**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** | Nguyen Phu Tien | **Student ID** | GCH210269 |
| **Class** | GCH1005 | **Assessor name** | Do Hong Quan |
| **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** | Tien |

**Grading grid**

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

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

Contents

[I.Introduction 4](#_Toc133000713)

[II.OOP general concepts (P1,P2) 5](#_Toc133000714)

[III.OOP scenario (P1,P2) 6](#_Toc133000715)

[III.1.Scenario 6](#_Toc133000716)

[III.2.Usecase Diagram 7](#_Toc133000717)

[III.3.Class Diagram 8](#_Toc133000718)

[IV.Conclusion 10](#_Toc133000719)

[V.References 10](#_Toc133000720)

# I.Introduction

In this assignment, I will provide the concepts of OOPs which is stand for Object-Oriented Programming.I'll also talk about the scenario, use case diagram, and class diagram that will be used to tackle the problem in the scenario. In addition, I'll talk about the design pattern and the situation that go well with each design pattern. I'll discuss the connection between design patterns and OOP last but not least.

# II.OOP general concepts (P1,P2)

- A method of programming referred to as object-oriented programming (OOP) focuses software design on data rather than functions and logic. A data field getting unique features and behavior is called an object. Instead of focusing on the logic needed to control the objects, object-oriented programming (OOP) focuses on the objects that developers want to handle. Massive, complex, frequently updated or maintained projects are best suited for this type of programming. This involves mobile apps as well as design and manufacturing software; for instance, OOP may be utilized for modeling manufacturing systems. The nature of object-oriented software makes it beneficial for organizing duties during collaborative development. OOP also has the benefits of efficiency, scalability, and reuse of code. (Gillis, 2021)

- OOPs primarily consists of four elements: class, object, method, and attribute. Classes, which are user-defined data types, provide the foundation for individual objects, attributes, and methods. There are properties and methods in the class. Attributes, which are specified in the class template, serve as a representation of an object's state. The attributes field of objects will save data. The characteristics of the class represent the characteristics of the class. Methods are functions that are declared within a class and describe how an object operates. Each method in a class definition starts with a reference to an instance object. Instance methods also refer to an object's constituent subroutines. Programmers employ methods to guarantee reuse and limit functionality to a single object at a time. (Gillis, 2021)

- On the other hand, OOPs have four key characteristics :

+ **Abstraction** : In object-oriented programming, objects only reveal internal mechanisms that are necessary for the use of other objects, while hiding any extraneous implementation code. This allows for the expansion of functionality in derived classes, making it easier for developers to make adjustments or additions over time. Essentially, the goal is to create flexible and adaptable code that can be easily modified without affecting other parts of the system. (Gillis, 2021)

+ **Encapsulation** : The concept states that an object has access to all the data it needs, but that only a piece of it is visible to other things. Each object's implementation and state are kept secret in a class that is only available to it, making it impossible for other objects to alter it. They are instead restricted to a small number of public functions and methods. By removing accidental information destruction, this data-hiding approach improves program security. In essence, the goal is to safeguard the object's internal state by restricting direct access to it and providing a safe and regulated method of interaction. (Gillis, 2021)

+ **Inheritance** : Classes have the ability to reuse code from other classes. Developers can create subclasses and links between objects to reuse code while maintaining a clear hierarchy. This OOP feature encourages a more thorough data analysis, reduces the amount of time needed for development, and increases accuracy. (Gillis, 2021)

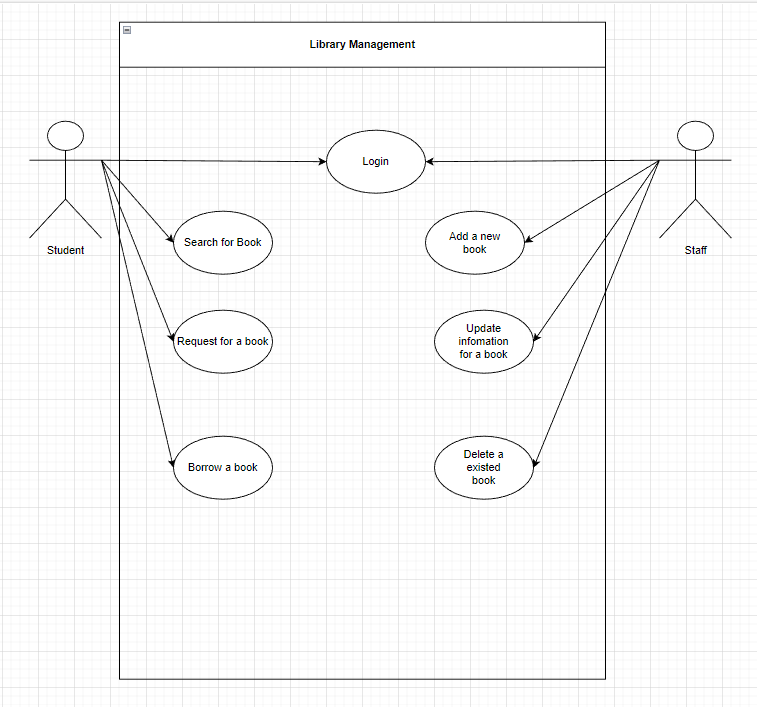
+ **Polymorphism** : The forms and sizes of objects vary depending on and they have been created to share behaviors. To reduce the need for duplicating code, the program will determine whether usage or meaning is necessary for each execution of an object from a parent class. A child class is then created, extending the parent class's functionality. Polymorphism allows different types of objects to flow through the same interface. (Gillis, 2021)

# III.OOP scenario (P1,P2)

## III.1.Scenario

- OOP scenario is : Create a management system for libraries. The system is made up of a number of classes, including Books, Users, BorrowingsBooks, and others. Title, author, publication date, and availability information about the books in the library are all stored in the Books class. Members class data includes name, address, contact information.The Borrowings class keeps track of the books that members have borrowed, along with the due date and the condition of each book.

## III.2.Usecase Diagram



- The use case diagram for the Library Management System scenario illustrates the interactions between the Student and Library actors. The Student class represents customers of libraries and provides information about their contact information, book requests, and book borrowings. The Library class serves as a representation of the library and includes data about the books and requests. The personnel class, which represents library personnel and enables them to log in and handle the books, is introduced as an extra class. The functionalities that students and staff can carry out, such as book searching, book requests, book borrowing, and collection management, are described in the use case diagram. The ability to create, update, and delete books is specifically available to members of the Staff class.

## III.3.Class Diagram

Diagram

Description automatically generated

- Here I have class Book has fields(id, title, authorName, languages, year, quantity, available) and methods(getId(), getTitle(), getAuthorName(), getLanguages(), getYear(), getQuantity(), isAvailable). It will be extended by class ClassBook(fields: grade, subjects), class ReferenceBook(field: major) and class OtherBook(field: type). Additionally, I have class Library to store a list of books(field: books(List<Book>)).

- Class User is an abstract class, it’s extended by class Student(fields: role, className) and class Teacher(fields: role, yearsOfExperience).

- Class BorrowingBooks has fields(user, books) and methods(getUser(), getBooks(), addBook(), removeBook()), use dependencies from class Book and class User. It is used for one user at one time.

## III.4. Implementation

Text

Description automatically generated

- class Book: has these fields: id, title, authorName, languages, year, quantity, available

+ These field has get and set method: title, authorName, languages, available

+ Method Id:

+) Get: return id value

+) Set: check if value is greater than 0 or not, if it’s greater, assign that value to id

Text

Description automatically generated

+ Method Year:

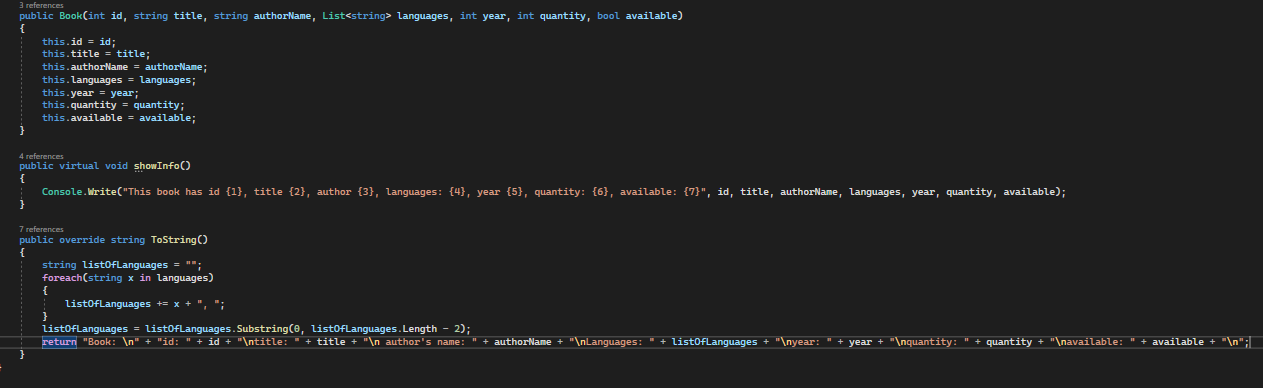
+) Get: return year value

+) Set: check if value is greater than 0 or not, if it’s greater, assign that value to year

+ Method Quantity:

+) Get: return quantity value

+) Set: check if value is greater than 0 or not, if it’s greater, assign that value to quantity



+ All Args Constructor: assign each variable to value

+ Virtual void method showInfo(): show all info

- Class ClassBook (extend class Book):

Text

Description automatically generated

+ It has fields: grade and subjects

+ Method Grade:  
+) Get: return grade value

+) Set: check if value is greater than 0 or not, if it’s greater, assign that value to grade

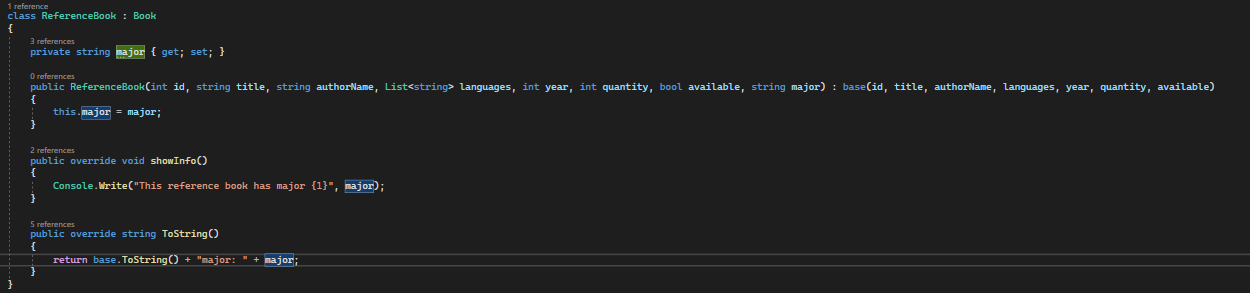
Graphical user interface, text

Description automatically generated

+ All Args Constructor: assign each variable to value

+ Override void method showInfo() from class Book: show all additional info of this classbook

- Class ReferenceBook (extend class Book):



+ It has field major: get and set method

+ All Args Constructor: assign each variable to value

+ Override void method showInfo() from class Book: show all additional info of this reference book

- Class OtherBook (extend class Book):

Text

Description automatically generated

+ It has field type: get and set method

+ All Args Constructor: assign each variable to value

+ Override void method showInfo() from class Book: show all additional info of this other book

- class User:

Text

Description automatically generated

+ It’s an abstract class

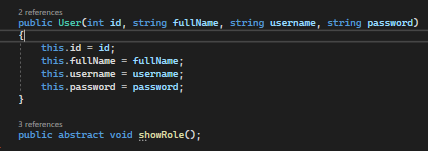
+ It has fields: id, fullName, username, password

+ These fields has get and set method : fullName, username, password.

+ Method Id:

+) Get: return id value

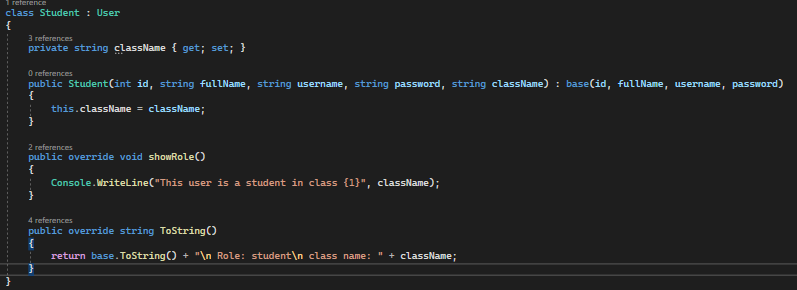
+) Set: check if value is greater than 0 or not, if it’s greater, assign that value to id



+ All Args Constructor: assign each variable to value

+ Abstract method void showRole()

- class Student (extend class User):



+ It has field: className(get and set method)

+ All Args Constructor: assign each variable to value

+ Overriding the Abstract method void showRole() from class User: show this is a student and show className

+ Method toString()

- class Teacher (extend class User):

Text

Description automatically generated

+ It has field yearsOfExperience (get and set method)

+ All Args Constructor: assign each variable to value

+ Method YearsOfExperience

+) Get: return yearsOfExperience value

+) Set: check if value is greater than 0 or not, if it’s greater, assign that value to yearsOfExperience

+ Method toString()

+ Overriding the Abstract method void showRole() from class User: show this is a teacher and show years of experience

- class BorrowingBooks:

Text

Description automatically generated

+ It has fields: user, books

+ All Args Constructor: assign each variable to value

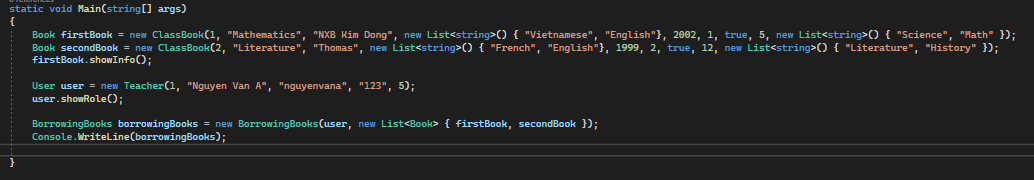
+ Method addBook(Book book): add more book to the list of books

+ Method removeBook(Book book): check if this list has the book need to remove, if it has, remove that book

+ Method toString(): show all info

- Main Overview:

+ Overview:



+ Output:

Text

Description automatically generated

# IV.Conclusion

- My system includes class Book, it is being extended by class ClassBook, class ReferenceBook, class OtherBook (additional information needed), includes abstract class User, which is being extended by class Student, class Teacher(additional information needed). One final class is class BorrowingBooks has 1 user and list of books, indicate the borrowing books session in the library for easier management.

=> I think my system is suitable enough for manage the borrowing books session in the library nowadays. It has 2 main class to be extended to indicate the difference and additional information through these classes, and has a class to store information about these action.

# V.References

[1] Gillis, A. (2021). *What is Object-Oriented Programming (OOP)?* [online] TechTarget. Available at: <https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP.(Accessed> 7th April 7, 2023)

‌