Assignment #3

UML – CSP 586

Name: Ritika Kamari

CWID: A20414073

Project Overview Statement:

Build an application for the GameSpeed retailer that will allow its customers to buy/trade-in products from the retailer either in-store or online.

Deliverables:

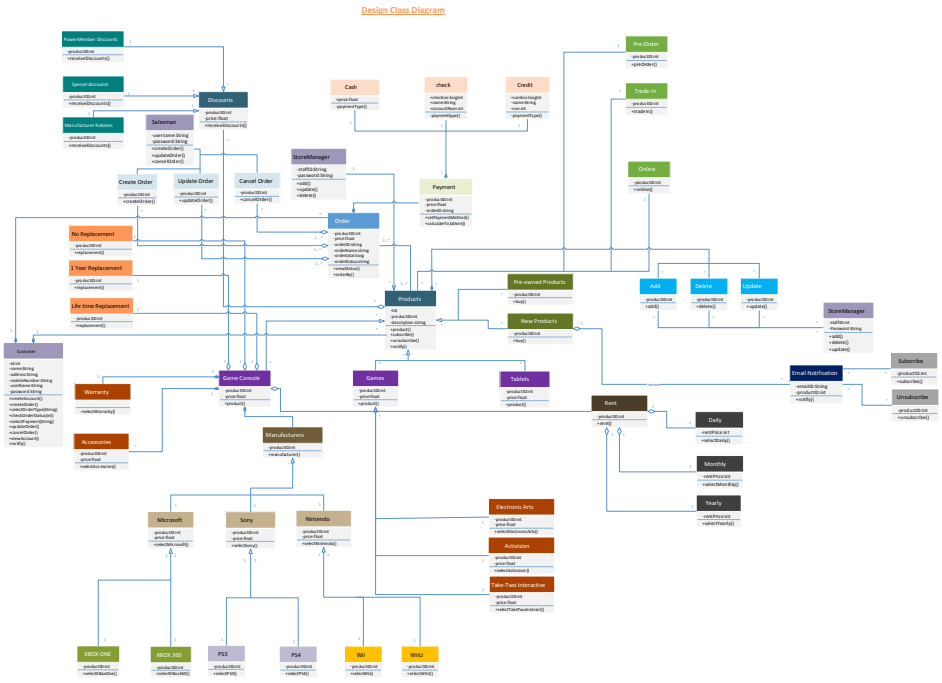
GameSpeed Retailer

1. Complete list of classes used in your design

Sl No.	Class name
1	Customer
2	Products a) Pre-Owned Products b) New Products
3	Order
4	Payment a) Cash b) Check c) Credit
5	Discounts a) Power Member discounts b) Special-discounts c) Manufacturer rebates
6	Salesman a) Create Order b) Update Order c) Delete Order
7	Store Manager d) Add Products e) Update Products f) Delete Products
8	Games a) Electronic Arts b) Activision c) Take-two Interactive
9	Game consoles
10	Tablets
11	Accessories
12	Warranty

13	Manufacturers a) Microsoft 1. XBOXONE 2. XBOX360 b) Sony 1. PS3 2. PS4
	c) Nintendo 1. Wii 2. WiiU
14	Rent a) Daily b) Monthly c) Yearly
15	PreOrder
16	Trade in
17	Online
18.	Email Notification a) Subscribe b) Unsubscribe

2. Complete UML Design Model/class diagram



3. List of the Design pattern(s) that you have used

The following are the design patterns which I have used.

a. Factory Method Design Pattern: Factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

The Factory Method design pattern solves problems like:

- How can an object be created so that subclasses can redefine which class to instantiate?
- How can a class defer instantiation to subclasses?

Creating an object directly within the class that requires (uses) the object is inflexible because it commits the class to a object and makes it impossible to change the instantiation independently from (without having to change) the class.

b. Abstract Factory Design Pattern: The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes. Client software creates a concrete implementation of the abstract factory and then uses the generic interface of the factory to create the concrete objects that are part of the theme. This pattern separates the details of implementation of a set of objects from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface.

The Abstract Factory design pattern solves problems like:

- How can an application be independent of how its objects are created?
- How can a class be independent of how the objects it requires are created?
- How can families of related or dependent objects be created?

Creating objects directly within the class that requires the objects is inflexible because it commits the class to particular objects and makes it impossible to change the instantiation later independently from (without having to change) the class. It stops the class from being reusable if other objects are required, and it makes the class hard to test because real objects can't be replaced with mock objects.

c. Observer Design Pattern: The observer pattern is a software design pattern in which an object, 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.

The Observer pattern addresses the following three problems:

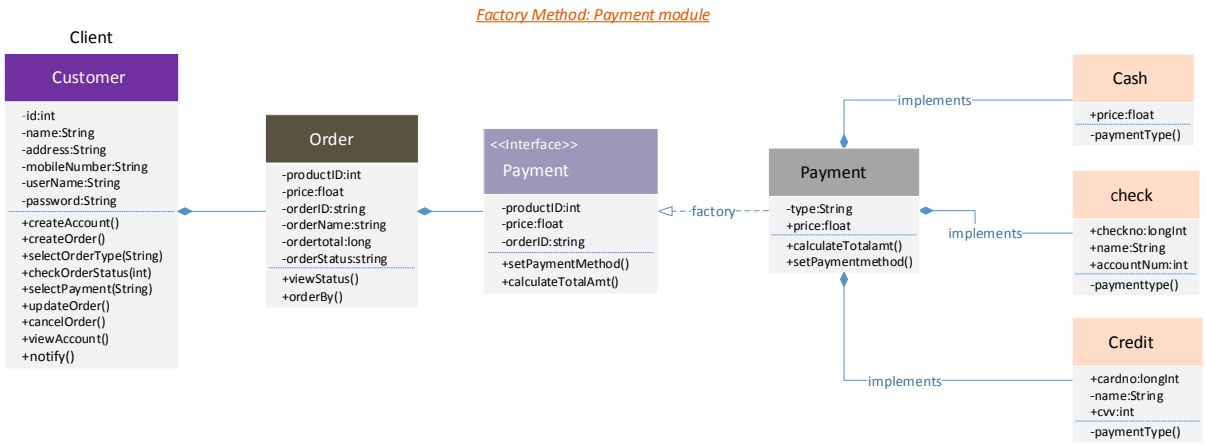
- A one-to-many dependency between objects should be defined without making the objects tightly coupled.
- It should be ensured that when one object changes state an open-ended number of dependent objects are updated automatically.
- It should be possible that one object can notify an open-ended number of other objects.

The observer pattern is implemented with the "subject" (which is being "observed") being part of the object whose state change is being observed, to be communicated to the observers upon occurrence.

4. Documentation how these design patterns are used in your design

The usage of Design Pattern are as follows:

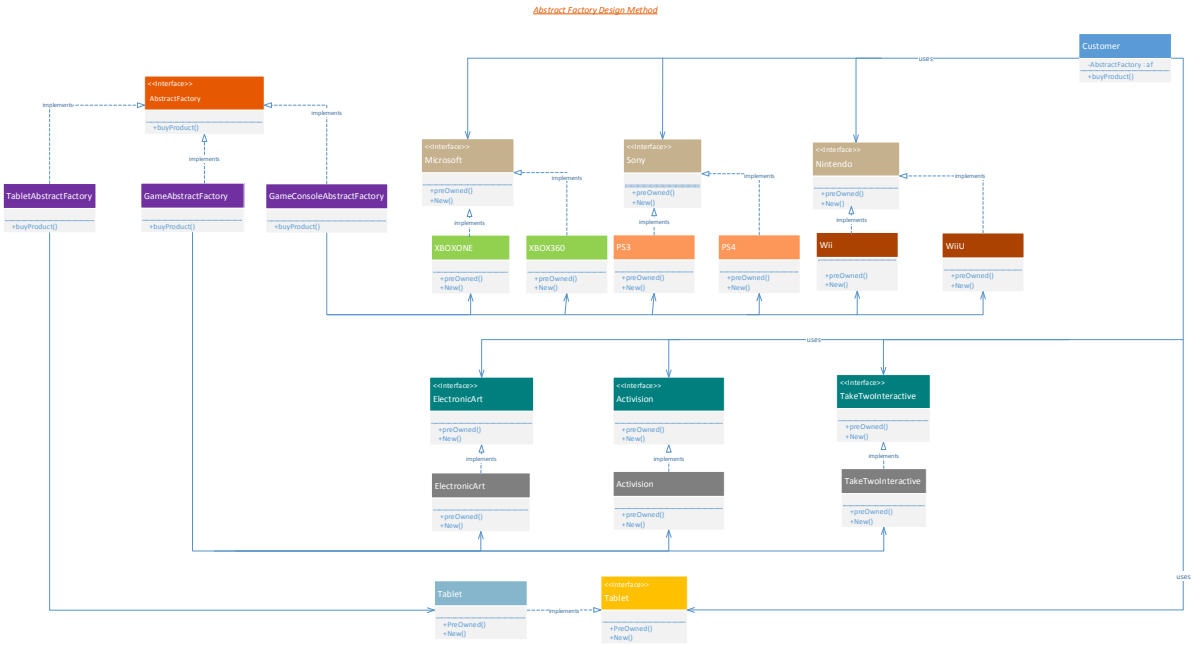
a. Factory Method Design Pattern:



On Payment module, I have implemented factory method. Here we have different methods to pay for the order for example payment by cash, check, and credit. Each payment type has its own creator class connected to them. All the methods are provided by the Payment method which is super class acting as an interface for them. Customer uses the desired method to pay for the order. The Order class is declared with the Payment method statically. This is the factory where the customer process payment.

Customer has various option to pay for the services he has availed based on the monthly billing. All Payment methods have creator and they are connected to each of them. Payment Method is the Interface which provide various interface of payment type. Customer use this interface to select appropriate payment method suited best to his situation. Hence, the Factory design pattern is used in this case.

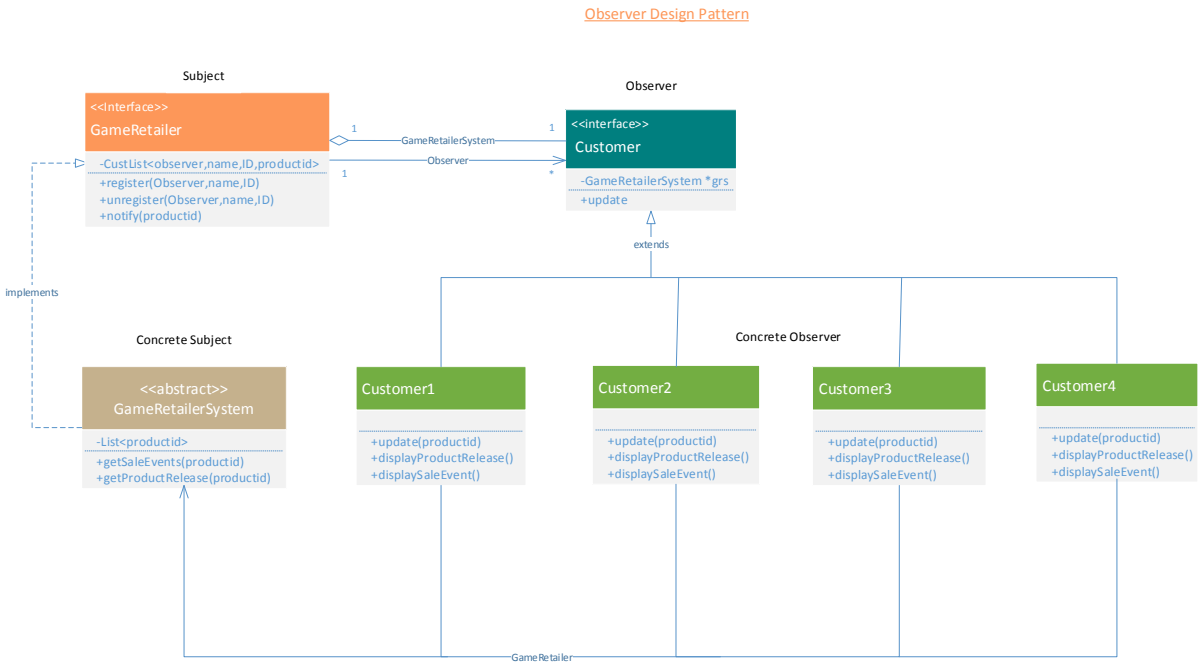
b. Abstract Factory Design Pattern:



On Product module, I have implemented abstract factory design pattern. The abstract factory pattern provides a way to encapsulate a group of individual factories (Table, Game Console and Games) that have a common method without specifying their concrete classes. This pattern separates the details of implementation of a set of objects from their general usage and relies on object composition, as object creation is implemented in methods exposed in the factory interface.

Hence, Abstract Factory Design Pattern is being used in this case.

c. Observer Design Pattern:



In Email subscription module observer pattern is used. Here the customer can subscribe / unsubscribe for email notifications when a new product is released and when there are any special sale events. In our case, Customer is the observer who use the interface subscription to use that offer. To subscribe / unsubscribe for email notification is being provided by the email notification interface.

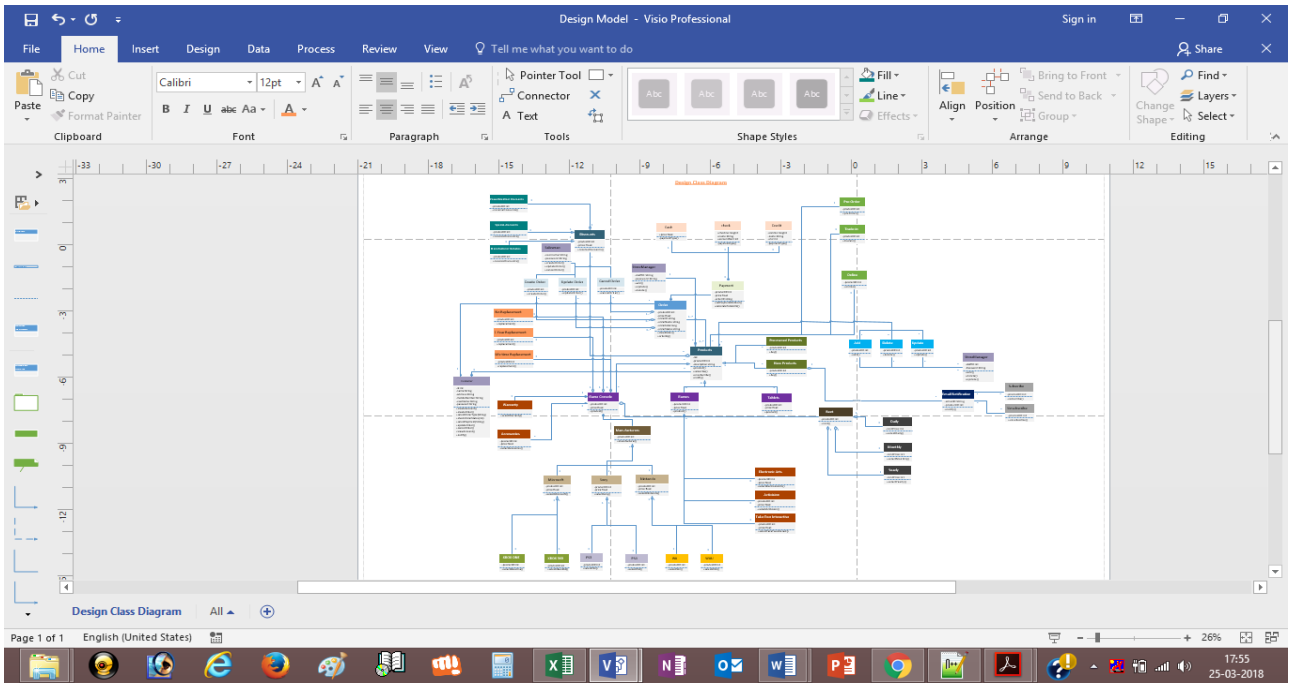
Here, this design pattern is used in the Customer Class. Customer can subscribe to email notifications for the game. This pattern generally represents a one to many relationships between observer and the many objects, i.e. the Customer and its subclasses. Also, the customer here can either choose to subscribe for updates from the customer class.

If there is any kind of change in the Customer Class irrespective of the reason, the classes will be notified. For example, if any new products will come, the customer will get email notification for that products and at the same time, customer can subscribe or unsubscribe the notification.

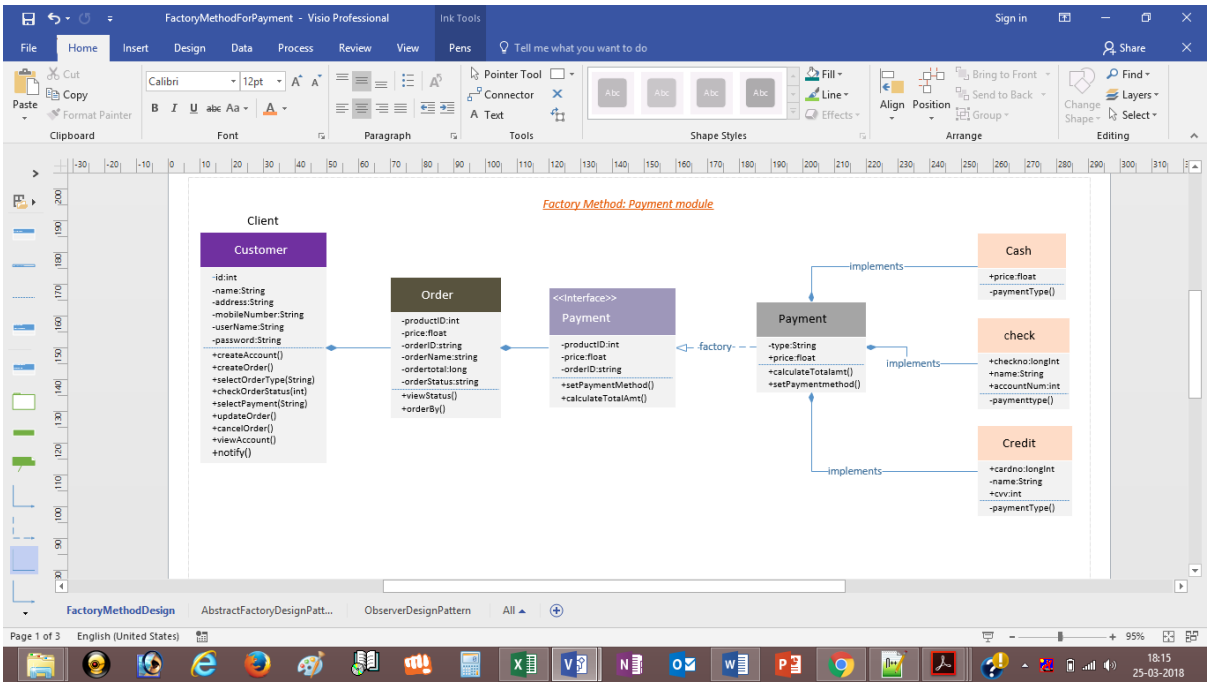
Hence, the Observer Design Pattern is used in this case.

5. <u>Capture design model class diagram(s) (using shft+PrtScr) and save it/them as image(s) in the PDF file that you are submitting as the solution for this homework.</u>

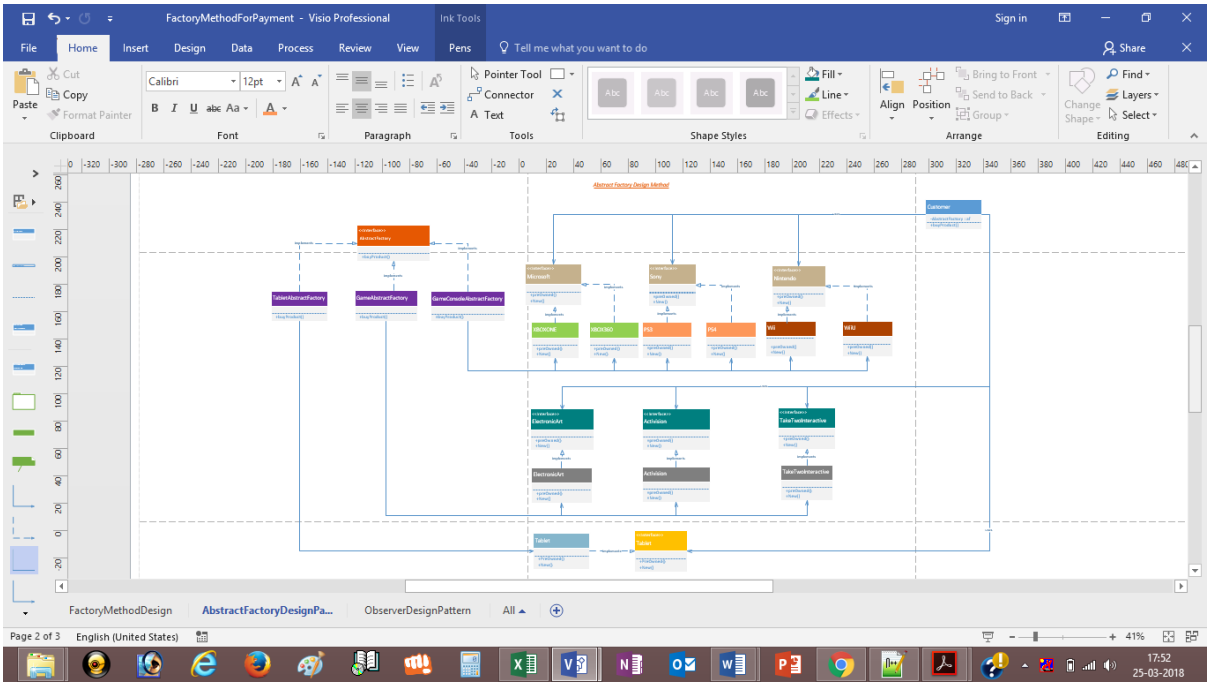
a. Screenshot of Design Class Diagram



b. Screenshot of Factory Method Design Pattern



c. Screenshot of Abstract Factory Design Pattern



d. Screenshot of Observer Pattern