Ryerson University

Department of Electrical & Computer Engineering
Faculty of Engineering & Architectural Science

Course Title:	OBJECT ORIENTTED ENGINEERING ANALYSIS AND DESIGN
Course Number:	COE 528
Semester/Year (e.g.F2016)	FALL 2017

Instructor:	Dr.Olivia Das

Assignment/Lab Number:	PROJECT REPORT
Assignment/Lab Title:	COE528 PROJECT:

Submission Date:	NOVEMBER 24th, 2016
Due Date:	NOVEMBER 24th, 2016

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Suri	Sandeep	500637194	2	Sandeep
Chandrakumar	Amsanaa	500660580	1	Amsanaa
Gupta	Archit	500644487	5	Archit

Reset Form

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Problem Description:

In this project, an application for an online store was created. For this application, Manager class, manages the product availability and keeps track of the items of each department there. It allows the customer to add and remove items, more specifically, store all the information about each item in the Main Store. Customers must register with the manager to checkout items. The items that are allowed to be checked out of the main store are categorized based on their type; such as Grocery, Electronics and Home Supplies. Application allows the customer to add items with the products name. The application also requests the customer to enter the details of each item when they added to the online store system. Each item should contain details of their product's title, year, price and delivery. When looking at the delivery, the manager will notify if there's fast shipping or not. If there's fast shipping, it'll take a shorter time and if there's no fast shipping, it'll take longer. The shipping time period will be determined by the manager. The customer is able to see how many available and unavailable items there are, and details of each individual purchased items. Moreover, the customer may add as many products as they like. When all the information of the items and its title are entered in the system, the product will be added to the checkout.

Functional Requirements:

Our developed simple application has two main significant uses. The first one is to add items to the checkout and the second is to display total number of items stored as well as the details of the items that are being checked out.

Use Case 1

1. **Name:** Add items
2. **Participating actor:** Customer
3. **Entry condition:**
    - **a.** The item to be checked out in the main store is one of the following item type.
        - i. Electronics
        - ii. Grocery
        - iii. Home Supplies
    - **b.** The fast shipping will be determined by the manager.
4. **Exit condition:**
    - **a.** The item is added to checkout. Or
    - **b.** The item is removed from checkout. Or
    - **c.** The product type is not supported.

5. **Flow of events:**
    - **a.** Customer logged in the application.
    - **b.** The application check if the product type is compatible with checkout.
    - **c.** If the product type is not compatible with main store, system notify the customer that the item is not supported.
    - **d.** If the product type is compatible with main store, the application will check if there is available item at checkout.
    - **e.** If there are available products in the main store, application will request the details of the item to be added.
    - **f.** Item is added.
    - **g.** Fast shipping will be determined.
6. **Special requirement:** Customer need to be logged in.

Use Case 2

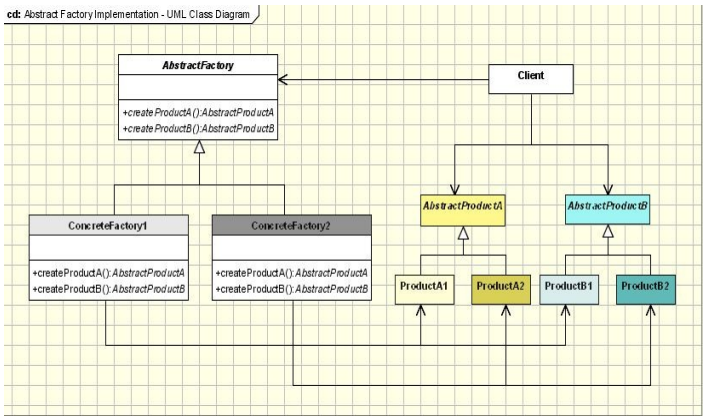
1. **Name:** Get Store details
2. **Participating actor:** Customer
3. **Entry condition:** The customer request for item details.
4. **Exit condition:** The items availabilities are displayed to the customer.
5. **Flow of events:**
    a. Customer logged in the application.
    b. Request main store information.
    c. If there is no item added, then system just display the online store details.
    d. If there is item added, Application displays total number of items stored.
7. **Special requirement:** Customer need to be logged in.

Design Patterns:

The abstract factory pattern is used to design our project. This pattern is usually used when a level of indirection that abstracts the creation of families of related or dependent objects without directly specifying their concrete classes. We chose this pattern because the pattern is best utilized when a system has to create multiple families of products or you want to provide a library of products without exposing the implementation details. In our case, the application is created for an online store system which support only a certain type of items. For future implementation, the online store would be able to expand and buy different types of items instead of just Grocery, Electronics and Home Supplies. In this case, the pattern is extremely future proof. If we are creating new type of products then adding new concrete types is needed,

all we have to do is modify the client code and make it to include new factory, which is far easier than instantiating a new type, which requires changing the code wherever a new object is created.

The pattern basically works as shown below, in the UML diagram:



Abstract Factory - declares an interface for operations that create abstract products.
Concrete Factory - implements operations to create concrete products.
Abstract Product - declares an interface for a type of product objects.
Product - defines a product to be created by the corresponding Concrete Factory.
Client - uses the interfaces declared by the Abstract Factory and Abstract Product classes.

The abstract factory defines the interface for how to create each member of the family object. Each family is created by having its own unique concrete factory. In our design, MainStore is the abstract factory which is an interface for adding our different family of item type (Grocery, Electronics and Home Supplies). These different type of departments are the concrete factory in our design. Each individual item created when added to the checkout will be referred to a product in above UML diagram. Those individual items are created from their own type concrete factory. For example, if the customer wants to add a Grocery type item, then that item will be created by the concrete Factory class Grocery. Same procedure for other type items too. In the future, if we want to make the online store compatible with Shoe type, only thing to modifies is to add a new concrete factory class called shoe and make the client adopt the class. Finally, the Manager and CheckOut classes in our design are the client that uses the interface declared by the MainStore to create new items from all three different type when they are added to the online store system.

Manager:

Manager class is an immutable class. Consists of a Login method(main), menu

method(manmenu). Also, consists of a addItem, and removeItem method. This class is responsible for confirm the login of the customer and allow the customer to add and remove items. Also display their product's title, year, price and delivery(whether it's fast shipping or not). The login details of the application is built in the main method.

CheckOut:

CheckOut is Mutable. This class has methods to find item size. This class is responsible for adding new items to the Arraylist and calculate the total number of items stored.

MainStore:

This is an interface that act as factory for Main Store. This constructor consists with getter and setter that will call the getters and setters methods in department type (Grocery, Electronics, Home Supplies) to create new item depending on their type.

Grocery, Electronics, Home Supplies:

These are three concrete factory class that create new items depending on their type as they added to the main store. It is the task of the MainStore class to create the products for each family. In our case it was used for new item. A concrete factory will specify its products from method for each of them. Even if the implementation might seem simple, using this idea will mean defining a new concrete factory for each product family, even if the classes are similar in most aspects.

Testing:

Test Case ID	Test case description	Whitebox/Blackbox
testAddItem()	Testing the addition of the object inside the newItem method. It is type void so nothing will be returned. It adds an object of MainStore into the newItem Array.	Blackbox
testRemoveItem()	Testing the removal of the object from the checkout method. This method is of type void and requires a string. The string is the item field of the Checkout and is removed when the item name is found in the array.	Blackbox
testToString()	Testing the toString method that is type String and it has a return. It returns a string representing the total number of items stored.	Blackbox
testTypeOfItem ()	Testing user input for creating new item object. Type of Item can ONLY be Grocery or Electronic. Integrated while loop will continue to loop until correct parameters are inputted. This test only loops once.	Whitebox
testConstructor()	Testing constructor of HomeSupplies object. Valid arguments are provided for the pass of the test. If invalid arguments were passed, the test would fail.	Blackbox

Conclusion:

The objective of this project was to create a software application based on the object-oriented techniques that were taught throughout the entire semester. It started off with the introductory step. After deciding the application specifications, we created the UML diagram. The UML use case diagram served as an outline for the application to be done and meet the mandatory requirements. It also allows the programmer to easily be able to modify the structure of the program. Although it might be a very shallow outline compared to what the coding part is meant to do, it served as a checklist to what needed to be completed. One of the challenges that were faced was when the classes and code part started becoming long. It was hard to track down all the variables that were used at the very beginning. At that point, the UML class diagram importance was realized. Having the Class Diagram was critical for the organizing part. The way the Class Diagram is and was put together made recognizing variable, their types and relationship possible.

The next challenge was keeping of the coding flow and the relationships between all the methods. The only way to overcome that was going back to the UML Sequence Diagram. The Sequence Diagram made it easier to track down the flow of information between the methods with related variables. Those variables were getting passed over from one method to another and it can be challenging to debug any issue down the road without having a UML Sequence Diagram. What made Sequence diagrams essentially one of the cores of the program, is that both of them were created alongside with the code implementation. In regard to design patterns, the designers had to use Abstract Factory Pattern at some point when a solution for a problem was needed. This pattern was used considering the future proofing the software. Since the software is developed for an online store, it is possible the store will be expanded and checkout for more different type of items. By using the abstract factory pattern, the software can be easily modified adopting new online store system.

In conclusion, the successful implementation of the project included almost all the material taught in the course. It was also beneficial to work in a team of three designers where ideas and views were shared and discussed to achieve the best results possible. It was definitely a beneficial task where the acquired knowledge, such as the use of modeling and design patterns, was applied. The project gave the developers a sense of what actual object-oriented programming is.