ASSIGNMENT 1 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit 20: Advanced Programming		
Submission date	27/6/2021	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Nguyen Vu Thai	**Student ID**	GCH190530
Class	GCH0803	**Assessor name**	Doan Trung Tung
Student declaration I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.			
		Student's signature	Thai

Grading grid

P1	P2	M1	M2	D1	D2

❒ **Summative Feedback:**		❒ **Resubmission Feedback:**
Grade:	**Assessor Signature:**	**Date:**
Lecturer Signature:		

Table of Contents

I. Introduction: 6
II. OOP general concepts 6
1. Introduction about OOP 6
1.1 Objects 6
1.2 Classes 7
1.3 Instance 7
2. Method 7
3. Constructor 7
4. Features of object-oriented programming 8
4.1 Encapsulation. 8
4.2 Inheritance 8
4.3 Polymorphism 8
4.4 Abstraction 9
4.5. Association 9
III. OOP scenario 10
1. Scenario 10
2. Use case Diagram 11
3. Class Diagram 15
IV. Design Patterns 17
1. Factory Method 18

1.1 Scenario 18
1.2 Structure 18
1.3 Class diagram 20
2. Abstract Factory 21
2.1 Scenario: 23
2.2 Diagram 24
3. Facade Design Pattern 25
3.1 Scenario 25
3.1 Diagram 27
4. Behavior Pattern 28
4.1 Scenario: 28
4.2 Diagram 30
V. Design Pattern vs OOP 32
1. Discuss / analyze the relationship between design pattern & OOP 32
1.1 Facade Method 32
1.2 Observer Pattern 32
1.3 Factory Pattern 33
VI. Conclusion 33
References 34

1. Use case diagram 11
2. Class diagram 16
3. Solution of Factory Method 19
4. Example of Factory Method 20
5. Solution of Abstract Factory 22
6. Example of abtract factory 24
7. Solution of Facade 26
8. Example of Facade 27
9. Solution of Observer 28
10. Example of observer ..... **Error! Bookmark not defined.**

I. Introduction:

In this report, we will learn about OOP through the introduction to OOP. Next, we will explore patterns and analyze the relationship between design patterns and OOP.

II. OOP general concepts

1. Introduction about OOP

As you know, procedural programming is a very popular programming style that almost any programmer has learned. This type of procedural programming has a lot of disadvantages such as poor security. This is an older style of programming. So today I will introduce you to a new programming style that is object-oriented programming. Some programmers wanted to build a programming method that faithfully describes the system as close to reality as possible, so they created object-oriented programming. Object-oriented programming is a programming method of building programs in which objects are the foundation. Another definition of object-oriented programming is class and object-based programming. (Akin, 2001)

1.1 Objects

According by (Akin, 2001) object usually consists of: attributes is information, properties of objects and method are the actions of the object. When creating an object of a class, that object will have the attributes and methods of that class and we can also pass the objects to call attributes and methods.

1.2 Classes

A class is a data type consisting of predefined properties and methods. This is the abstraction of the object. Unlike a normal data type, a class is a (abstract) unit consisting of a combination of methods and properties. More roughly, objects with similar properties are grouped into a class of objects. (Akin, 2001)

1.3 Instance

An instance is a specific instance of a class. All instances of a class have the same properties as described in the class. The term object is often confused with instance. But object is a real entity, and instance is a virtual copy. Many people confuse object with instance, but these are two different concepts. (Pecinovsky, 2013)

2. Method

Methods are the behaviors or ways of working of an object. A method that collects a sequence of statements to perform a certain task. The main function is also considered a method. Classes can also own methods . Methods are also often divided into two types: functions with parameters and functions without parameters. (Pecinovsky, 2013)

3. Constructor

A constructor is a special function of a class. It will be automatically called when a class with an object is instantiated. There are usually two types of constructors: those with arguments and without parameters. The constructor without parameters is usually called when the user initializes without passing parameters to the initializer property, this function will automatically initialize the object with the default properties. The parameterized constructor is called when the user initializes an object with full parameters. (Pecinovsky, 2013)

4. Features of object-oriented programming

4.1 Encapsulation.

Methods related to each other are encapsulated into classes for easy management and use. This property allows information concealment, control read and write data by the user. Other objects cannot directly affect the data inside and change the state of the object but must go through public methods provided by that object. (Akin, 2001)

4.2 Inheritance

This property is used a lot. Inheritance in programming is how a class can inherit properties and methods from another class and use them as their own. Inheritance allows building a new class (Child class), inheriting and reusing properties and methods based on the old class (Parent class) that existed before. This feature allows us to share common information for reuse and at the same time makes it easy to upgrade and easy to maintain. (Akin, 2001)

4.3 Polymorphism

Objects in different classes will interpret the same message in different ways. There are two types of polymorphism, static and dynamic, which are static and dynamic polymorphism. In static polymorphism, the response to a function is defined at compile time. Whereas with dynamic polymorphism, it is decided at runtime. Condition: All classes must inherit from a parent class. Polymorphic methods must be override in child classes. (Akin, 2001)

4.4 Abstraction

Abstraction eliminates the unnecessary complexity of the object and focuses only on what is essential and important. Abstraction is a process of hiding the implementation details and exposing the feature only to the user. Abstraction allows you to eliminate the complexity of an object by exposing only the properties and methods needed by the object in programming.

Abstraction and encapsulation are two related characteristics in object-oriented programming. Abstraction allows making related information visible, and encapsulation gives the programmer the ability to implement inherited abstraction. Abstract classes and pure virtual methods: Pure virtual method is a virtual method and has no internal definition. An abstract class is a class that contains pure virtual methods. (Akin, 2001)

4.5. Association

Association is a relationship between two distinct classes based on their objects that are established. The link can be one-to-one, one-to-many, many-one, many-to-many. Composition and Aggregation are two forms of association. (Akin, 2001)

Aggregation is a form of association that represents a has - A relationship ie one has that and is a one-way relationship. These two entities can exist separately, which means that ending one entity will not affect the other. (Akin, 2001)

Composition is a relationship where two entities are highly dependent on each other. Composition also represents a part-of relationship, that is, a part. In this relationship, both entities are dependent on each other. When there is a composition relationship between two entities, the composed object cannot exist without another entity. (Akin, 2001)

III. OOP scenario

1. Scenario

Today, due to the Covid-19 disease, shopping for groceries online has become more necessary than ever, so in Asm1, my team decided to choose an online purchasing system for this OOP scenario. The system is interacted by 3 main users. Customers, guests and admins of the system. For each user the functions used are different. User must log in to the system to perform to use the system's functions. If user forgot password or username, they can recover forgot password as well as username. Except for the product view function, Guest can view the product but not display the product details, so Guest has an additional registration function to be able to view product details. Customers can view product details and add to cart the products they want to buy as well as delete products from their shopping cart. Admin can manage the products on the system.

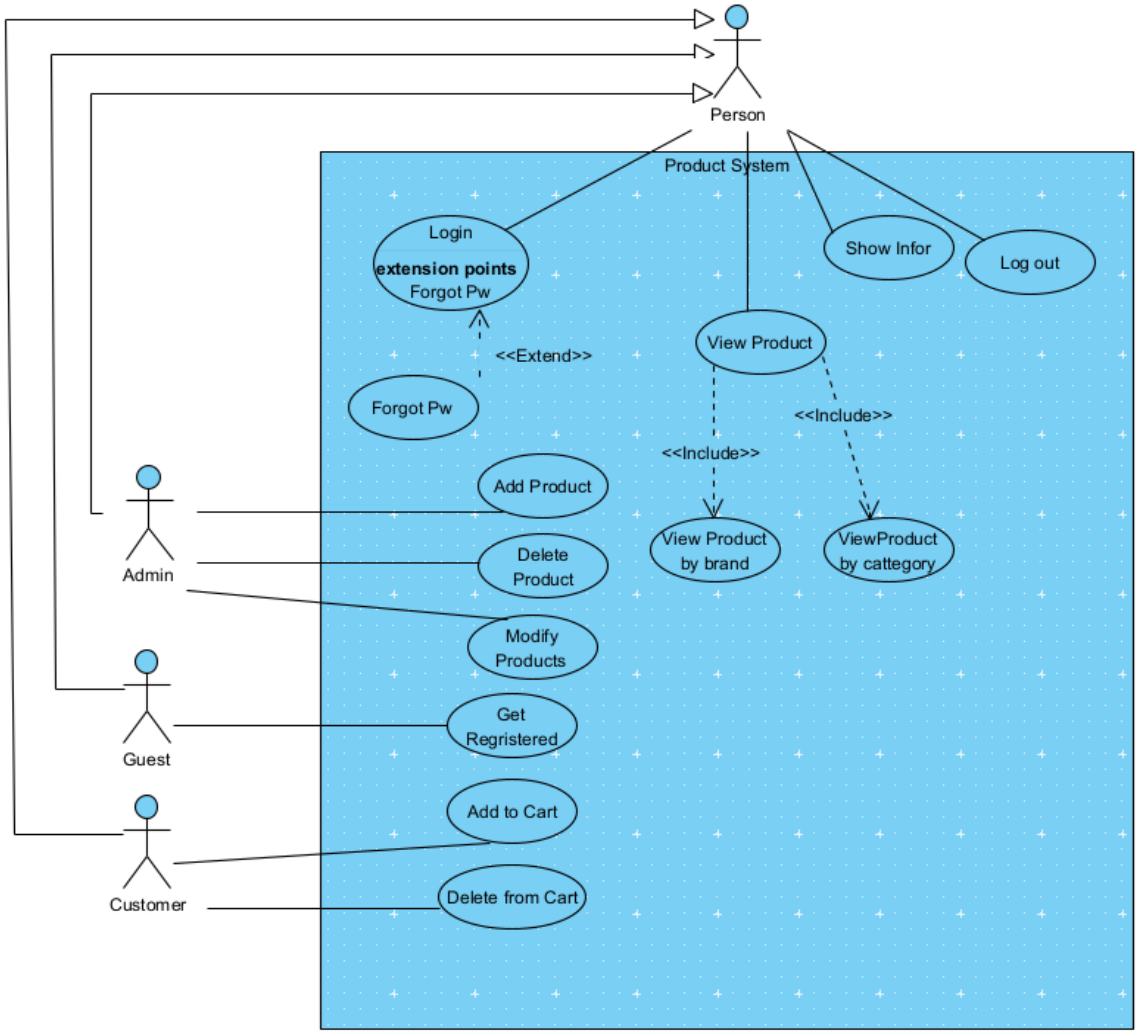
Functions included in the program for each User:

Customer: View personal Information by himself, view of product (include view by category or view by brand). Customer can Add product they want to shopping cart and delete product they don’t want from shopping cart.

Admin: Admin can manage the products on Store.

Guest: Can register to become a customer, can view products (limited mode).

2. Use case Diagram



1. Use case diagram

Explanation:

	Name of Use Case:	Add Product		
	Created By:	Group 1	**Last Updated by:**	6/17/2021
	Date Created:	6/12/2021	**Last Revision Date:**	6/24/2021
	Description:	Allow Admin to add new Products into the system		
	Actors:	Admin		
	Preconditions:	Admin needs to login into system first		
	Postconditions:	The new product is added to system and it assigns a number to each product		
	Flow:	1. Admin select Add Product button from the menu option 2. Admin enter correctly information of product 3. The record of the products is created and assign number to each product has been done.		
	Alternative Flows:	2.1 Admin entered incorrect product information format		
	Exceptions:	No exception		
	Requirements:	No		

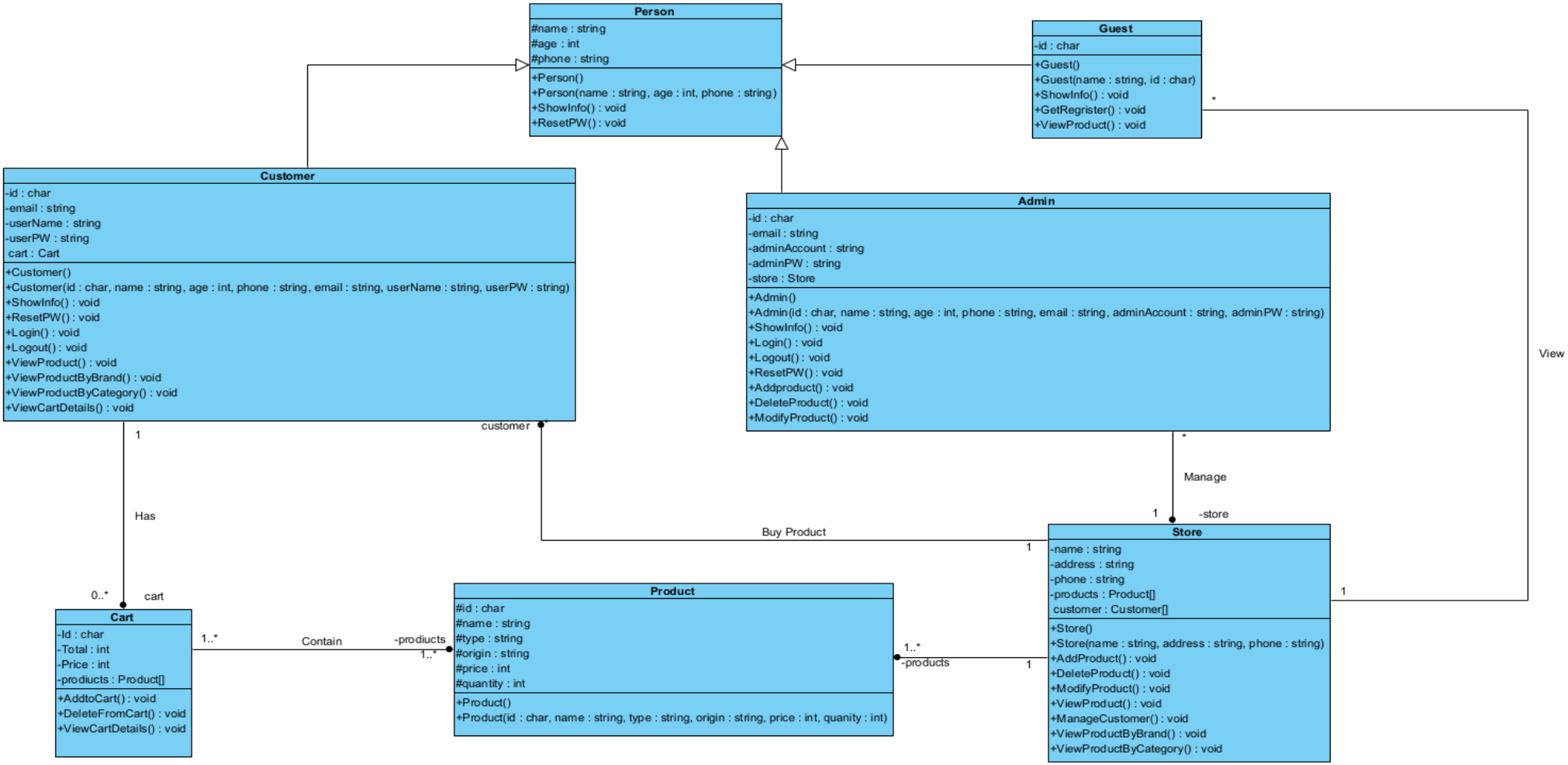
	Name of Use Case:	View Product		
	Created By:	Group 1	**Last Updated By:**	6/17/2021
	Date Created:	6/12/2021	**Last Revision Date:**	6/24/2021

	Description:	Person use this function to view information of product
	Actors:	Admin, Guest, Customer
	Preconditions:	None
	Postconditions:	None
	Flow:	1. Person selects any product he/his wants to view. 2. Selected information will display information on the screen and the use case end.
	Alternative Flows:	None
	Exceptions:	1.1 Person can select product by category 1.2 Person can select product by brand

Requirements:	None

	Name of Use Case:	Login		
	Created By:	Group 1	**Last Updated By:**	6/17/2021
	Date Created:	6/12/2021	**Last Revision Date:**	6/24/2021
	Description:	Customer must login into system to Buy Product online Admin also must login into system to manage Product		
	Actors:	Admin, Customer		
	Preconditions:	Customer and Admin is required to register an account before logging in and use that account to log in.		
	Postconditions:	No post condition		
	Flow:	1. Customer and admin select “Login” button 2. Customer and admin type his/her username and password on the login form 3. The system validates the account from the database 4. The system displays the Main Form and the use case ends.		
	Alternative Flows:	2.1 Customer and admin input incorrect username or password		
	Exceptions:	Forgot Password		
	Requirements:	None		

3. Class Diagram



2. Class diagram

Explanation:

First in the class diagram, I already divided into classes. Each class has attribute and methods. I user the inheritance in the Person class with classes: Admin, Customer, Guest. The secondly, I use encapsulation in all the classes in the program for the purpose of easy management and at the same time concealing information so that people can limit which parties can interfere or revision, edit the information of any object. In Person class and Product class, the symbol "#" represents the access directive of that attribute as protected. Only child that inherit from this Person class can access it. All classes contain 2 constructors to declare objects. Consists of 1 constructor without parameters and 1 constructor without parameters.

In Customer, Admin and Guest class, I use overriding to override for ResetPW and ShowInfo method because the way to recover the password of each object will be different, the Admin can reset the password by himself directly through the system, and the Customer is required to reset the password via email, as well as the information of each displayed object class will also be different.

Next, I will analyze the relationships between the attributes in the class diagram. Firstly, I have a relationship between Customer and Cart which association with multiplicity is 1-0…* it's mean that a customer can have multiple carts, and when he doesn't want to buy anything, he doesn't have any of the carts. Next, we can see the relationship between Customer. Guest and Admin with Store are association with 1-* multiplicity because so many Customer, Admin and Guest can buy products, view products and manage products in store. Following the relationship between Store and Product is association with 1-1…* multiplicity because store has more than 1 product. Next, I have a relationship between Product and Cart which 1…*-1…* multiplicity association it's mean that a cart can contain many different products and a product can be contained by many different customer carts.

IV. Design Patterns

The design patterns of object-oriented programming are like the built-in house and town patterns. Each template is used for repetitive problems, which can be reused. Simply put, design patterns are total solutions used in common problems when designing optimized software. Each design pattern has a different use and is reusable. (KevinZhang, 1994)

Benefits of using design patterns:

Make the product easier to maintain, change and flexible. When there is a change in requirements, the software will bloat. Design pattern will help the performance to be more optimized. Design patterns help us develop software faster. When faced with problems that have already been solved, design patterns help us to solve them in an easier way. Help others understand your code quickly. (KevinZhang, 1994).

Classify:

Creational Patterns: Abstract Factory, Builder, Factory, Prototype, Singleton. This group will help you a lot in object initialization, which you will hardly notice (it will not use the new keyword as usual). (KevinZhang, 1994)

Structural Patterns: Adapter, Bridge, Composite, Decorator, Facade, Fly weight, Proxy. This group will help us establish and define relationships between objects. (KevinZhang, 1994)

Behavior Patterns: Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template method, Visitor. This group will help us establish and define relationships between objects (KevinZhang, 1994)

Creational Patterns

According by (Sourcemaking, 2021): CDPs are design patterns that help support object creation mechanisms so that object instantiation works in a way that is appropriate for a given situation. They solve this problem by controlling the creation of objects and reduce the complexity and instability of basic object creation method. Some of the Creational patterns are: Abstract Factory, Builder, Factory Method, Object Pool, Prototype and Singleton.

1. Factory Method

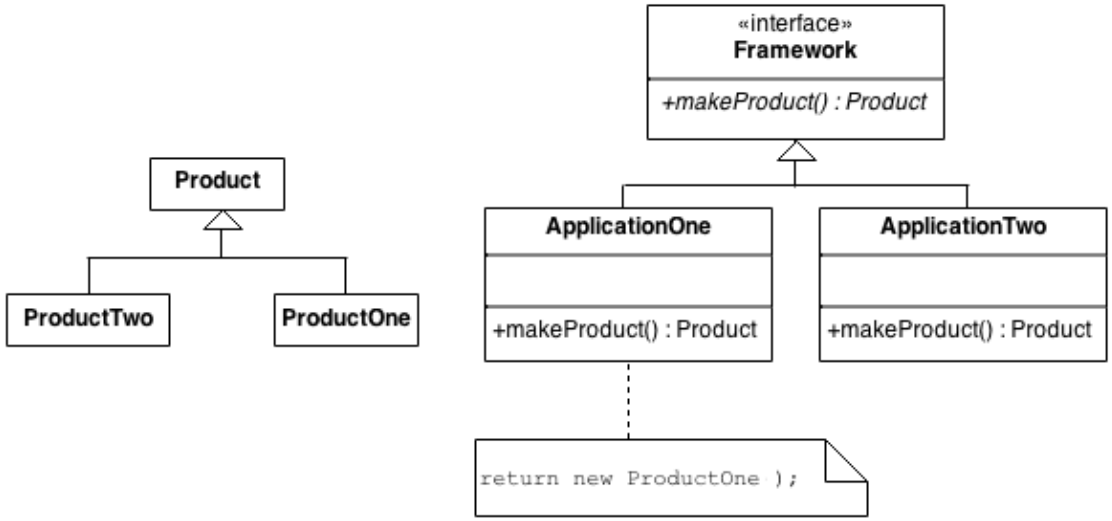
The Factory Method design pattern is used to define an interface or abstract class to create an object but delegates to the concreate factories to determine which classes are instantiated.

1.1 Scenario

Burger order system for 2 Burger stores, 2 stores have same types of Burger. Users can order at the store what store they like. Customers can check the making of process burger until delivery to the customer. For this scenario I choose the Factory Method to solve this because Factory Method pattern suggest that I can replace direct construction calls with calls to a special factory method. More specific, I will explain after showing Class-diagram.

1.2 Structure

According by (Sourcemaking, 2021):

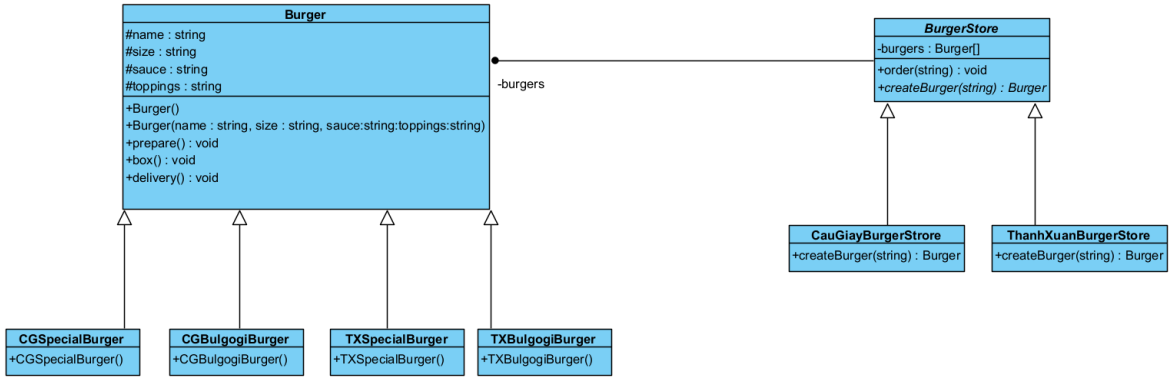


3. Solution of Factory Method

Explain:

- Product is a common class that all child product must implement.
- ProductTwo and ProductOne are concreate child product that implements from the Product interface
- Framework is interface that declare a factory method with a return data type of Product that allows returning new products.
- ApplicationOne and Application Two are classes inheriting from Framework is responsible for overriding factory method and returning the corresponding product type.

1.3 Class diagram



4. Example of Factory Method

Explain:

According to the design above, can be seen:

CGSpecical Burger and CGBulgogi Burger are Product One. They are 2 burger created from the abstract method create Burger() of CauGiay Burger Store. TXSpecial Burger and TXBulgogi Burger are also Product Two Classes and was created from the abstract method create Burger () of ThanhXuan BurgerS tore. Method createBurger() is the method to initialize objects, with flexible use of the method in different stores it helps to create objects corresponding to each of those stores.

Abstract Burger Store is the Framework that contains an abstract method and a method to call Abstract

CauGiay Burger Store is Appllication One and ThanhXuan Burger Store is Appllication Two. These 2 classes will inherit the abstract class method of the Abstract Burger Store. Two Classes inherit the abstract create Burger (). But each class will implement a

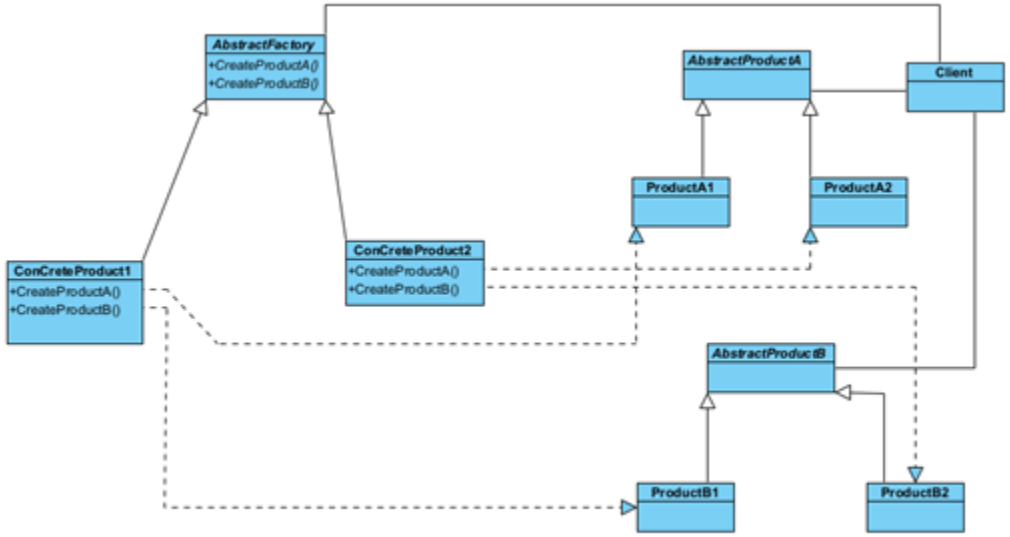
different burger creation method. The result of the abstract method will generate a corresponding object. For example. CauGiay Burger Store will create CG Special Burger and ThanhXuan Burger Store will create TX Special Burger.

This application will use Burger Store to making burgers depending on the location of store (Thanh Xuan, CauGiay) This application only uses Abstract Burger store and doesn't care how burger are created. This application simply calls burgerstore.order() to make corresponding burger.

2. Abstract Factory

When your system has many objects with similar properties and behavior, this pattern is used. The Abstract factory pattern is like a factory that produces these similar objects for you. We should use abstract factory when creating similar objects, provide complete method to create objects, create special objects from superclass, easily extend system from old system. Abstract Factory can be visualized as a large factory within which there are small factories that produce related products. (KevinZhang, 1994)

Solution:



5. Solution of Abstract Factory (KevinZhang, 1994)

Explain:

- Abstract Factory: Declare an abstract or interface class for product creation activities.
- Concrete Factory (ConcreateProduct1, ConcreateProduct2): Inherited from the Abstract Factory class, implements the creation of concrete products but is still subject to design.
- Abstract Product: Create an interface or abstract class for a product type.
- Concrete Product (ProductA1, ProductB1, …): Identify the product created by the respective Concrete Factory and inherited from the abstract product class to conform to the designs.

- Client: Objects using Abstract Factory and Abstract Product.

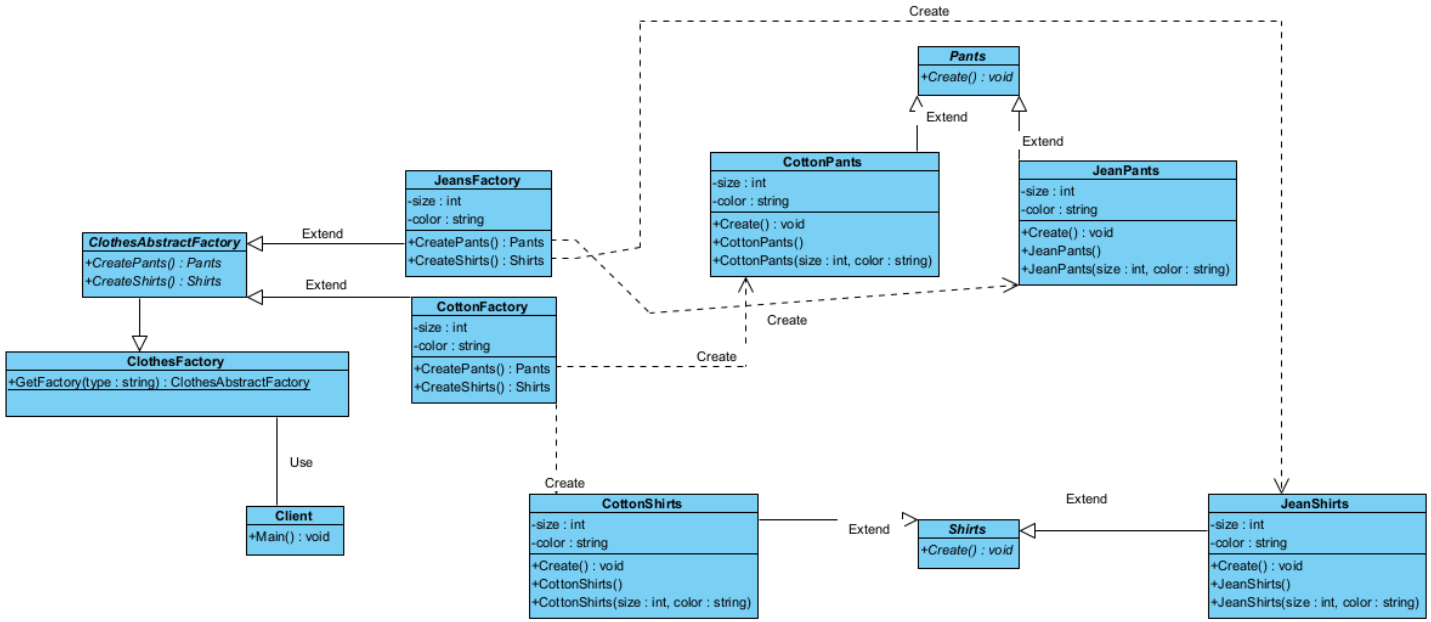
Benefit of abstract factory:

First, this pattern creates independent concrete classes, it helps you control the object classes of an application. It encapsulates the creation of objects and isolates the client from the implementation class. The client will manipulate through the abstract class, the product class name is isolated in the implementation. Second, it avoids excessive use of conditional logic (if-else) and is easy to manage. Third, can be easily extended to accommodate more factories and other sub-classes. Easily build an encapsulate system: usable with many groups of objects (factories) and create many different products. (KevinZhang, 1994)

2.1 Scenario:

A garment factory specializing in the production of all kinds of shirts: Cotton, Jean. With favorable business situation, the garment factory decided to produce more types of pants. With the experience from the production of shirts, the company still uses the same pants as the shirt production materials: Cotton and Jeans. However, the clothing production process is different, so the garment factory decided to separate 1 factory into 2 factories: 1 jeans factory (JeansFactory) and 1 cotton fabric factory (CottonFactory) but both can produce pants and tops (ClothesAbstractFactory). When customers want to buy something, they just need to go to the store (client) to buy it (ClothesFactory). Then, for each goods and material, it will be transferred to the workshop to produce pants (Pants) and shirts (shirts).

2.2 Diagram



6. Example of abtract factory

Explain:

Jean Factory and Cotton Factory are Concrete Factory. Clothes Abstract Factory is Abstract Factory. Cotton Pants, Cotton Shirts, Jean Pants, Jean Shirts are Concreate Product. Pants, Shirts is Abstract Product. Clothes Factory contains the function Get Factory (type: String), this function is called when the user(client) orders cotton or jeans pants or shirt, this class will hide the implementation for the user (Client).Client is the user interface. The creation of a pair of pants or a shirt of what material will be declared in the main function.

3. Facade Design Pattern

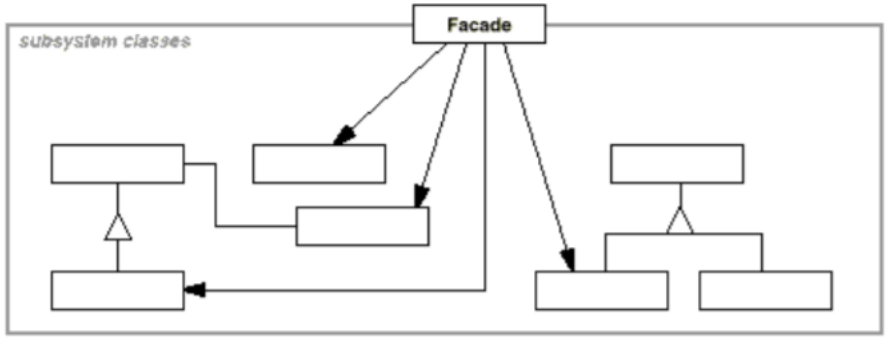
What is Facade pattern?

In software engineering, the facade pattern provides a unified interface to a set of interfaces in a subsystem. Facade defines a higher level's interface that makes the subsystem easier to use. (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 2001)

3.1 Scenario

We have set up a simple system . This system has the function of generating information for the characters in the game and managing them. Information includes: name, location, ... Instead of having to access different screens to enter information, we have come up with a method for customers to access through the main screen to Fill in all required information. This action of mine will be explained and shown through the class diagram.

Structure:

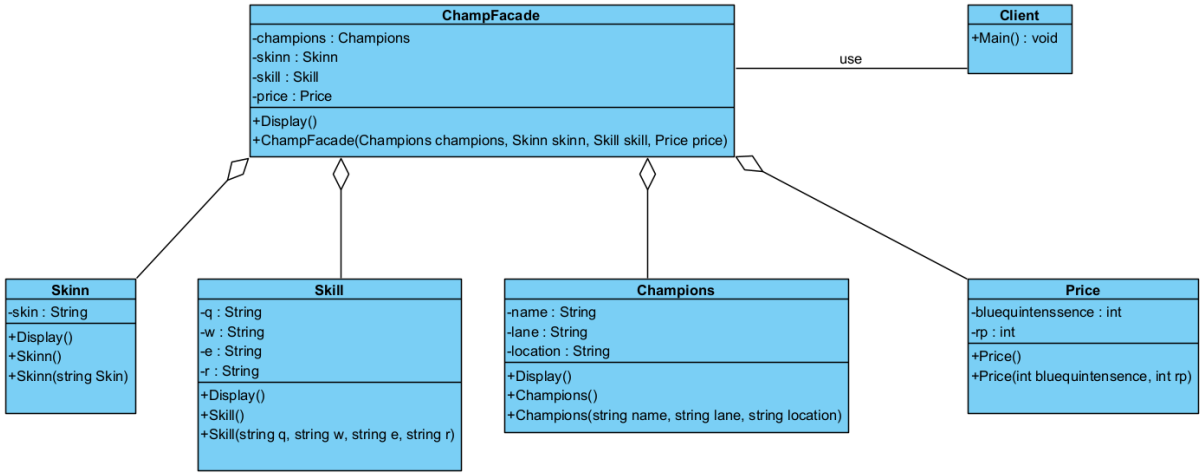


7. Solution of Facade

Explain:

The Facade Pattern is used to make it easier for client-side applications to communicate with the system. Instead of having to work with multiple subsystems, the Facade Pattern helps the client application only have to communicate with a single system.

3.1 Diagram



8. Example of Facade

Explain:

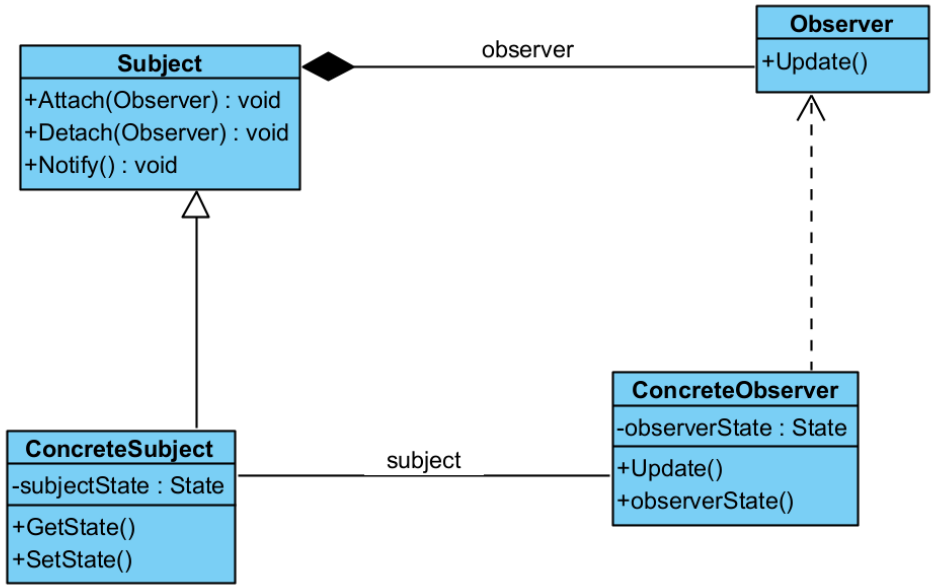
We want to manage the above information by building a ChampFacade class to take advantage of the above classes.class champFacade is the class facade. The subclasses include: Skinn, Skill, Price, Champions. Client is where the user uses the system.Class ChampFacade has-a 4 subclasses : Skinn , Skill , Price , Champions .ChampFacade is used to make it easier for subclasses to communicate with the client. Instead of having to work with many subclasses, the champ Pattern helps the client application to only have to communicate with a single system.

4. Behavior Pattern

What behavior pattern?

According to (Gamma, Helm, Johnson and Vlissides, n.d., 1994) In software engineering, behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.

4.1 Scenario:



9. Solution of Observer

According to (Freeman et al., 2011) :

Subject : contains a list of observers, providing methods to add and remove observers.

Observer : defines an update() method for objects that will be notified by the subject until there is a change in state.

ConcreteSubject : implements Subject's methods, stores the state of a list of ConcreateObservers, and sends notifications to its observers when there is a change in state.

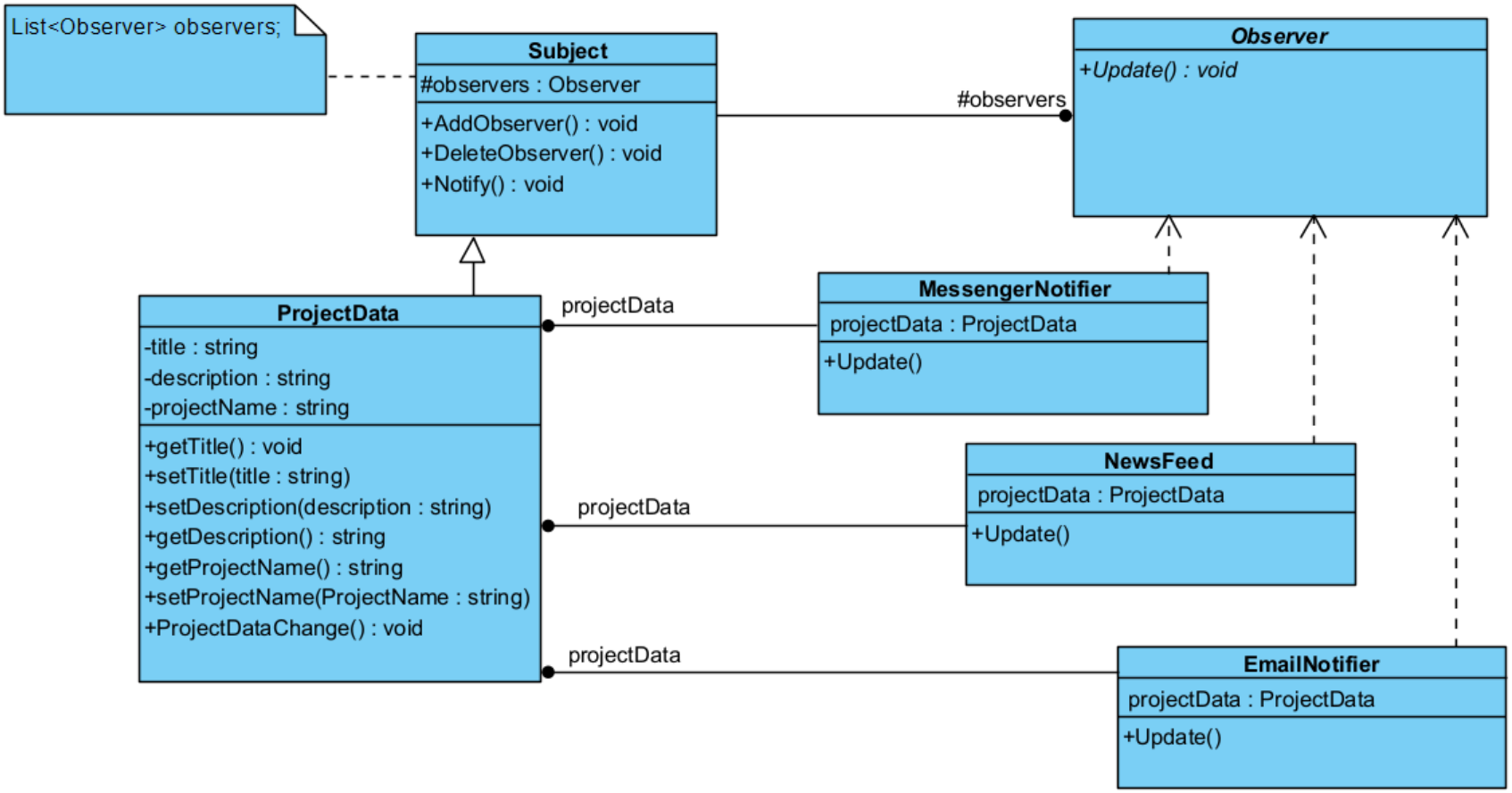
ConcreteObserver : implements Observer's methods, stores the state of the subject, performs updates to keep the state consistent with the subject sending notifications..

The interaction between the subject and the observers is as follows: whenever the subject has a state change, it traverses its list of observers and calls a method that updates the state on each observer, possibly passing itself to the method. mode so that observers can retrieve its state and process it.

Example:

Mr. A is a technology director, he has a team of employees to help him complete the projects he has received. In the past, every time he invested in a project, he had to text each person to a phone number or email to inform his staff about the project, or each time to change some components in the project. the project he will have to do the same thing , that is very time consuming and inefficient . so this system will apply observer pattern to design a system that can help him to send all notifications to receiver in any form he wants.

4.2 Diagram


List<Observer> observers;
Subject
#observers : Observer
+AddObserver() : void
+DeleteObserver() : void
+Notify() : void
#observers
Observer
+Update() : void
ProjectData
-title : string
-description : string
-projectName : string
+getTitle() : void
+setTitle(title : string)
+setDescription(description : string)
+getDescription() : string
+getProjectName() : string
+setProjectName(ProjectName : string)
+ProjectDataChange() : void
projectData
MessengerNotifier
projectData : ProjectData
+Update()
projectData
NewsFeed
projectData : ProjectData
+Update()
projectData
EmailNotifier
projectData : ProjectData
+Update()

Class<Subject>: is a class created to contain Observers and then stores Observer objects into a list. The class has functions like AddObserver (add observer), DeleteObserver (delete observer), Notify (call to Update method of Observers).

Class<ProjectData> : is a class to change the information of the message content. A message has properties like Title (Project's subject), ProjectName (Project's name), description (display details of the message). Besides, the SetTitle, SetProjectName, SetDescription method is the method to initialize the Title , ProjectName, Description information. The GetTitle, GetProjectName,GetDescription methods are methods that return Title , ProjectName , Description information.

Abstract Class <Observer> : is a class that initializes Observers. The abstract method Notify() is overridden by the subclasses MessengerNotifier, NewsFeed, EmailNotifier.

Class<MessengerNotifier><NewsFeed><EmailNotifier> are ConcreteOberver classes, when ProjectData changes, it will call the overridden update() function of Observer class and display a message on the screen.

Reasons for choosing the Observer algorithm:

It can solve the problem of sending notSifications to everyone via selected methods when using one to many relationships between objects. When an object changes, the related objects are notified of the change.Besides, when we change an object, other related objects also make the change.

V. Design Pattern vs OOP

1. Discuss / analyze the relationship between design pattern & OOP

When we use OOP but not Design pattern we will have some problems. Firstly, in terms of time, we will run the program longer, it takes a lot of time to think and find a solution, but the efficiency is not high. But when using the design pattern, we will reduce the time and effort to think of solutions to the problems that have solutions and directions. We only need to apply the Design patterns compatible with the program, saving us time and effort. Besides, it makes the program run more smoothly, easy to manage the operation process, easy to upgrade and maintain. Besides, the design pattern will solve some problems that the OOP method cannot solve or will be difficult to solve.

1.1 Facade Method

As the theory has been stated above, if a problem is handled in a normal way, if not coded carefully or properly, it will make the code confusing, difficult to control and difficult to control. maintenance. But when applying the Design pattern Facade it will make the system simpler to use.

1.2 Observer Pattern

If a problem were handled in a normal way, it would be difficult to get a 1-n relationship between objects. If not used proficiently, it will make the code confusing and difficult to control, besides the maintenance and upgrading will be difficult and will violate the taboos in the code, which is the Open / Close Principle (OCP). However, the Observer pattern will solve that problem. they can reuse Subjects without reusing Observers and vice versa, it allows to add Observers without modifying another Subject or Observer..

1.3 Factory Pattern

When using the factory pattern, it helps to reduce dependencies between modules, making the program independent of concrete classes. It's easier to extend the code, making it easier to initialize the hidden object and handle the internal logic. Easily manage the file cycle of objects created by the Factory pattern. Agree on naming conversion.

VI. Conclusion

In this report, we learned more about OOP knowledge through concepts, diagrams and use case . beside, learn more about the design patterns and the functionality of each design pattern. And Finally compare the relationship between OOP and design pattern.

References

Akin, 2001. *Object Oriented Programming via Fortran 90/95*. Texas ed.

Akin, E., 2001. *Object Oriented Programming via Fortran 90/95*. Texas.

Akin, E., 2003. *Object-oriented programming via Fortran 90-95*. Cambridge University Press.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 2001. Design Pattern. In *Design Pattern*.

KevinZhang, 1994. *Design Patterns Elements of Reusable Object-Oriented Software*.

KevinZhang, 1994. *Design Patterns Elements of Reusable Object-Oriented Software*.

Pecinovsky, R., 2013. *Learn Object Oriented Thinking and Programming*.

Sourcemaking, 2021.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J., n.d. *Design patterns*.

Freeman, E., Freeman, E., Sierra, K., Bates, B. and Baland, M., 2011. *Design patterns*. Brest: Digit Books.