**ASSIGNMENT 1 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 20: Advanced Programming | | |
| **Submission date** |  | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** | Ngo Quynh Nga | **Student ID** | GCH18255 |
| **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** |  |

**Grading grid**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P1 | P2 | M1 | M2 | D1 | D2 |
|  |  |  |  |  |  |

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

Table of Contents

[I. Introduction 4](#_Toc75209876)

[II. OOP general concepts 4](#_Toc75209877)

[1. Definition 4](#_Toc75209878)

[2. Characteristic of OOP 4](#_Toc75209879)

[3. Advantages of OOP 5](#_Toc75209880)

[III. OOP scenario 5](#_Toc75209881)

[1. Scenario 5](#_Toc75209882)

[2. Usecase Diagram 7](#_Toc75209883)

[3. Class Diagram 9](#_Toc75209884)

[IV. Design Patterns 10](#_Toc75209885)

[1. Creational pattern 10](#_Toc75209886)

[2. Structural pattern 11](#_Toc75209887)

[a. Adapter Design Pattern 11](#_Toc75209888)

[3. Behavioral pattern 14](#_Toc75209889)

[References 15](#_Toc75209890)

# Introduction

# OOP general concepts

## Definition

Object-oriented programming(OOP) combines a collection of variables (properties) and functions (methods) into a single entity known as an "object." These items are classified into classes, which allow individual objects to be grouped together. OOP may assist you in considering objects in a program's code and the many activities that could occur in connection to the objects.

This programming approach is prevalent in widely known programming languages such as Java, C++, and PHP. These languages aid in the organization and structure of software applications. When creating complicated applications, programmers frequently utilize OOP.

## Characteristic of OOP

* **Encapsulation:**

The many objects within each application will attempt to communicate with one another automatically. If a programmer wishes to prevent items from interacting with one another, they must be contained in separate classes. Classes cannot alter or interact with an object's particular variables and functions due to the encapsulation process.

The idea of encapsulation works in a digital fashion to build a protective barrier around the information that isolates it from the rest of the code, just as a pill "encapsulates" or retains the drug inside of its coating. This object can be replicated by programmers in other areas of the program or in other applications.

* **Abstraction:**

Abstraction is similar to encapsulation in that it conceals some attributes and functions from outside code in order to simplify the interface of the objects. Abstraction is used by programmers for a variety of reasons. Overall, abstraction helps to isolate the impact of code modifications so that if something goes wrong, the change only affects the variables presented and not the outside code.

* **Inheritance:**

Using this idea, programmers may expand the functionality of existing classes in the code to reduce redundant code. For example, HTML code components such as a text box, select field, and checkbox share particular attributes with certain methods.

Rather than rewriting the attributes and methods for each type of HTML element, they may be defined once in a generic object. By naming that object "HTMLElement," other objects will inherit its attributes and functions, allowing you to minimize needless code.

The superclass is the primary object, and any objects that follow it are subclasses. Subclasses can have their own components while relying on the superclass for what they require.

* **Polymorphism:**

This method, which translates as "many forms or shapes," enables programmers to generate numerous HTML components based on the kind of item. This idea enables programmers to reimagine how something works by altering how it is done or the portions that are done. Overriding and overloading are polymorphism terms.

## Advantages of OOP

* OOP supports the modular framework for application development and offers it. Good definitions are implemented by concealing internal information, abstrait data types are implemented
* The retention and modification of the current code is easy, whilst existing code may be changed to existing objects with little modifications.
* The OOP provides an excellent code library foundation for quickly adjusting and modifying program components.

# OOP scenario

## Scenario

The selling system will be used as an example for OOP. There are three actors in this system: the customer, the seller, and the administrator. Login, search products, check details, and buy products are all available to customers. The seller can log in, search products, enter data, and check for client information such as phone numbers, etc. The administrator can log in, search for and manage all client information. Admins can also add, update, and delete products, as well as change their prices based on market conditions. There will be a memory for all of the items that will keep track of all of the locations that they have added. Memory can be managed by the administrator, but not by customers, and memory data cannot be removed. Customers can only buy products after registering an account.

## Usecase Diagram

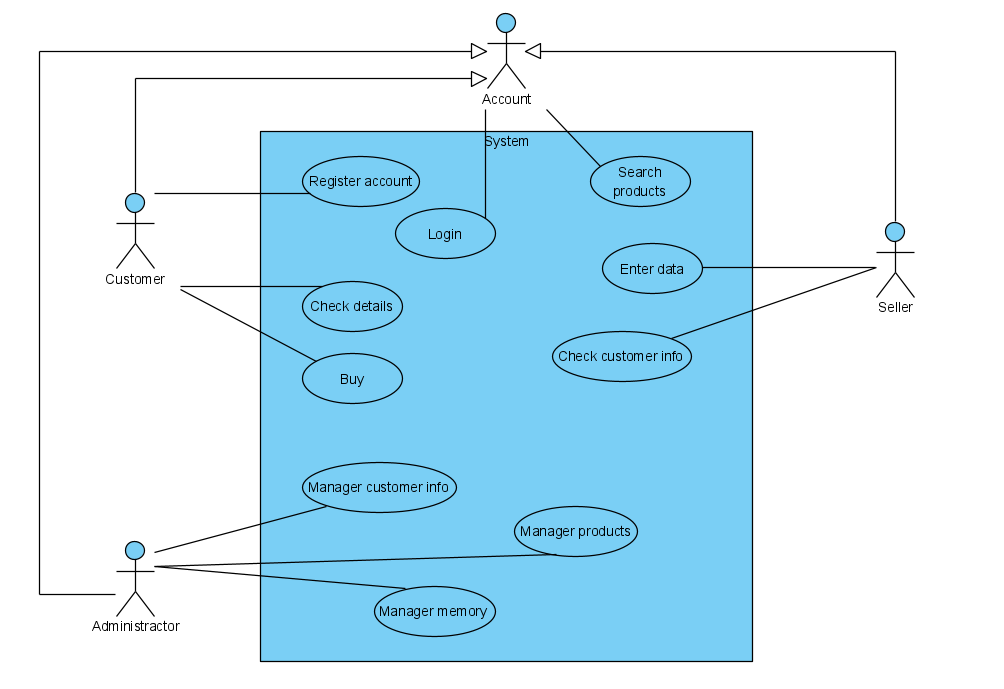


Figure 1: Usecase Diagram

In the use case diagram applied to the scenario that consists of three actors:

* Customer: Login, search products, check details products, and buy products are all available to customers. Customers can only search for products, view information of products, and buy them after registering an account.
* Seller: Seller can log in, search products, enter data, and check for client information such as phone numbers, etc.
* Administrator: Administrator can log in, search for and manage all client information. Admins can also add, update, and delete products, as well as change their prices based on market conditions. Memory can be managed by the administrator.

## Class Diagram

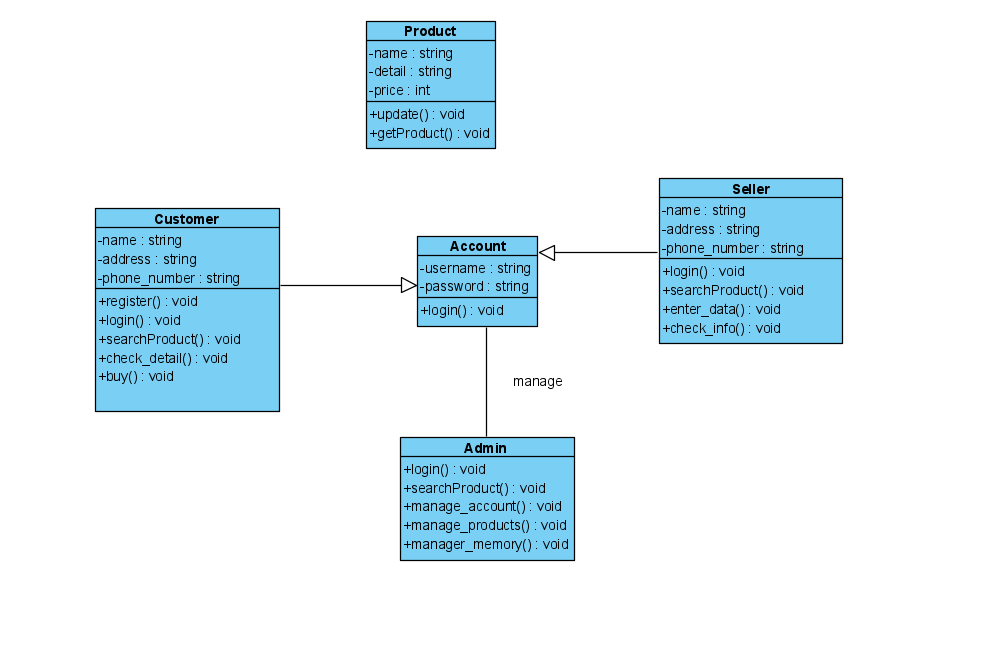


Figure 2: Class Diagram

# Design Patterns

**Definition:**

Design Patterns are a technique in object-oriented programming. Design Pattern is used frequently in OOP languages. It provides you with “design patterns”, solutions to common problems, commonly encountered in programming. The problems you face may be your own to come up with a solution, but it may not be optimal. Design Pattern helps you solve problems in the most optimal way, providing you with solutions in OOP programming (Coder, 2018).

**According to** (Beasley, 2017) **pros and cons of design pattern**

**Pros:**

* Easy to adapt to predictable changes in business needs.
* Easy to unit test and validate individual components.
* Can provide organization and structure when business requirements become very complicated.

**Cons:**

* Beginner engineers may not understand them.
* Oftentimes used improperly without a realistic understanding of how the software is likely to change.
* Can add memory and processing overhead, so sometimes not appropriate for applications such as low level systems programming or certain embedded systems.

## Creational pattern

**Definition:**

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done (Making, 2021).

* **Abstract Factory:** Creates an instance of several families of classes (Making, 2021).
* **Builder:** Separates object construction from its representation (Making, 2021).
* **Factory Method:** Creates an instance of several derived classes (Making, 2021).
* **Object Pool:** Avoid expensive acquisition and release of resources by recycling objects that are no longer in use (Making, 2021).
* **Prototype:** A fully initialized instance to be copied or cloned (Making, 2021).
* **Singleton:** A class of which only a single instance can exist (Making, 2021).

## Structural pattern

**Definition:**

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality (Making, 2021).

* **Adapter:** Match interfaces of different classes (Making, 2021).
* **Bridge:** Separates an object’s interface from its implementation (Making, 2021).
* **Composite:** A tree structure of simple and composite objects (Making, 2021).
* **Decorator:** Add responsibilities to objects dynamically (Making, 2021).
* **Façade:** A single class that represents an entire subsystem (Making, 2021).
* **Flyweight:** A fine-grained instance used for efficient sharing (Making, 2021).
* **Private Class Data:** Restricts accessor/mutator access (Making, 2021).
* **Proxy:** An object representing another object (Making, 2021).

### Adapter Design Pattern

**Definition:**

Adapter Pattern is one of the Patterns in the Structural Pattern group. The Adapter Pattern allows unrelated interfaces to work together without having to modify them directly. The object that helps to connect the interfaces is called Adapter (Coder, 2018).

An Adapter Pattern consists of the following basic components:

* Client: class that uses objects with the Target interface.
* Target: an interface containing the functions used by the Client.
* Adaptee: interface definition is not compatible, needs to be integrated.
* Adapter: integration class, helping incompatible interfaces integrate with the working interface. Perform interface conversion for Adaptee and connect Adaptee to Client.

**When to use the Adapter Design Pattern:**

By using the adapter, we don’t need to spend much of time to restructure the old resource. It will be acceptable when we restructure the resource the first time, but it will be very waste of time when we want to restructure the resource more than one time. This is the reason why the adapter is used.

**Basic Structure:**

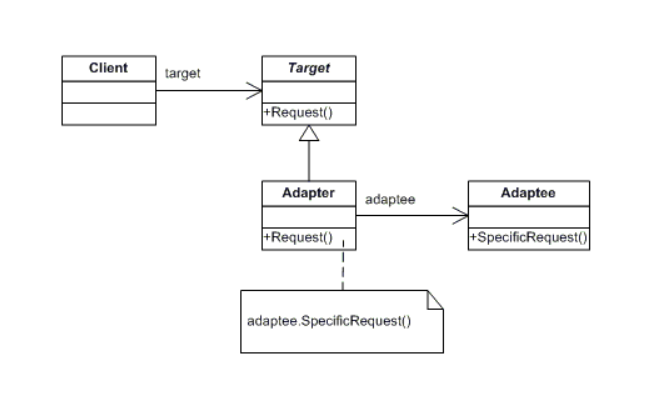
****

Figure 3: Basic Adapter Structural Diagram (dofactory)

**Scenario:**

A customer bank account management application. Bank details include account holder name, account number and bank name. Currently the bank wants to issue a credit card based on the account holder name, account number and valid bank name to issue a credit card. We need to use Adapter to convert bank details including account holder name, account number and bank name to deploy credit card issuance based on bank details

**UML class diagram:**

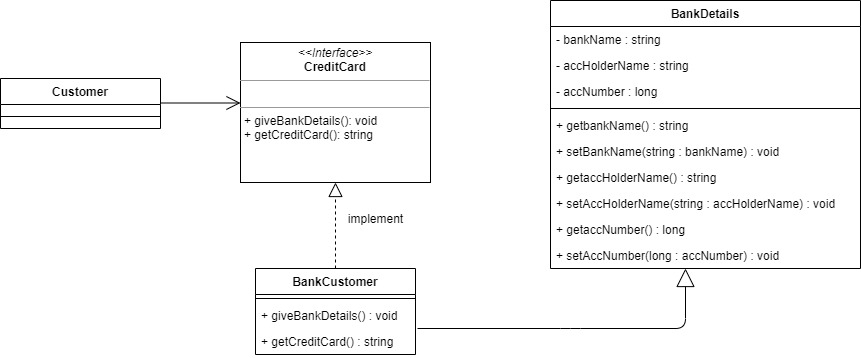


Figure 4: Adapter Diagram

**UML explain:**

There are four classes in my class diagram. The relationship between Client and CreditCard is an association relationship. And BankCustomer inherits BankDetails and then implements it in CreditCard class.

* Client: Customer is Client
* Target: CreditCard is Target Interface. This is the desired interface class which will be used by the clients namely the issuance of credit cards
* Adaper: BankCustomer is Adaper. This class is a wrapper class which implements the desired target interface and modifies the specific request available from the Adaptee class. Inherit the information in the BankDetails class and then implement it in the CreditCard class.
* Adaptee: BankDetails is Adaptee. This is the class which is used by the Adapter class to reuse the existing functionality and modify them for desired use.

## Behavioral pattern

**Definition:**

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects (Making, 2021).

* **Chain of responsibility:** A way of passing a request between a chain of objects (Making, 2021).
* **Command:** Encapsulate a command request as an object (Making, 2021).
* **Interpreter:** A way to include language elements in a program (Making, 2021).
* **Iterator:** Sequentially access the elements of a collection (Making, 2021).
* **Mediator:** Defines simplified communication between classes (Making, 2021).
* **Memento:** Capture and restore an object's internal state (Making, 2021).
* **Null Object:** Designed to act as a default value of an object (Making, 2021).
* **Observer:** A way of notifying change to a number of classes (Making, 2021).
* **State:** Alter an object's behavior when its state changes (Making, 2021).
* **Strategy:** Encapsulates an algorithm inside a class (Making, 2021).
* **Template method:** Defer the exact steps of an algorithm to a subclass (Making, 2021).
* **Visitor:** Defines a new operation to a class without change (Making, 2021).

# References

Beasley, J., 2017. *What are some pros and cons of using Design Patterns to describe your business model?.* [Online]   
Available at: https://www.quora.com/What-are-some-pros-and-cons-of-using-Design-Patterns-to-describe-your-business-model  
[Accessed 20 June 2021].

Coder, G., 2018. *Giới thiệu Design Patterns.* [Online]   
Available at: https://gpcoder.com/4164-gioi-thieu-design-patterns/  
[Accessed 19 June 2021].

Coder, G., 2018. *Hướng dẫn Java Design Pattern – Adapter.* [Online]   
Available at: https://gpcoder.com/4483-huong-dan-java-design-pattern-adapter/  
[Accessed 19 June 2021].

Making, S., 2021. *Design Patterns.* [Online]   
Available at: https://sourcemaking.com/design\_patterns  
[Accessed 20 June 2021].