Order, Base Products and Composite products classes are implemented here to be used in the Restaurant class.

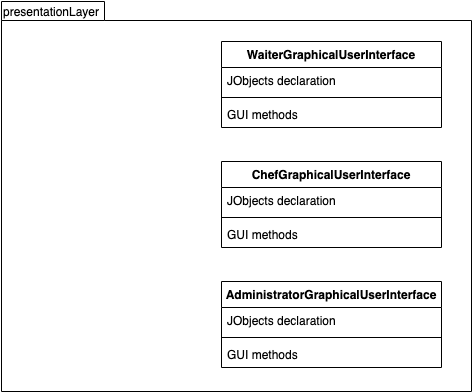
Controller Package

The controller package contains the Controller class used to implement the logic for controlling the GUI, the backend from the businessLayer and syncronizing them.

DataLayer package

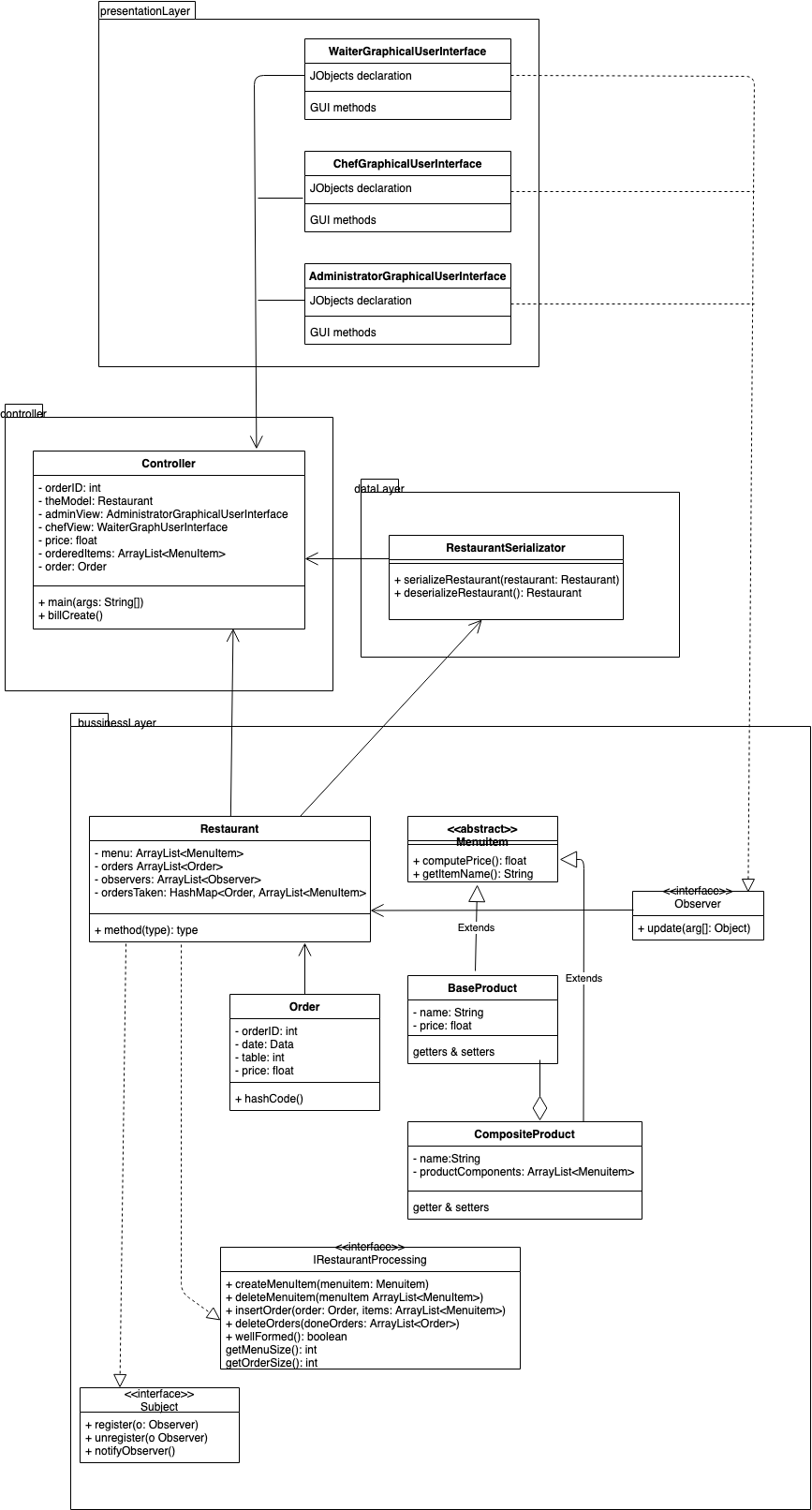
This package contains the RestaurantSerializator class that implements the logic for serialization and deserialization of an object – we will use the Restaurant type Object.

PresentationLayer package



This package is containing the 3 graphical user interface used for the administratorof the restaurant, for the chef and for waiters respectively.

It contains only UI logic ment to provide an intuitive look for all the actions that a specific agent can take in syncronisation with all the other agents.



Mentionable datastructures:

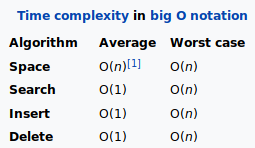
Hash table [[5]](https://en.wikipedia.org/wiki/Hash_table)

In [computing](https://en.wikipedia.org/wiki/Computing), a hash table (hash map) is a [data structure](https://en.wikipedia.org/wiki/Data_structure) that implements an [associative array](https://en.wikipedia.org/wiki/Associative_array) [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type), a structure that can map [keys](https://en.wikipedia.org/wiki/Unique_key) to [values](https://en.wikipedia.org/wiki/Value_(computer_science)). A hash table uses a [hash function](https://en.wikipedia.org/wiki/Hash_function) to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. During lookup, the key is hashed and the resulting hash indicates where the corresponding value is stored.

The hash table is used for storing the orders and their ordered products:

* Keys – represented by the Order Object
* Value – represented by an array of products

The usage of hashtable reduces the search time in the program.



Imlpementation

BaseProduct Class

This class represent the abstractisation of a base product that has as only attributes name and price.

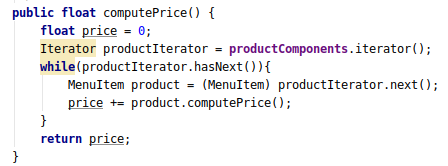


CompositeProduct Class

This class represent the abstractisation of a composite product that has in composition one or more basic products. The only attributes are name and the array of products that are in the composition of the composite product.



The application was designed to compute the price of the composite product based on the contained products only via the compute price method.



Menu Item abstract class

This abstract class is used to declare the method implemented in the products featured in the menu of the restaurant . The two mehtods are:

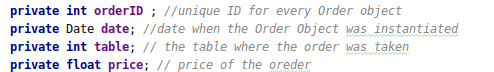
* Abstract public float computePrie()
* Abstract public String getItemName()

Observer Interface

An interface that is used to to implement the Classes that will implement the observers in the Observer Design Pattern. The only method required is the update method that takes as parameter a vector of Objects.

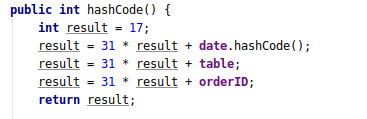
Order class

This class is used to abstractize an order that is taken at a restaurant. The attriutes are the following:



The Order type Objects are used as keys in a hashTable that has as value the array of products that has been ordered in that order.

In order for the hashtable to work with as least colisions as possible the hashCode function is Overriden in this class.



The reason why prime numbers are used is to minimize collisions when the data exhibits some particular patterns.

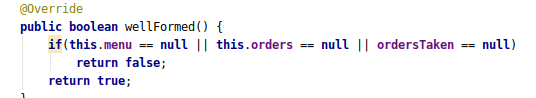
If the data is random then there’s no need for a prime number, you can do a mod operation against any number and will have the same number of collisions for each possible value of the modulus.

Restaurant Class

This class represents the abstractisation of the restaurant with the logic implemented correspondingly. The attributes are :

* ArrayList<MenuItem> menu - represents the array of items in the menu
* ArrayList<Order> orders – represents the array of the orders taken and not completed
* ArrayList<Observer> observers – an array of observers that need to be notified using the notifyAll method
* HashMap<Order, ArrayList<MenuItem>> ordersTaken – the hashtable with the Order object as key and the array of requested items as value.

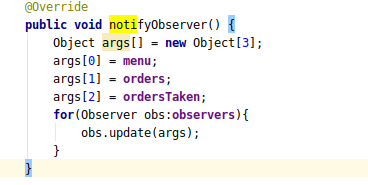
An important method in this class is the wellFormed() method that represents an invariant for the class - it must return the boolean value true every time the program is correctly running. It basically verifies if the arrays and hashtable are instantiated.



The class implements the IRestaurantProcessing interface that requires it to impelmet the following methods:

* CreateMenuItem(MenuItem menuItem) - that creates and adds a menu item in the menu array.
* DeleteMenuitem(ArrayList<MenuItem> menuItem) - the menu items passed to this function are deleted from the menu array.
* DeleteOrders(ArrayList<Order> doneOrders) - the orders passed to this function are deleted from the orders array
* InsertOrder(Order order, Arraylist<MenuItem>) - this method is used to insert an ordewith the specific items in the orderstaken table.

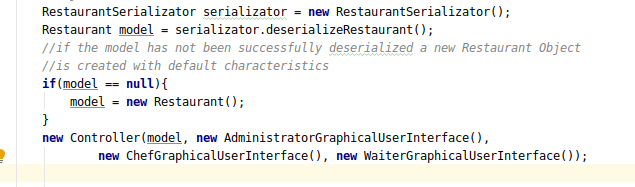
This class implemets the Subject implementation so the register unregister and notifyObserver methods are implemented. The notifyAll methds updates the the observers after the menu array, orders array, and ordersTaken hashtable are renewed:



The Controller Class

The instantiation of this class will be used as a controller in the MVC design.

In this class we also choose to implement the main class that contains an instantiation of the constructor. As discused above the restaurant object represents the model of the program and we want to deserialize it at the beginning of the program and serialize it at the end of the program.

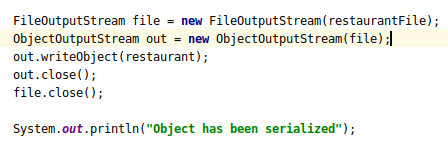
The most important attributes are : Restaurant theModel, AdministratorGraphicalUserInterface adminView, WaiterGraphicalUserInterface waiterView, ChefGraphicalUserInterface chefView, that represent the model and the multiple views of the program.

A very important aspect of the controller is that it binds the listener type classes to the approapiate view.

RestaurantSerializator class

This class is used for serialization and deserialization of a Restaurant Object via the methods serializeRestaurant and deserializeRestaurant.

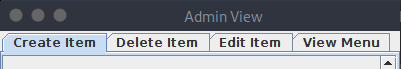
serializeRestaurant(Restaurant restaurant) - The file where the Restaurant object is stored is named “RestaurantFile.ser” and if it does not exist it will be created first.



AdministratorGraphicalUserInterface class

This is the class that contents the logic and design for the administrator graphical user interface. Most of the attributes are used for the swing framework. One important attribute is the array of menu items that is displayed in the GUI.

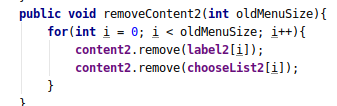
The main Pane for this interface is of type JTappedPane containing 4 tabs: Create Item, Delete Item, Edit Item, View Menu – the titples are quite self explanatory.



The last tab is represented as a JTable and the other 3 are panels with a box layout along the PAGE-AXIS. For example for the second tab we have a content2 of type JPanel:



Every time the content is update – i.e the menu of the application is changed the content is changed also. This is done by first eliminating all the old content:



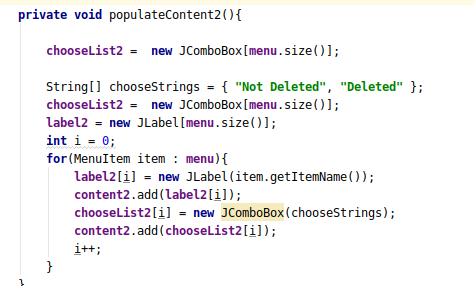
As it can be seen in the removeContent2 method all the components that might be removed are declared globally and the removed. The number or components to be removed is stored in the argument oldMenuSize.

Here the JLabel and JComboBox type objects are removed. They are displayed in the following format:



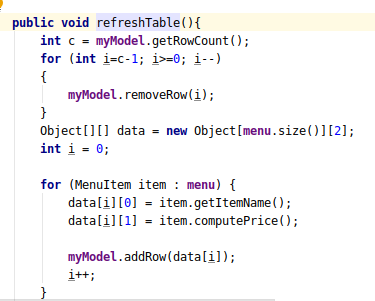
For every panel the format is different and so is the method that does the removing of the componets but the result must be the same: remove all the content and then update with the new one.

For updating the content after removing, for every panel we call a populateContent method like the following:



Every component (previously removed) is added to the component.

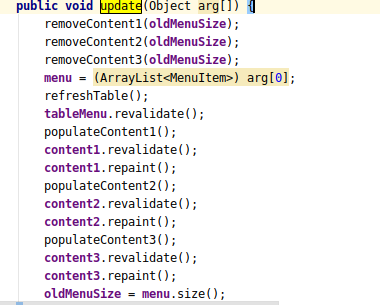
The refreshing of the table is done in a similar manner but the removing and adding of the components – rows in this case – is done in a single function:



In this specifil table the menu items are displayed with their specific fields:



One of the most important methods in this class is the update method – used by the observable Object in the Observer Design Pattern – where all the necesarry removing and repopulating of the content is done:



The vector of arguments contains in this case only the updated menu array.

The backend of the application is noticed every time an item is created, deleted or edited through listeners via the Controller. For example the createItemListener added to the “Create” Button in the view using the method:



The actual logic of the listener is implemented in the CreateItemListener class that has the implementation transmited via the Controller:



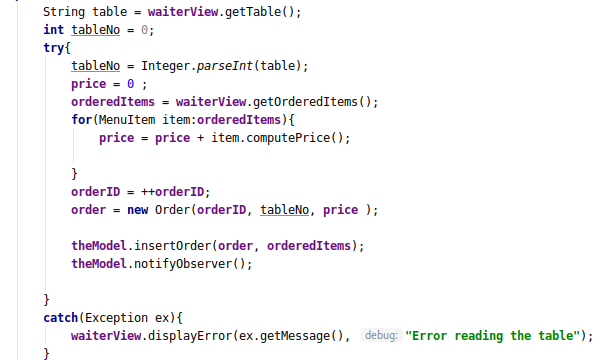
The same MVC strategy is used for all the 3 tabs for can change the internal state of the application, They have specific triggering buttons: Create, Delete, Edit.

WaiterGraphicalUserInterface class

As in the case of the administrator GUI a tabbed pane is used with two tabs: Create Order, View Orders.

The create tab contains a text for filling the table where the order was taken, two buttons: an Order button for creating an order and a Bill button to create a bill and a list of selected items for the order.

Every time the Order button is pressed the listener attached to it is triggerd executing the following logic from the ActionListener type class:

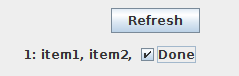
As it can be seen above the table number is tried be parsed – if it did not succeed an error is displayed in GUI. The price of the selected products is computed by summing the individual prices and stores the result in a global variable price. The new order is created and inserted into the array of orders and then all the Observers are notified of the update.

The recently created order is stored locally in the controller class and used for computing the bill when the Bill button is pressed.

ChefGraphicalUserInterface class

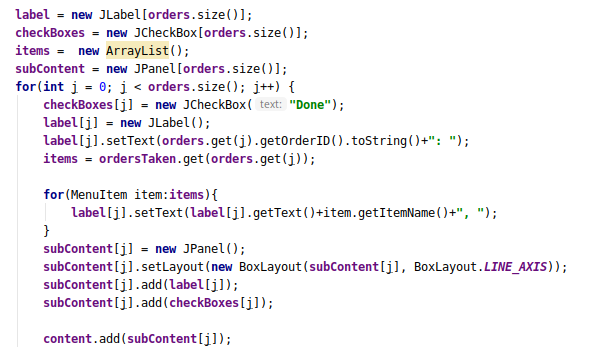
Like the above graphical user interface classes this class implements the Observer interface to be notified every time a new Order is taken.

A list with all the orders and their ordered products are displayed for the chef to prepare. When the chef is done with the order products he or she can check the Done box then press the Refresh button to refresh the orders for displaying only the undone orders.



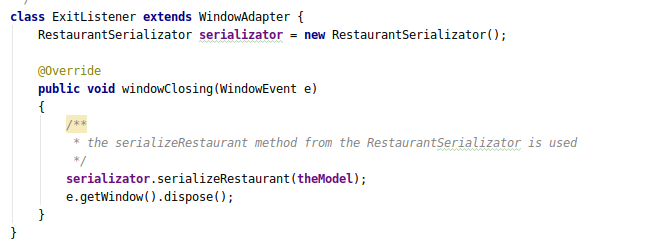
Besides swing components the most important attributes are: the array of orders and the hash table ordersTaken for locally store the orders and products that need to be displayed.

The actual displaying of the content is done in the populateContent method :

Every pair of labels and check boxes are added to a separate BoxLayout type panel. Those pannels are storred in a vector of pannels called subContent and included in the main ”content" panned.

This way when refreshing the content the deletion of the content is done with respect to the checked box .

Every GUI presented has its own Jframe Object that has attached a WindowAdapter listener that is triggered every time a frame exits – i.e the program is closed by the user. This way the call for serialization of the Restaurant object is made before the exit.



Conclusion

The restaurant management system functions as expected for creating, deleting, editing, viewing menu items by the administrator , creating, viewing orders by the waiter type user, and an intuitive and fast way of viewing and refreshing the undone orders for the chef type user. The application is easy to use and any user mistake can be easely signaled and solved. The information stored in the application is stored when the host system is shut down via the serialization and deserialization mechanism.

Bibliography

[1]<https://ro.wikipedia.org/wiki/Model-view-controller>

[2]https://en.wikipedia.org/wiki/Design\_by\_contract

[3]<https://en.wikipedia.org/wiki/Design_by_contract>

[4]<https://stackoverflow.com/questions/8902331/what-is-a-class-invariant-in-java>

[5]https://en.wikipedia.org/wiki/Hash\_table