



Higher Nationals in Computing

**Unit 20: Advanced Programming  
ASSIGNMENT 1**

Learner’s name: Trần Gia Huy

ID: GCS190255

Class: GCS0805

Subject code: 1651

Assessor name: **PHAN MINH TÂM**

Assignment due: Assignment submitted:

**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** | Trần Gia Huy | **Student ID** | GCS190255 |
| **Class** | GCS0805 | **Assessor name** | Phan Minh Tâm |
| **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:** |
| **Signature & Date:** | | |

**ASSIGNMENT 1 BRIEF**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Business** | | |
| **Unit number** | Unit 20: Advanced Programming | | |
| **Assignment title** | Examine and design solutions with OOP and Design Patterns | | |
| **Academic Year** | 2021 - 2022 | | |
| **Unit Tutor** | Phan Minh Tam | | |
| **Issue date** | 28 September 2021 | **Submission date** |  |
| **IV name and date** |  | | |

|  |
| --- |
| **Submission Format:** |
| *Format:* The submission is in the form of a **group written report**. This should be written in a concise, formal business style using single spacing and font size 12. You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system. Please also provide a bibliography using the Harvard referencing system.  *Submission* Students are compulsory to submit the assignment in due date and in a way requested by the Tutors. The form of submission will be a soft copy in PDF posted on corresponding course of <http://cms.greenwich.edu.vn/>  *Note:* The Assignment *must* be your own work, and not copied by or from another student or from  books etc. If you use ideas, quotes or data (such as diagrams) from books, journals or other sources, you must reference your sources, using the Harvard style. Make sure that you know how to reference properly, and that understand the guidelines on plagiarism. *If you do not, you definitely get fail* |
| **Assignment Brief and Guidance:** |
| **Scenario**: You have recently joined a software development company to help improve their documentation of their in-houses software libraries which were developed with very poor documentation. As a result, it has been very difficult for the company to utilise their code in multiple projects due to poor documentation. Your role is to alleviate this situation by showing the efficient of UML diagrams in OOAD and Design Patterns in usages.  **Tasks**  You and your team need to explain characteristics of Object-oriented programming paradigm by applying Object-oriented analysis and design on a given (assumed) scenario. The scenario can be small but should be able to presents various characteristics of OOP (such as: encapsulation, inheritance, polymorphism, override, overload, etc.).  The second task is to introduce some design patterns (including 3 types: creational, structural and behavioral) to audience by giving real case scenarios, corresponding patterns illustrated by UML class diagrams.  To summarize, you should analyze the relationship between the object-orientated paradigm and design patterns.  The presentation should be about approximately 20-30 minutes and it should be summarized of the team report. |

|  |  |  |
| --- | --- | --- |
| **Learning Outcomes and Assessment Criteria** | | |
| **Pass** | **Merit** | **Distinction** |
| **LO1** Examine the key components related to the object-orientated programming paradigm, analysing design pattern types | | |
| **P1** Examine the characteristics of the object-orientated paradigm as well as the various class relationships. | **M1** Determine a design pattern from each of the creational, structural and behavioural pattern types. | **D1** Analyse the relationship between the object-orientated paradigm and design patterns. |
| **LO2** Design a series of UML class diagrams | | |
| **P2** Design and build class diagrams using a UML tool. | **M2** Define class diagrams for specific design patterns using a UML tool. | **D2** Define/refine class diagrams derived from a given code scenario using a UML tool. |

**Table of Contents**

[1. Introduction 1](#_Toc85361992)

[2. Oop general concepts 1](#_Toc85361993)

[**2.1 Introduction OOP** 1](#_Toc85361994)

[**2.1.1 Class:** 2](#_Toc85361995)

[**2.1.2 Object** 3](#_Toc85361996)

[**2.1.4 Encapsulation** 5](#_Toc85361997)

[**2.1.5 Inheritance** 6](#_Toc85361998)

[**2.1.6 Polymorphism** 7](#_Toc85361999)

[3. Object-orientated class relationships 7](#_Toc85362000)

[**3.1. Association** 7](#_Toc85362001)

[**3.2 Aggregation** 9](#_Toc85362002)

[**3.3 Composition** 9](#_Toc85362003)

[**3.4 Dependency** 10](#_Toc85362004)

[4. OOP scenario 10](#_Toc85362005)

[**4.1 Scenario** 10](#_Toc85362006)

[**4.2 Usecase Diagram** 11](#_Toc85362007)

[**4.3 Class Diagram** 12](#_Toc85362008)

[**4.4 Sequence diagrams** 13](#_Toc85362009)

[REFERENCES 18](#_Toc85362010)

**Table of Figures**

[**Figure 1** 1](#_Toc85448702)

[**Figure 2** 2](#_Toc85448703)

[**Figure 3** 3](#_Toc85448704)

[**Figure 4** 4](#_Toc85448705)

[**Figure 5** 4](#_Toc85448706)

[**Figure 6** 5](#_Toc85448707)

[**Figure 7** 6](#_Toc85448708)

[**Figure 8** 7](#_Toc85448709)

[**Figure 9** 8](#_Toc85448710)

[**Figure 10** 9](#_Toc85448711)

[**Figure 11** 9](#_Toc85448712)

[**Figure 12** 10](#_Toc85448713)

[**Figure 13** 11](#_Toc85448714)

[Figure 14 12](#_Toc85448715)

[**Figure 15** 13](#_Toc85448716)

[**Figure 16** 14](#_Toc85448717)

[**Figure 17** 15](#_Toc85448718)

[**Figure 18** 16](#_Toc85448719)

[**Figure 19** 17](#_Toc85448720)

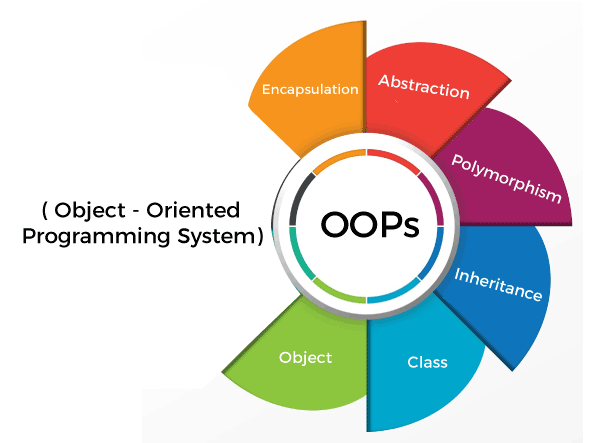
**ASSIGNMENT 1 ANSWERS**

# Introduction

This report will introduce about OOP (short for Object Oriented Programming) and how they work. Next we have to draw OOP diagrams and learn each stage and their functions. We will know their steps and explain each part.

# 2. Oop general concepts

## **2.1 Introduction OOP**

­

**Figure 1**

Object – Oriented Program (OOP) is refers to a programming method that uses object and classes. This is a prototype that is developed based on the concept of subject technology. In there, the subject contains the data, in the field is commonly known as identity. The source code is hosted as protocol for the subject to access, to edit other objects's data fields that the object is currently associated with.

A simple understanding, programmed the subject to support the OOP technology. It's working to increase productivity and accelerate the process. That word allows the programmers to affect the hosts as they are in fact. Go over there to repair the software, maintenance, maintain the system. The language of the subject is very divers, most of it is programming languages. Each subject has the ability to receive any information, process the data and send answers to other subjects or to the environment.

OOP usually focuses on manipulating objects rather than logic to be able to manipulate them, and it is a familiar foundation of today's design patterns. OOP aims to manage source code to increase reusability and, most importantly, be able to summarize the known procedures through the use of objects.

Characterictist in Object- Oriented paradigm:

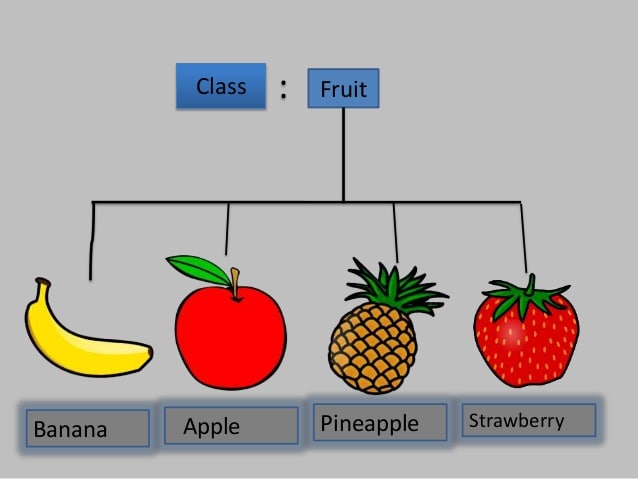
**- Class and Object**

**- Encapsulation**

**- Polymorphism**

**- Inheritance**

### **2.1.1 Class:**



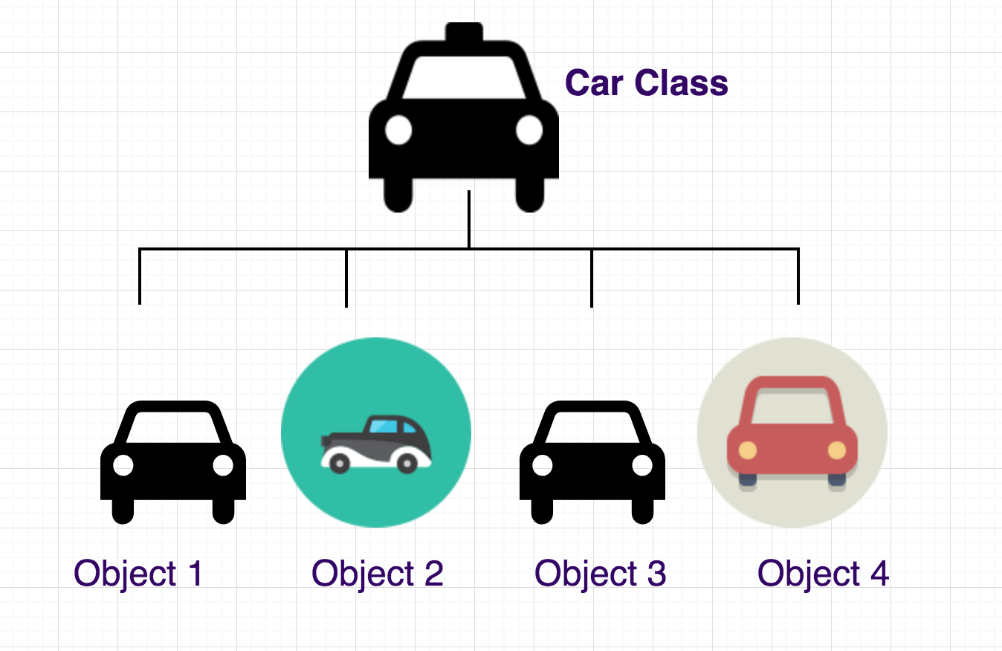
**Figure 2**

A class is a model or blueprint or prototype of an object that defines or specifies all the properties of the objects. Classes have their associated data and functionality wrapped in them. Class defines the state and behaviors of an object. Before creating an object, we know what properties or data members an object will contain based on the class. Classes contain data members and member functions. Class is a logical and not physical entity because class represents a category of objects.

Example:

Fruit is a non-physical type that represents a type for objects like Apples, Bananas, Mangoes, Pineapple etc. With common attributes like color, taste, price, etc.

### **2.1.2 Object**

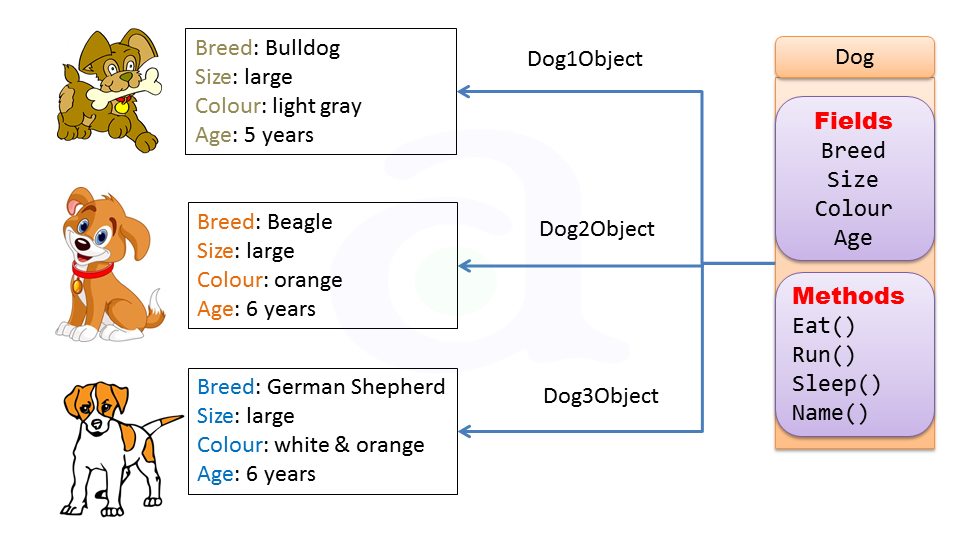


**Figure 3**

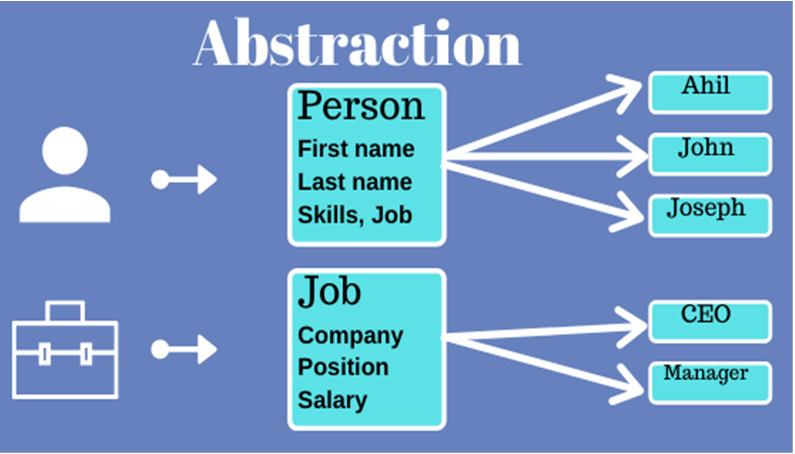
An object is an instance or instance of a class. Objects are primarily a physical entity but it can also be a logical entity. Every object has state and behavior. State of objects means values ​​or data in the form of variables and behavior means actions or functions that perform some operation on the data. Whether. Objects exist together with classes. They are essentially separate structured data fields that can be defined by the programmer. When you call an object, the program creates an instance. An instance is a concrete object created from a class.

Example:

Dogs have state (name, color, breed, hunger) and behavior (barking, grabbing, tail wagging).



**Figure 4**

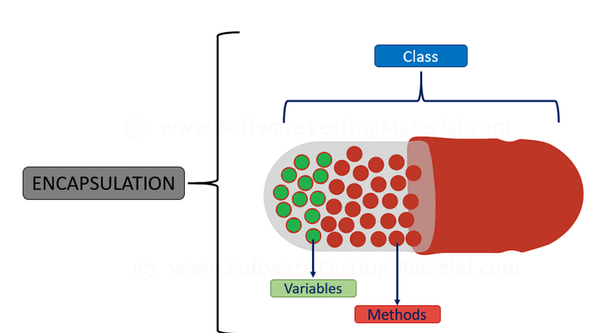
**2.1.3 Abstration**

**Figure 5**

Abstraction is a powerful methodology for managing complex systems. Abstraction is managed by well-defined objects and their hierarchical classification. Abstraction shares the same benefits and origins as encapsulation. The principle of abstraction applies when a program only discloses to the user data relevant to the object. Its main goal is to reduce complexity by hiding details that are not directly relevant to the user. That allows the user to continue the work necessary to rely on a abstract entity that is provided without understanding or even think of all the elaborate complications hidden inside.

Example : Your car is a great example of abstraction. You can start a car by turning the key or pressing the start button. You don't need to know how the engine is getting started, what all components your car has. The car internal implementation and complex logic is completely hidden from the user.

### **2.1.4 Encapsulation**

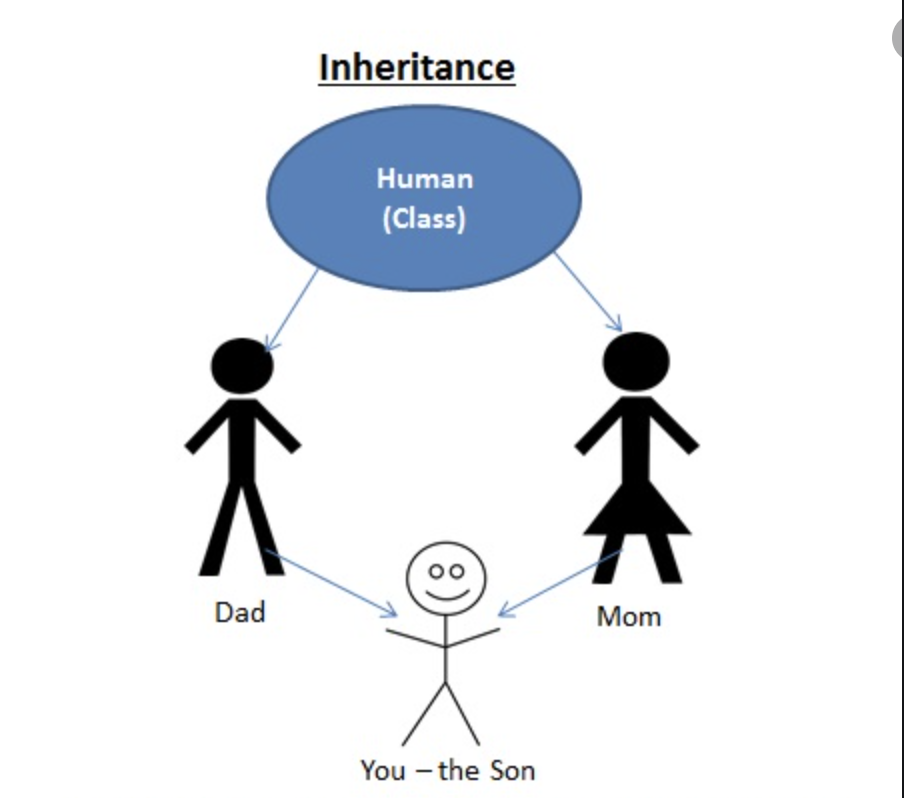


**Figure 6**

Encapsulation is the binding of an object's state and behavior together. This happens when properties and methods are wrapped into a unit and/or class. Due to encapsulation, class fields are not directly accessible to the public. In addition, encapsulation offers flexibility and maintainability. Encapsulation is the process of grouping or wrapping data and functions to perform actions on data into a single unit. The single unit is called a class. Packaging is like wrapping in a capsule. It is the grouping of related operations and data related to an object into that object. It keeps data and code safe from outside interference. The primary purpose or use of encapsulation is to provide security for a class's data. To ensure data security, we need to use private access modifiers that will restrict access to data outside the class. Access modifiers are used to define the access level or scope of class members such as data and function members.

Example: all the three data members (or data fields) are private which cannot be accessed directly. These fields can be accessed via public methods only. Fields empName, ssn and empAge are made hidden data fields using encapsulation technique of OOPs.

### **2.1.5 Inheritance**

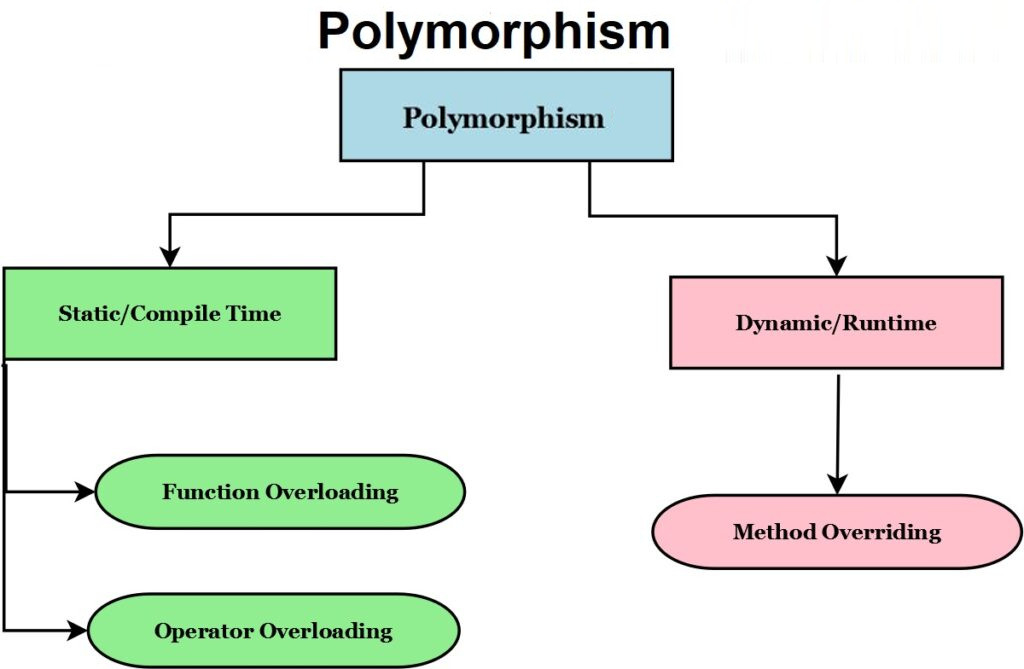


**Figure 7**

Inheritance is the process of acquiring the properties of a base class into a subclass. The concept of inheritance is very useful for reusability. When a program has many classes with related properties, inheritance can simplify the landscape. In object-oriented programming, subclasses can inherit parent classes and similar features. Inheritance is a word that includes the word "Inherit", which means "To develop". So Inheritance is defined as the tendency of a class to derive properties and characteristics from other classes. It provides additional functionality to extract features from the base class and significantly imply it into other derived classes. It provides a mechanism for establishing relationships and building class hierarchies in object composition. Inheritance means the use of pre-written or pre-generated code. And one thing to keep in mind is that we are just using the code and not updating or changing it. Functions and methods defined in a class can be used to manipulate other data members of the class.

Example: car, bus, bike – all of these come under a broader category called Vehicle. That means they've inherited the properties of class vehicles i.e all are used for transportation.

### **2.1.6 Polymorphism**



**Figure 8**

Polymorphism is the condition that an object can have many forms. That said, polymorphism can play out in many different ways. it means a method with multiple implementations, for a certain type of action. And which implementation to use will be decided at runtime depending on the situation (i.e. the data type of the object). The literal meaning of polymorphism is a state that has many forms. Applying this concept to object-oriented programming system (OOPS) languages ​​like Java and C++ makes it easy to deal with. This language can handle many different types of classes and objects through a single unified interface. Basically, polymorphism is one of OOPS's ability to redefine native class methods. The behavior of polymorphic functions depends on the data type used in the programming. Polymorphism is often used to implement program consistency.

An example of polymorphism in an object-oriented language is the relationship between a parent class object and a child class object. Objects that can satisfy IS-Many relationships become polymorphic In essence, object-oriented programming languages ​​(OOPS) have two main types of polymorphism.

# 3. Object-orientated class relationships

## **3.1. Association**



**Figure 9**

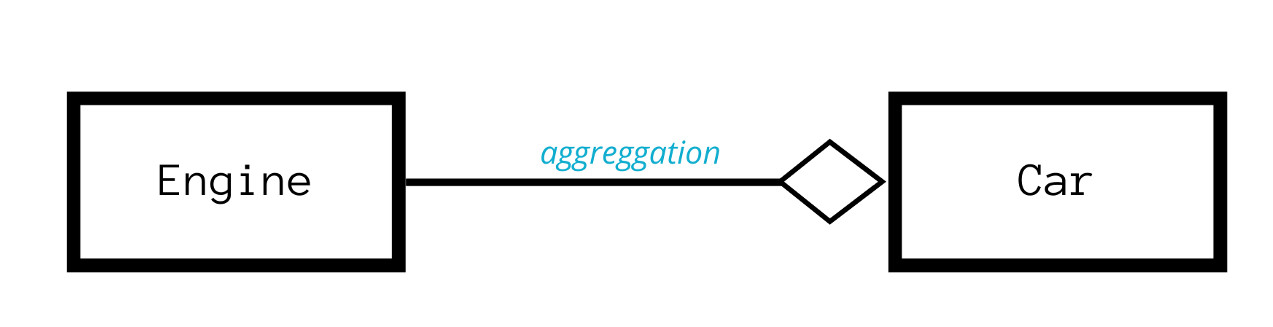
In the common software of the subject, the relationship between this object's function and another subject is called a bond. Note that the connection between the two subjects is not like the heir between the two layers. The link means a subject uses another subject or a function of another object. In other words, the bond is a relationship between the subject when a subject has one or more reference to other subjects. The coven is a relationship with the subjects. The bond is a relationship between the subjects. In the union, the relationship between the subjects determines what a subject can show another subject to perform an act on its behalf. We can also say that a possible combination of diversity between the subjects. We can identify the one-one relationship, a-a-lot, many-one and many-more-amongst the subjects. The League is a more comprehensive term to determine the relationship between the subjects. The bond means a subject of using another subject.

Example: A student can associate ( associate ) with many teachers. Faculty students have 2 teachers, Teacher1, Teacher2

Instances of the Teacher class are created outside of the Student class. So when the faculty was canceled, the two teachers, Teacher1, Teacher2 were not affected.

Other students like Teo, Lan, Ty can associate (associate) with Teacher through instances of Teacher1, Teacher2. So we say that Student has no ownership relationship with Teacher at all.

### **3.2 Aggregation**

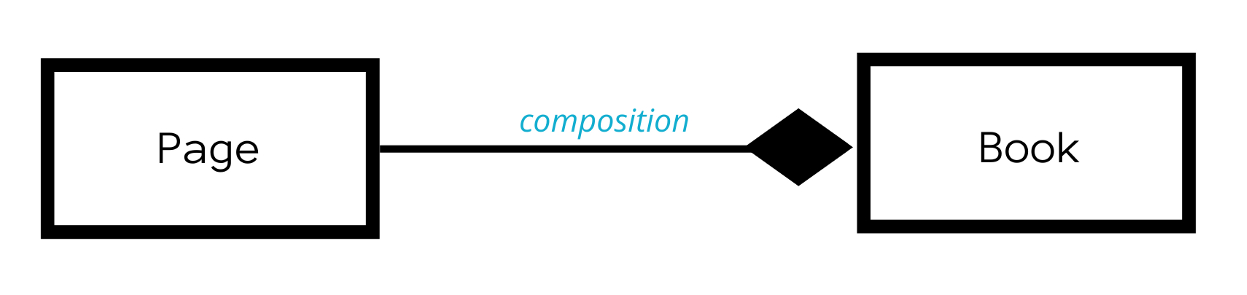


**Figure 10**

Aggregation is another form of association and is similar to composition, in that each object has its own life cycle but concurrent ownership exists. Aggregation is a typical full/partial or parent/child relationship but it may or may not represent material restraint. An essential property of an aggregate relationship is that the whole or parent (i.e. owner) can exist without a part or child and vice versa. Aggregation can feature a one-to-one, one-to-many, or many-to-many link between the participating items, similar to association. A one-to-many or many-to-many connection is referred to as a redundant relationship.

For example, an employee may belong to one or more departments within an organization. However, if the employee's department is deleted, the employee object will not be destroyed but will live on. Note that relationships between objects participating in a set are not mutual — that is, a department can “own” an employee, but that employee does not own the department.

### **3.3 Composition**



**Figure 11**

Composition is a powerful link. An association is said to be a composition if one object owns another and another object cannot exist without the owner object. Composition is an even stronger relationship than aggregation. To check if you are dealing with composition you should use the "no sharing rule". This rule says that in a layout relationship the part can only belong to one hole. So in our example above, the musician belongs to only one band? This is not the truth. A musician can be in several different bands. Same goes for product category relationships. Any particular product can appear in a number of different categories.

### **3.4 Dependency**



**Figure 12**

A dependency is defined as a relationship between two classes where one class depends on another but another class may or may not depend on first class. Hence any change in one class may affect the functionality of another, that depends on the first class. This is represented by a dashed arrow.

Let's take another example of a customer-supplier relationship. Customers depend on suppliers to deliver products. If the supplier does not provide the products, the customer cannot use those products.

# 4. OOP scenario

## **4.1 Scenario**

In the library need a book management software in the software library to make the book management system more convenient in :Add, Show all the book in store, Search book id, search book name, search author’s name.

-Library: books (name book, id book, date/month/year book, book genre, author's name)

-Reader: address, reader's name

-Borrowing slip: Borrow date and return date

-Functions of a software:

+ Add book title

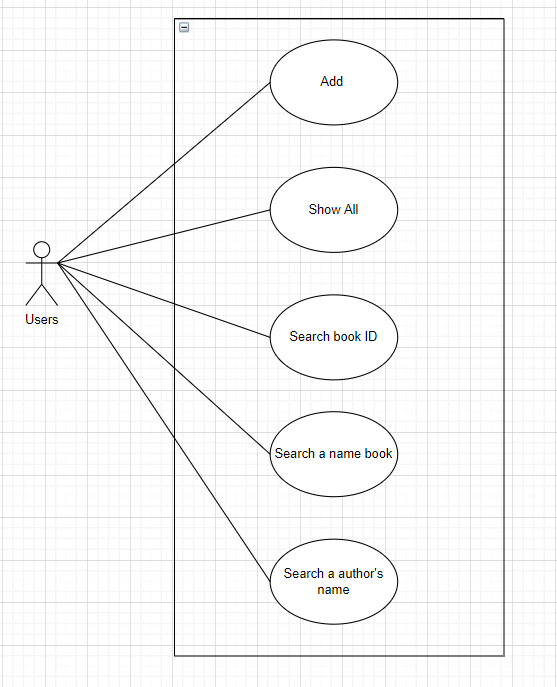
+ Show all

+ Search book id

+ Search a book name

+ Search the author's name.

### **4.2 Usecase Diagram**



**Figure 13**

### **4.3 Class Diagram**

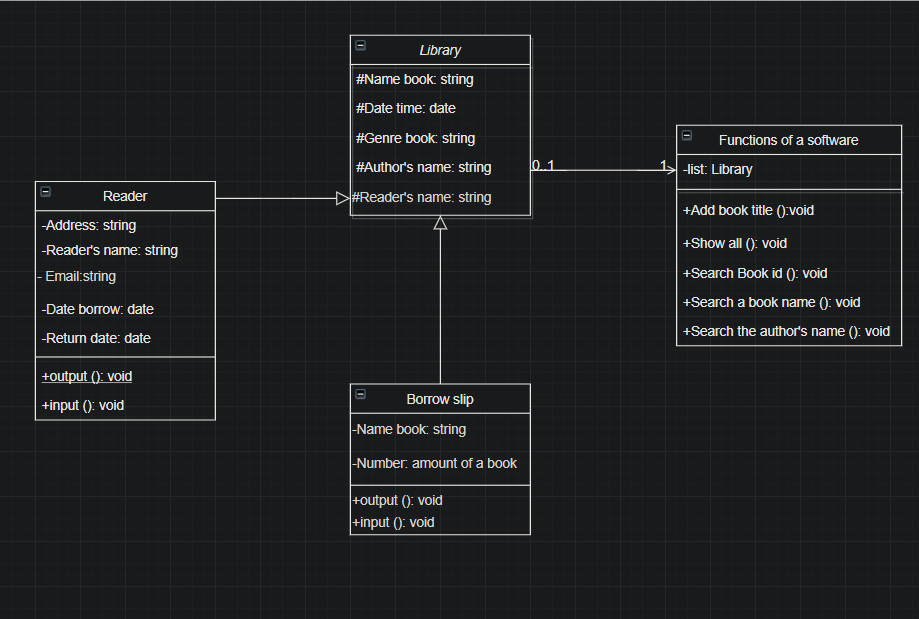


Figure 14

-Describe the data of the classes in the relationship system between the library and the borrower exists so that we can simply communicate and understand each other's opinion in the same project.

- The plus sign (+) at the top indicates public access, the minus (-) sign indicates privacy, and the pound sign indicates protected access.

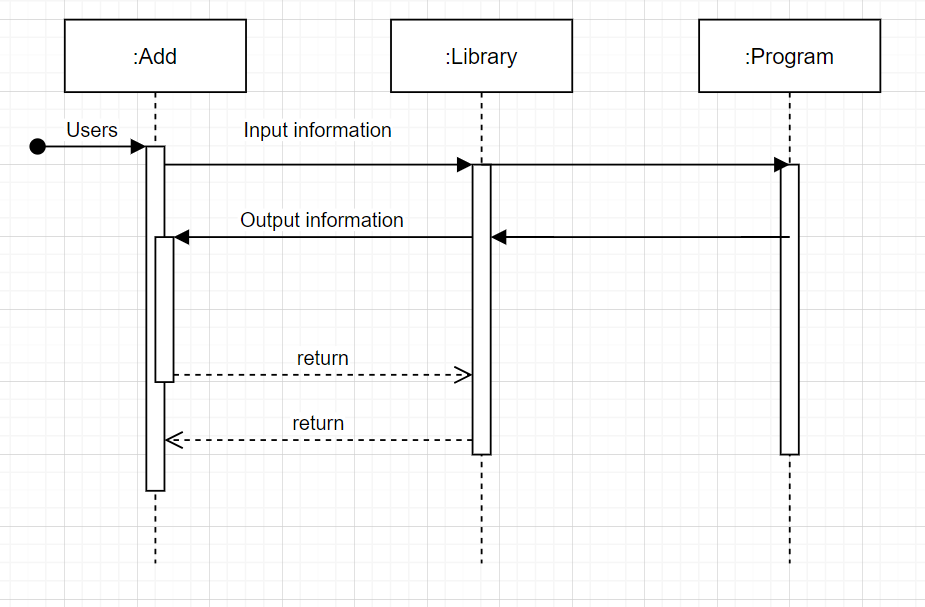
-After the second dot (:) Show the data type.

-Functions of a software consists of the Add book title, show all, search a book name, search a book name, search the name, search the name of the Input protocol. It's part of the menu so the library manager can find books and borrowers.

-The library class will be the successor of Reader and Borrow slip and it has a one-to-one relationship with Functions of a software.

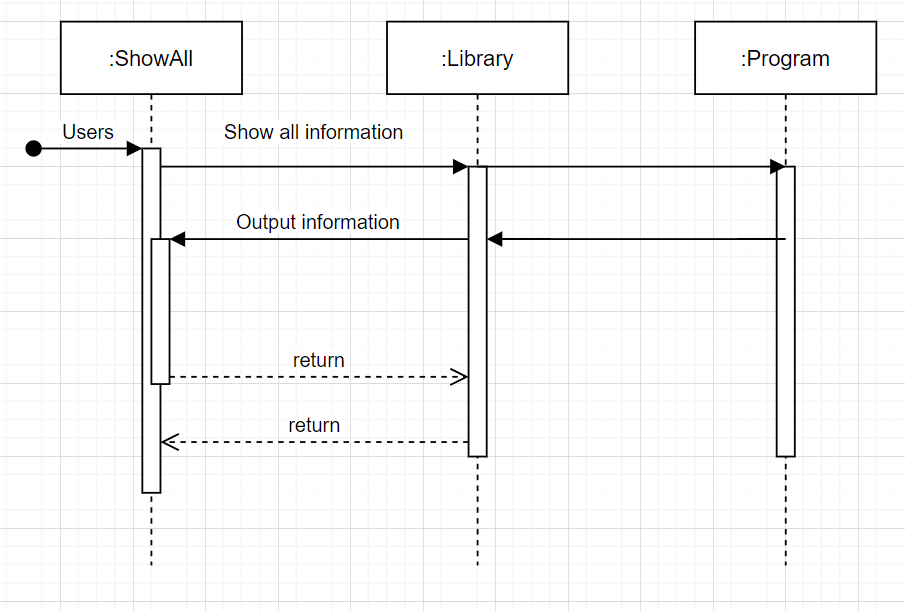
### **4.4 Sequence diagrams**

- Library management when using the software, they can choose many methods that the software already has such as: Add book title, Show all, search book id, search a book name, search the author's name.



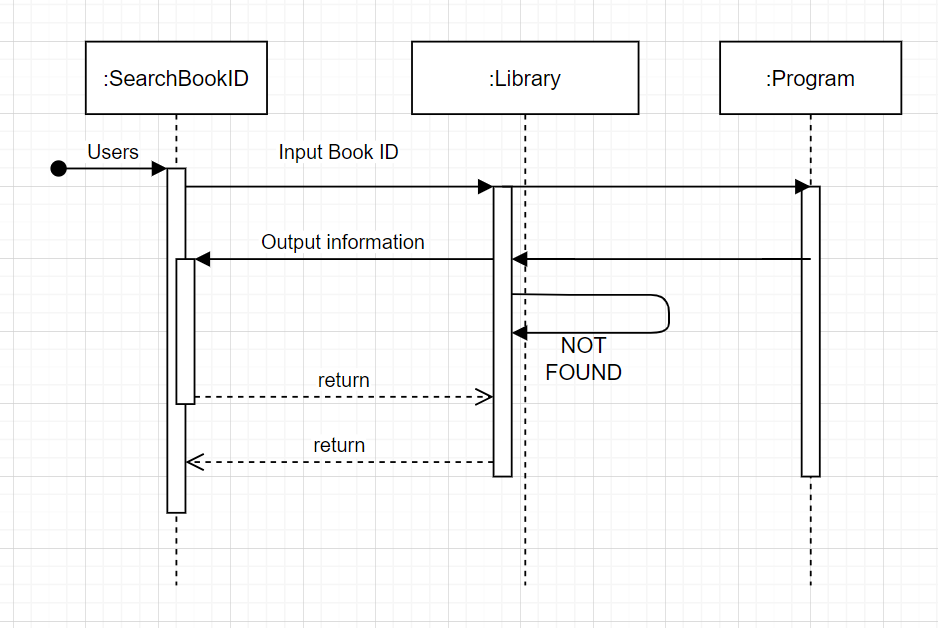
**Figure 15**

-When the library manager selects the Add book title command, requests such as Name book, author's book and Genre book will appear. In order for the manager to enter the full information of the book, this information will be stored in the software.



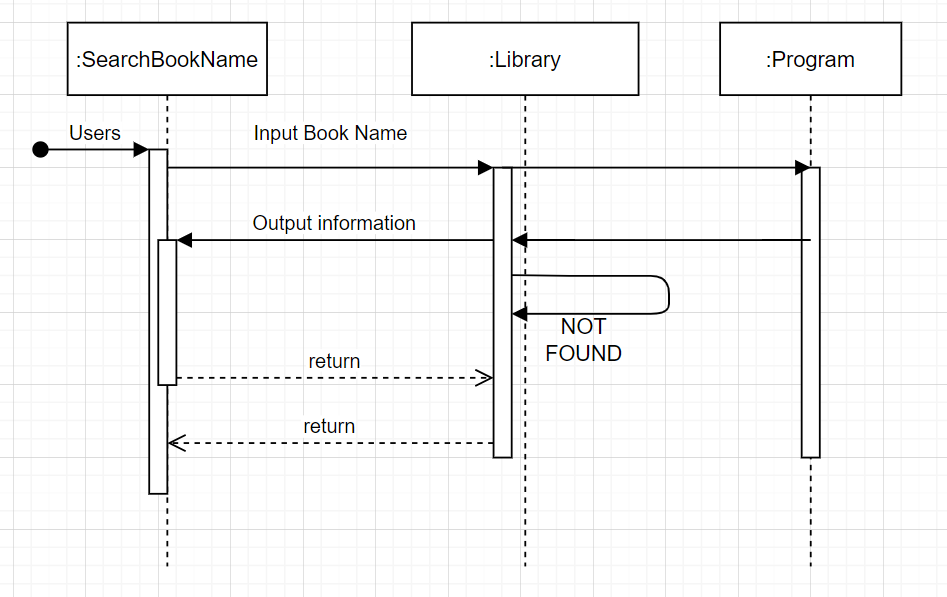
**Figure 16**

-Library management using the Show all command will display all the information of the Library section that has stored the information that has been entered from the input manager.



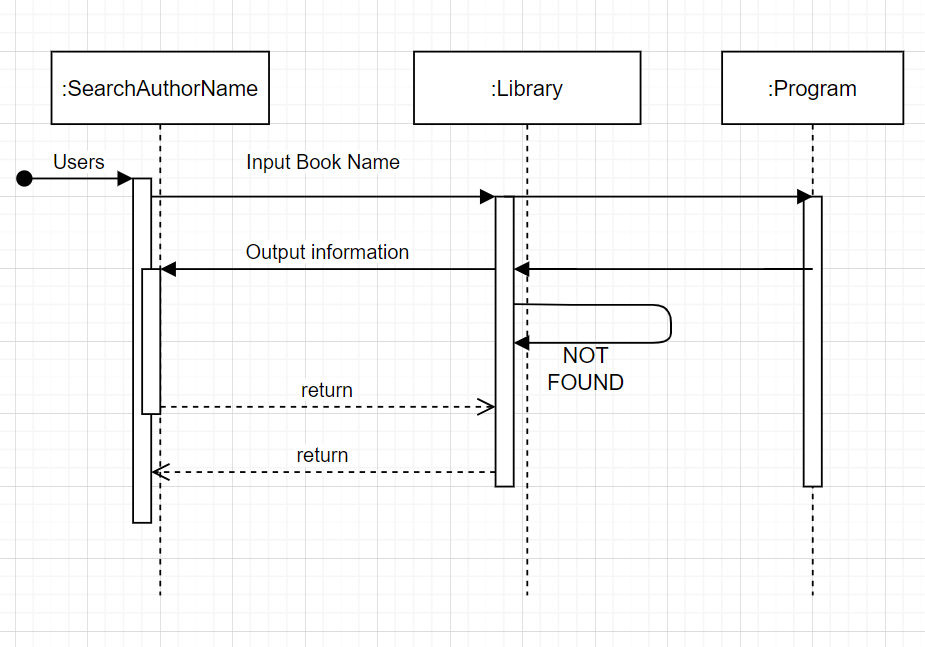
**Figure 17**

-Library manager when search book id, must enter the ID number in which the library has been divided and store the same ID number information as the library manager entered. From there, the information of the ID Book will be printed on the screen.



**Figure 18**

**-**Library manager when search book name, must enter the name of the book in which the library has been divided and store the same the name of the book information as the library manager entered. From there, the information of the name of the book will be printed on the screen.



**Figure 19**

-Library manager when search the author’s name, must enter the author’s name in which the library has been divided and store the same author’s name information as the library manager entered. From there, the information of the author’s name will be printed on the screen.

# REFERENCES

1. Blog | Got It AI. 2021. *4 tính chất của OOP - Lập trình hướng đối tượng - Blog | Got It AI*. [online] Available at: <https://vn.got-it.ai/blog/4-tinh-chat-cua-oop-lap-trinh-huong-doi-tuong> [Accessed 16 October 2021].
2. Guru99. 2021. *What is Class and Object in Java OOPS? Learn with Example*. [online] Available at: <https://www.guru99.com/java-oops-class-objects.html> [Accessed 16 October 2021].
3. TopDev. 2021. *Lập trình hướng đối tượng là gì? | Object Oriented Programming - OOP*. [online] Available at: <https://topdev.vn/blog/oop-la-gi/> [Accessed 16 October 2021].
4. Study.com. 2021. [online] Available at: <https://study.com/academy/lesson/oop-object-oriented-programming-objects-classes-interfaces.html> [Accessed 16 October 2021].
5. 2021. [online] Available at: <https://viblo.asia/p/oop-the-nao-la-mot-object-tot-gAm5ypMwldb> [Accessed 16 October 2021].
6. bugnetproject. 2021. *OOP là gì? Giải thích về lập trình hướng đối tượng*. [online] Available at: <http://bugnetproject.com/oop-la-gi-giai-thich-lap-trinh-huong-doi-tuong/> [Accessed 16 October 2021].
7. Blog ITNavi. 2021. *OOP là gì? Các nguyên lý cơ bản của lập trình hướng đối tượng - Blog ITNavi*. [online] Available at: <https://itnavi.com.vn/blog/oop-la-gi/> [Accessed 16 October 2021].
8. SearchAppArchitecture. 2021. *What is Object-Oriented Programming (OOP)?*. [online] Available at: <https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP> [Accessed 16 October 2021].
9. Educative: Interactive Courses for Software Developers. 2021. *What is Object Oriented Programming? OOP Explained in Depth*. [online] Available at: <https://www.educative.io/blog/object-oriented-programming> [Accessed 16 October 2021].
10. Cite This For Me. 2021. *Save Time and Improve your Marks with CiteThisForMe, The No. 1 Citation Tool*. [online] Available at: <https://www.citethisforme.com/> [Accessed 16 October 2021].
11. <hi@ngoclb.com>, N., 2021. *OOP là gì? 4 đặc tính cơ bản của OOP*. [online] ITviec. Available at: <https://itviec.com/blog/oop-la-gi/> [Accessed 16 October 2021].