ITP4507 Assignment

Report for BSLOS

15 Oct 2020

Ho Sze King (190086602)

We declare that this is a group project and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgement is made explicitly in the text, nor has any part been written for us by another person.

Table of Contents

Scenario

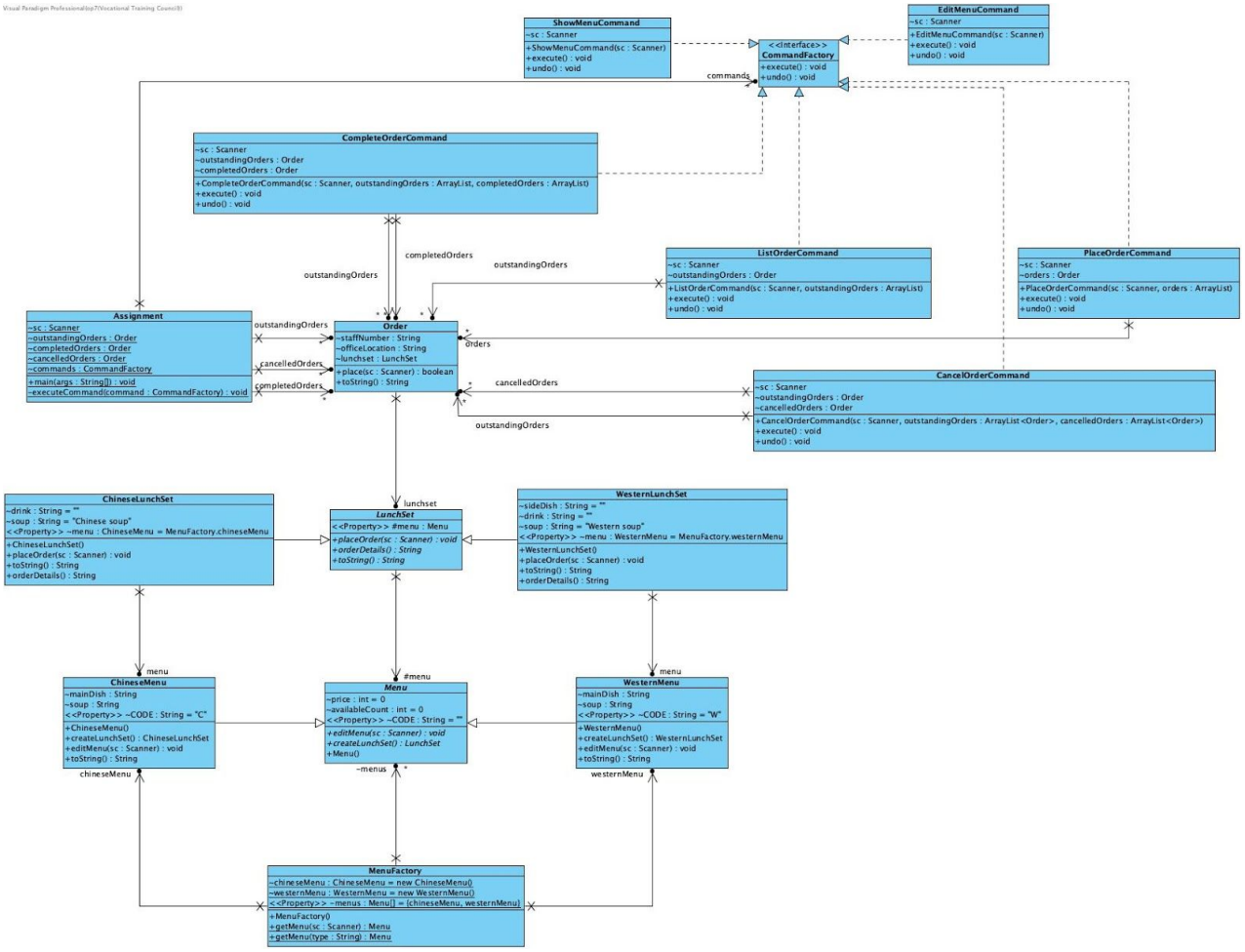
A company has a staff canteen which provides catering service for its staff. Both eat-in and take-away services are provided by the canteen. Recently, due to the COVID-19, restrictions about canteen accommodations for eat-in service have to be observed. The company management wants to encourage the staff to stay in the company for lunch in order to lower the risk of infection. Therefore, the company has decided to subsidize the canteen to provide low-price and high-quality Business Lunch Sets via a new ordering system called Business Set Lunch Ordering System (BSLOS). Staff can order a Business Lunch Set at an attractive price and the canteen will deliver the meal to the staff office directly.

The details of the system design will be covered in this report.

Assumptions

- Menu cannot be deleted. They can however be left empty.
- There should be at most one object for each subclass of Menu. The order is decided by the time of creation.

Application design with class diagram



Please refer to the source code for the full sized diagram.

Design patterns applied to the application

Command Pattern

All commands are stored inside the **CommandFactory** interface. The commands will implement the **execute()** and **undo()** method. The current system does not require the usage of undo() function, but the method has been declared for future implementation. All commands are stored in an ArrayList ***commands***.

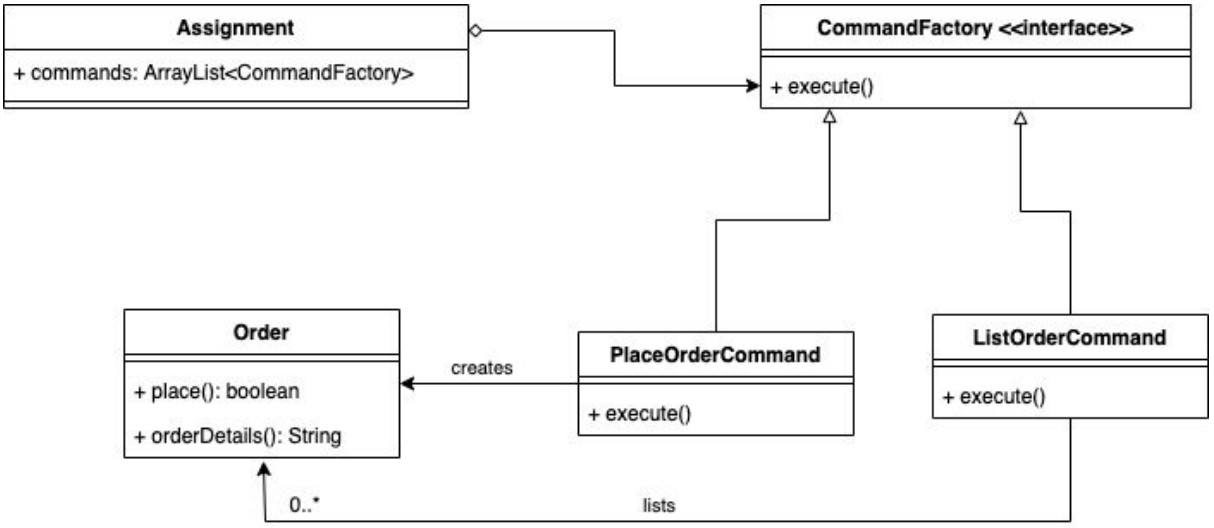


Figure 2: A simplified class diagram indicating how command pattern is implemented. Not all attributes are indicated.

Factory Method

The system follows the Open-closed principle, which means it is “Open for extension and closed for modification”. When designing the system, it has been taken into consideration that new types of lunch sets, or options towards different lunch sets will be changed. In such a case, new classes that extend from **Menu** and **LunchSet** can be created, and appropriate methods can be added towards **MenuFactory**, while the controller of the system and other existing classes do not have to be changed. In Figure 3, the usage of factory pattern is being shown. As there will only be one instance per menu, there will not be **createMenu()** method but only **getMenu()**, as explained in the *Singleton* section.

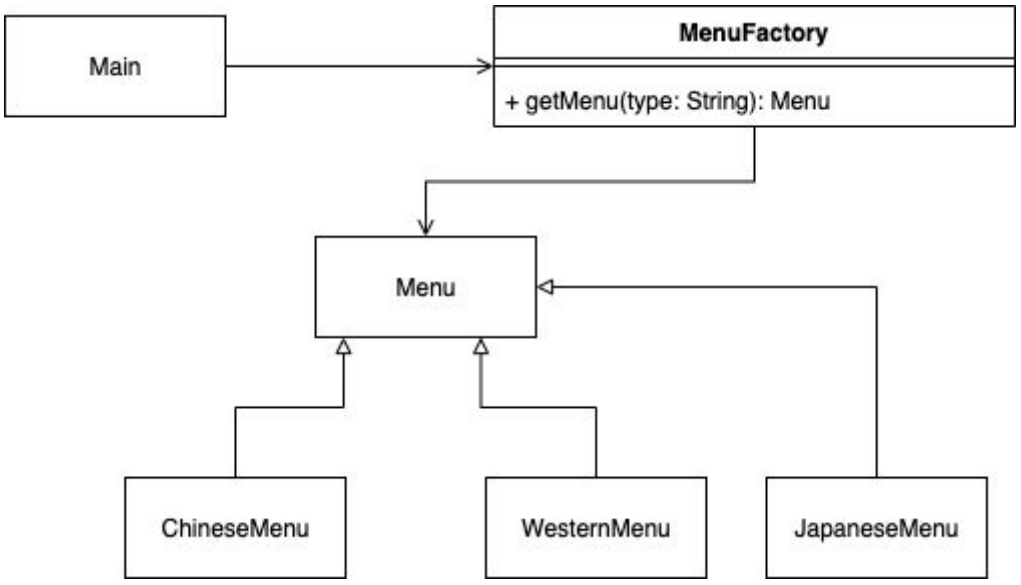


Figure 3: A factory is used to obtain the singleton instance of each menu by entering the type of that menu.

Memento Pattern

Memento objects are used to save the state of a menu, that includes the price, availableCount and mainDish. A user can choose to revert to the previous state of the order by calling **UndoCommand.execute()**, when they find there is an error to the new input.

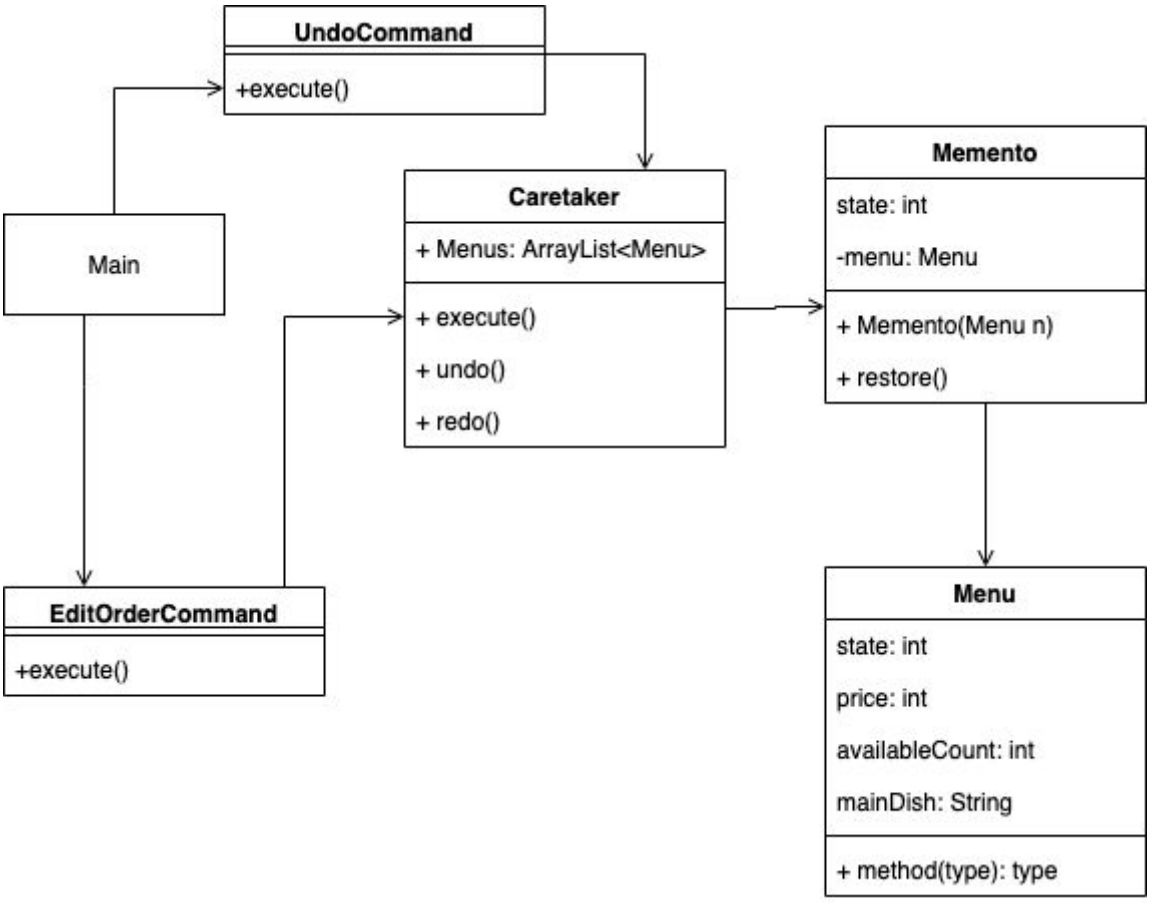


Figure 4: Simplified diagram for the memento pattern.

Singleton

To ensure the consistency of data while allowing the polymorphism to **Menu**, all subclasses of **Menu** are singleton, which means only one instance is used across the system for each subclass of **Menu**. This can be achieved by putting the instance inside **MenuFactory** and initializing them only once. The class is only meant to be accessed through a getter method **getMenu()**. If an instance exists, the instance will be returned. Otherwise a new instance will be created.

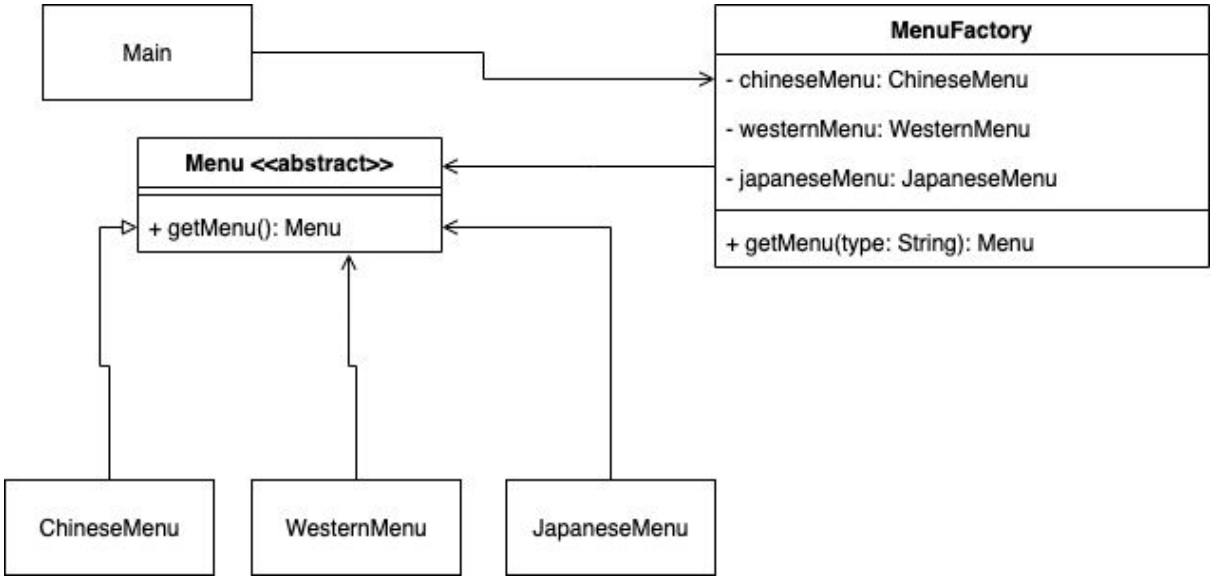


Figure 5: The use of the singleton pattern.

User Guide

The system supports the below function. Standard keyboard input will be used to navigate through the system, and major OS's including macOS, Windows and Linux are supported.

- e - Edit Menu

  A user can edit the main dish, price and available count of a specific menu, by choosing the corresponding menu.
- s - Show Menu

  All available menus are shown, given that they are not empty.
- p - Place Order

  A user can place an order by entering the staff number, office location and the options of the lunch set to place an order. When that menu is unavailable, the user will be prompted and they will be unable to place an order for that lunch set of that type of menu.
- c - Cancel Order

  A user can enter a staff number. If the staff has previously ordered, the **most recent** order of the staff will be cancelled. The stock count for the relevant menu will be increased by 1.
- l - List Order

  All outstanding orders will be listed.
- d - Complete order

  The first order in the queue will be marked as completed, and will be removed from the outstanding list of orders.
- q - Quit

  To leave the program.

Test Plan and Test Cases

As discussed, the main focus of this assignment is the use of different command patterns. Preliminary test cases are used to ensure that the program will work upon the entry of correct data. It is known that negative input, incorrect type of entry will make the program crash. However, test cases are run to ensure that the program behaves correctly upon entry of correct data.

Provided test case

The provided workflow from the assignment has been tested with a similarity of 91%.

Customized test cases

Command	Tests
(init)	output == “Please enter command: [e | s | p | c | l | n | d | q] e = Edit menu, s = Show menu, p = Place order, c = Cancel order, l = List orders, d = order is Done, q = Quit”
s	MenuFactory.getMenu(“c”).availableCount == 0 MenuFactory.getMenu(“w”).availableCount == 0 output == “ Chinese style Business Set Lunch main dish: null with rice, Chinese soup, Chinese tea price: 0 available count: 0 Western style Business Set Lunch main dish: null with rice/spaghetti/French fries price: 0

	available count: 0 “
P c	MenuFactory.getMenu(“c”).availableCount == 0 output == “Sold out!”
e c Steamed Fish 40 5	MenuFactory.getMenu(“c”).availableCount == 5
p c i 123 123	MenuFactory.getMenu(“c”).availableCount == 4 outstandingOrder.size() == 1
l	output == “Outstanding Orders C: 123, 123, Steamed Fish, Chinese Soup, Iced Tea “
d	output == “Complete Order C: 123, 123, Steamed Fish, Chinese Soup, Iced Tea Order marked as done “ outstandingOrder.size() == 0 completedOrder.size() == 1 MenuFactory.getMenu(“c”).availableCount == 4

The above has been tested with JUnit and was successful.

Appendix: Flowchart