**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** |  | **Student ID** |  |
| **Class** | GCH1102 | **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** |  |

**Grading grid**

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

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

Contents

[1. Introduction 3](#_Toc144409992)

[2. OOP general concepts 3](#_Toc144409993)

[2.1. What is OOP 3](#_Toc144409994)

[2.2. Structure 3](#_Toc144409995)

[2.3. APIE 5](#_Toc144409996)

[2.3.1. Abstraction 5](#_Toc144409997)

[2.3.2. Polymorphism 6](#_Toc144409998)

[2.3.3. Inheritance 7](#_Toc144409999)

[2.3.4. Encapsulation 7](#_Toc144410000)

[2.4. Class relationships 9](#_Toc144410001)

[2.4.1. Realization 9](#_Toc144410002)

[2.4.2. Inheritance 10](#_Toc144410003)

[2.4.2. Dependency 11](#_Toc144410004)

[2.4.3. Association 11](#_Toc144410005)

[2.4.4. Aggregation 12](#_Toc144410006)

[2.4.5. Composition 13](#_Toc144410007)

[3. OOP scenario 14](#_Toc144410008)

[3.1. Scenario 14](#_Toc144410009)

[3.2. Use case diagram 16](#_Toc144410010)

[3.3. Class diagram 18](#_Toc144410011)

[4. Conclusion 21](#_Toc144410012)

[References 21](#_Toc144410013)

# 1. Introduction

In this document, I will introduce the concept of object-oriented programming using the C# programming language. This will cover design patterns, their definitions and intended applications, as well as scenarios illustrating their usage. Additionally, class diagrams and user diagrams will be provided to enhance understanding.

# 2. OOP general concepts

## 2.1. What is OOP

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of “objects”, which can contain data and code. The data is in the form of fields (often known as attributes or properties), and the code is in the form of procedures (often known as methods). In OOP, computer programs are designed by making them out of objects that interact with one another. Any contemporary Object-Oriented Programming language is built around classes. A class must be created in OOP languages in order to represent data. A class is the blueprint for an object and includes functions to manipulate the data as well as variables to store it. A class is merely a logical representation of data because it will not take up any memory. Simply use the term "class" followed by the class name to create a class (Nyakundi, 2023)

## 2.2. Structure

In object-oriented programming (OOP), a class is a pattern or blueprint to define the specific objects that we will create in the program. The class acts as a "template" for creating objects, defining how they work, and managing their data and behavior. Among the important ideas about classes are:

* A class can have subclasses that can inherit all or some of the characteristics of the class. In relation to each subclass, the class becomes the superclass.
* Subclasses can also define their own methods and variables that are not part of their superclass.
* The structure of a class and its subclasses is called the class hierarchy.

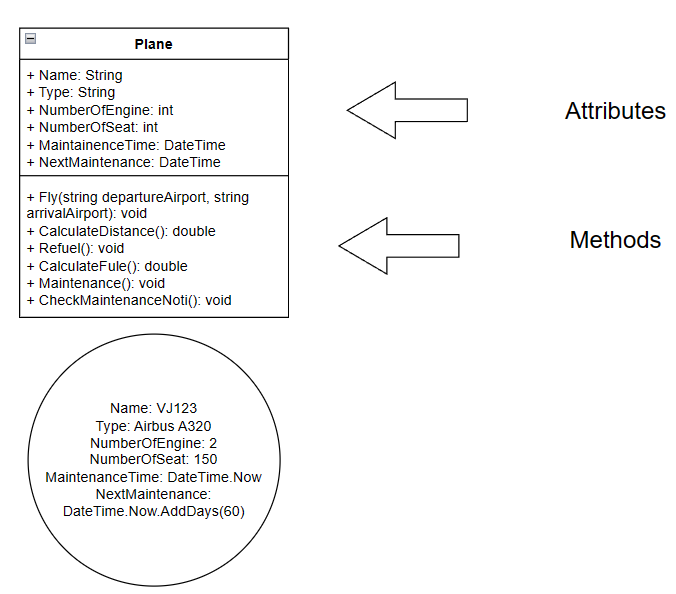


Figure : Example of class in OOP

This class diagram represents an plane consisting of the following components (Attributes):

* Name: is the name of plane with data type string
* Type: is plane’s type (passenger plane, cargo plane, combat aircrafts,…) with data type is string
* NumberOfEngine: the number of plane engines used with data type is int
* NumberOfSeat: the number of plane seat with data type is int
* MaintainceTime: Represents the aircraft's most recent maintenance date wih data type is DateTime

Plane’s operation (Methods):

* Fly(string departureAirport, string arrivalAirport): take off and land at airports
* Refuel(): appropriate refueling for each flight
* Maintenance(): Maintenance and set maintenance date

## 2.3. APIE

APIE stands for the first 4 words of the 4 OOP properties. Full names are Abstraction, Polymorphism, Inheritance, Encapsulation.

### 2.3.1. Abstraction

This is the ability of the program to ignore or not pay attention to some aspects of the information that it is directly working on, meaning it is able to focus on the essential core. Each object serves as an "animal" that can internally complete tasks, report, change its state, and communicate with other objects without needing to tell the object how. The object can perform operations. This property is often called data abstraction. Abstraction is also demonstrated by the fact that an initial object may have some characteristics common to many other objects such as its extension, but this initial object itself may not have implementations. . This abstraction is often defined in a concept called an abstract class or abstract base class. (Pankaj, 2022)

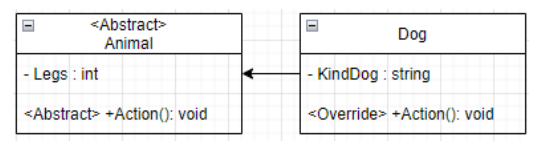


Figure : Abstraction

Super class is an Animal that has an abstract method with no content. The subclass is dog, inheriting and overriding the abstract part of the superclass. If the Sub class does not override Animal's abstract method, an error will occur

### 2.3.2. Polymorphism

Expressed through sending messages. Sending these messages can be compared to calling the internal functions of an object. The methods used to respond to a message will respond differently depending on the object to which the message is sent. The programmer can define a property (for example, by method names) for a series of close objects, but when implementing the same name, the implementation of each object will be automatic. occur correspondingly according to the characteristics of each object without confusion (Thorben, 2021)

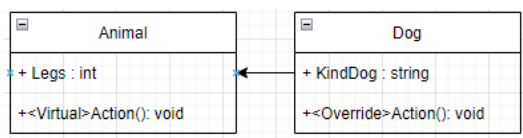


Figure : Polymorphism

In the above diagram, it can be seen in the Animal subclass that there is an Action() method and when the subclass overrides Action(), we will have 2 ways to implement the Action() method. In order for the Dog class to override the Action() method, the superclass's method declaration must be virtual, just like granting override permission to subclasses.

### 2.3.3. Inheritance

This property allows an object to have properties that another object already has through inheritance. This allows objects to share or extend existing properties without having to redefine them. However, not all object-oriented languages ​​have this property (Singh, 2020)

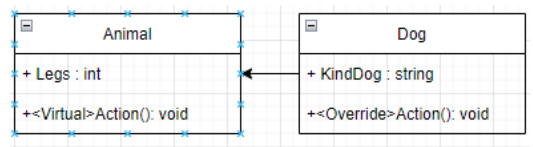


Figure : Inhertance

In the diagram above, it can be seen that the parent class is Animal with a Legs property and a method.When the inherited subclass is Dog, it will not have to re-declare the above variables and methods, but will always use Animal's variables and methods.

### 2.3.4. Encapsulation

One of the main benefits of using OOP instead of procedural programming is keeping your codebase tidy and therefore easier to maintain and with less code smell. Encapsulation is what drives this by creating classes for each entity and responsibility in the system. These classes are used to store and manage everything for the entity in question called access modifier, every time a variable or method is declared, the access scope is declared (if not declared, the default is public) after declaring the scope, other classes must comply with the scope and access rights (Khanna, 2021)



Figure : Public type

The example above is about public declaration, when this type of newspaper will be able to access anywhere



Figure : Private type

The example above is about the declaration of private variables, when this type of newspaper will only be accessible in this class, other classes will not be able to access



Figure : Protected type

The example above is about the protected variable declaration when this type of newspaper will only be able to access in this class and the inherits from this class.



Figure ; Internal type

The example above is about the Internal variable declaration when this type of newspaper will only be able to access in this packet and these non -packet classes will not be able to pursue.



Figure : protected internal type

The example above is about the declaration of protected internal, when this type of newspaper will only be able to access in this packet and those classes inherit each other in the same packet to be able to access

## 2.4. Class relationships

### 2.4.1. Realization

A relationship between a classifier that acts as a contract and a classifier that acts as an implementation. In other words:

The relationship between a class implementing an interface is called a realization relationship, represented by a dashed line with a triangular arrow pointing to the interface. (TiEo, 2020)

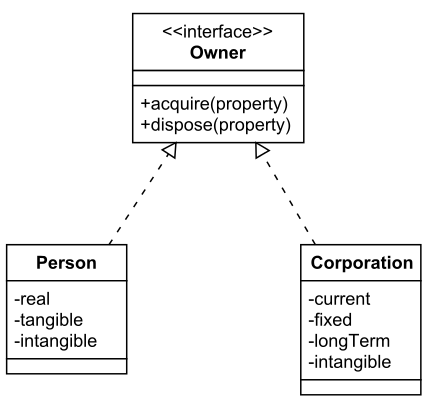


Figure :realization

A class implements an interface. For example, the Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways

### 2.4.2. Inheritance

A concrete object will inherit the properties and methods of a general object. (TiEo, 2020)

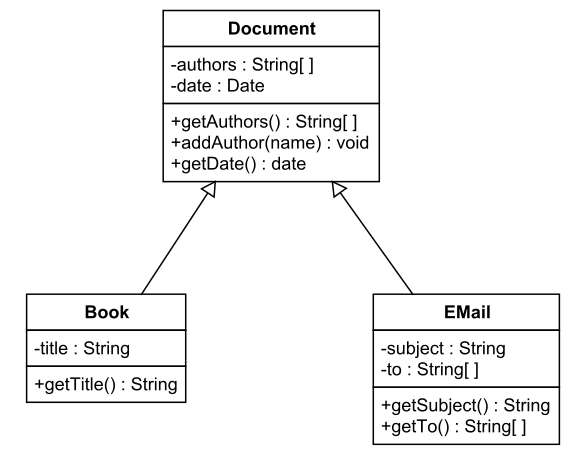


Figure : inhertance

Explain: A class extends another class. For example, the Book class might extend the Document class, which also might include the Email class. The Book and Email classes inherit the fields and methods of the Document class (possibly modifying the methods), but might add additional fields and methods

### 2.4.2. Dependency

A relationship between two elements in the model in which a change in one element (independent element) can cause a change in the other element (dependent element). (TiEo, 2020)

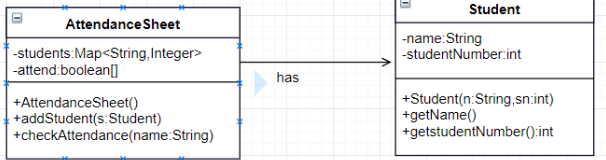


Figure : dependency

Dependency also has a number of other manifestations, often using the following stereotypes:

<<use>> : indicates that the semantics of the root class (arrow) depends on the top class (arrow). Especially in the case where the root class uses the top class as a parameter in some of its methods

<<permit>> : indicates that the root class has special access rights to the top class (for example, access to private operations).

<<refine>> : indicates that the base layer is at a higher level of refinement than the base layer. For example, a class created in the design stage must be refined with the same class created in the analysis stage

### 2.4.3. Association

Between two objects of two classes there is a pairing (husband - wife, teacher - student, customer - invoice ...) . A set of connections of the same type (same meaning) between objects of two classes forms an association, a relationship between two sets (2 classes).

Each relationship between two classes has a role, role is the role name of the relationship: for example: of, for, has, linked to, exchanged for, .... (usually the role name is accompanied by an arrow to indicate the direction of the applicable relationship from which class to which class) (TiEo, 2020)



Figure : association

Programmatically, properties can be stored as single variables, array variables, or pointer variables. With or without the digital version is fine. With or without arrow is fine. If there is a 1-way arrow, indicating the direction of the object of this class, only calling the object of the other class, not the opposite direction. If there is no arrow, it is equivalent to a 2-way arrow, or the direction is not important. (TiEo, 2020)

### 2.4.4. Aggregation



Figure : aggregation (TiEo, 2020)

ClassA's object contains (in its properties) ClassB's object

If ClassA's ObjectX is destroyed, ClassB's ObjectY (inside ObjectX) can still exist

Also called shared-aggregation. A form of association in which one element contains other elements.

Meaning : also known as : Whole A – Part B . That is, A is made up of many Bs combined, and B can be created independently, there is no need to create A, B can belong to a whole other than A.

### 2.4.5. Composition

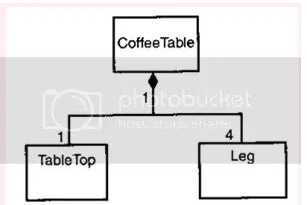


Figure : composition (TiEo, 2020)

Meaning: also known as Whole A – Part B. That is, A is made up of many Bs combined, but B cannot stand alone, B is only part of A and cannot be in another Whole.

Identified that ClassA and ClassB have an Association relationship with each other

Define more clearly:

ClassA's object contains (in its properties) ClassB's object

If ClassA's ObjectX is destroyed, then ClassB's ObjectY (within ObjectX) can no longer exist

# 3. OOP scenario

## 3.1. Scenario

ABC Furniture Company has an idea for an application that allows users to design their own custom table. The company will provide users with the highest quality available components, a wide variety of options, and countless accessories. The components of a table include:

* Tabletop (user-defined dimensions)
  + Triangle
  + Square
  + Circle
* Table legs (user-defined height)
  + K-shaped legs
  + U-shaped legs
  + Single center leg
  + Four legs
* Material:
  + Oak wood
  + Mahogany wood
  + Sandalwood
  + Ebony wood
  + Engineered wood
* Color:
  + White
  + Black
  + Brown
  + Gray
  + Natural wood color
* Accessories (customers can choose multiple or none)
  + Hanging hooks
  + Cup holder
  + Tablecloth
  + Glass top

All these components have different price levels, so customers should carefully consider their choices. Once all the components of the table are selected, the production workshop will assemble and complete the product. At this point, all the customer needs to do is wait for the table to be delivered and make the payment.

## 3.2. Use case diagram

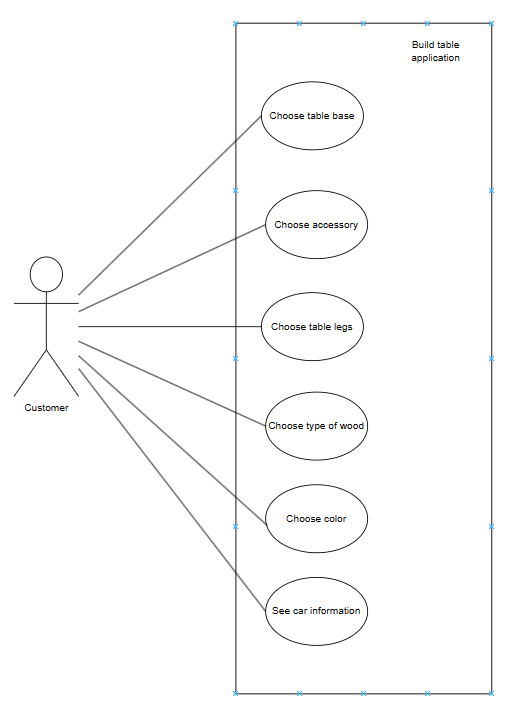


Figure : use case diagram

Through the scenario, the user interacts with the table elements. There are 6 parts for customers to choose from, each option has smaller options, which may or may not be required depending on the application requirements and customer needs. Options include: Choose table base, Choose accessory, Choose table legs, Choose type of wood, Choose paint color, See car information

|  |  |
| --- | --- |
| Description | Choose type of wood |
| Actors | Customer |
| Precondition | Customer wants to choose wich type of wood will be used for the wooden table |
| Postcondition | Customer has chosen 1 type of wood from different option |
| Flow | 1. Customer sees different options of wood types.  2. Customer chooses the wood type they want  (Ironwood, Hardwood, Redwood, or Oakwood).  3. Customer confirms the selection. |

|  |  |
| --- | --- |
| Description | Choose table base |
| Actors | Customer |
| Precondition | Customer want to choose which shape they want their table base to have |
| Postcondition | Customer has decided which base they want the table to have |
| Flow | 1. Customer choose a round base, they will have to specify the length of the diameter  2. Customer choose 1 typr of table base (Round, square, long base)  3. Customer confirms the selection |
| Alternative flow 1 | If customer choose a square base, they will have to specify the length of 4 side |
| Alternative Flow 2 | If customer chooses a square base, they will  have to specify the length of 4 sides |
| Alternative Flow 3 | If customer chooses a long base, they will have  to specify the length and width. |

|  |  |
| --- | --- |
| Descrption | Choose table legs |
| Actor | Customer |
| Precondition | Customer wants to choose a type of table legs |
| Postcondition | Customer has decided which type of table legs will be used |
| Flow | 1. Customer sees different option for table legs  2. Customer choose the table legs they wants  3. Customer choose the height of table legs  4. Customer choose how many table legs they want  5. Customer confirms the selection |

|  |  |
| --- | --- |
| Description | Choose paint color |
| Actors | Customer |
| Preconditions | Customer wants to choose a color of paint |
| Postcondition | Customer has decided which color the table will have |
| Flow | 1. Customer sees different color of paint.  2. Customer chooses the paint color they want.  3. Customer confirms the selection. |
| Alternative Flow | Customer choose no paint color |

|  |  |
| --- | --- |
| Description | Choose accessory |
| Actor | Customer |
| Precondition | Customer wants to choose accessories. |
| Postconditions | Customer has decided which accessories the wooden table will have. |
| Flow | 1. Customer sees different options of accessories.  2. Customer chooses an accessory.  3. Customer confirms the selection. |
| Alternative flow | Customer chooses no accessory. |

### 3.3. Class diagram

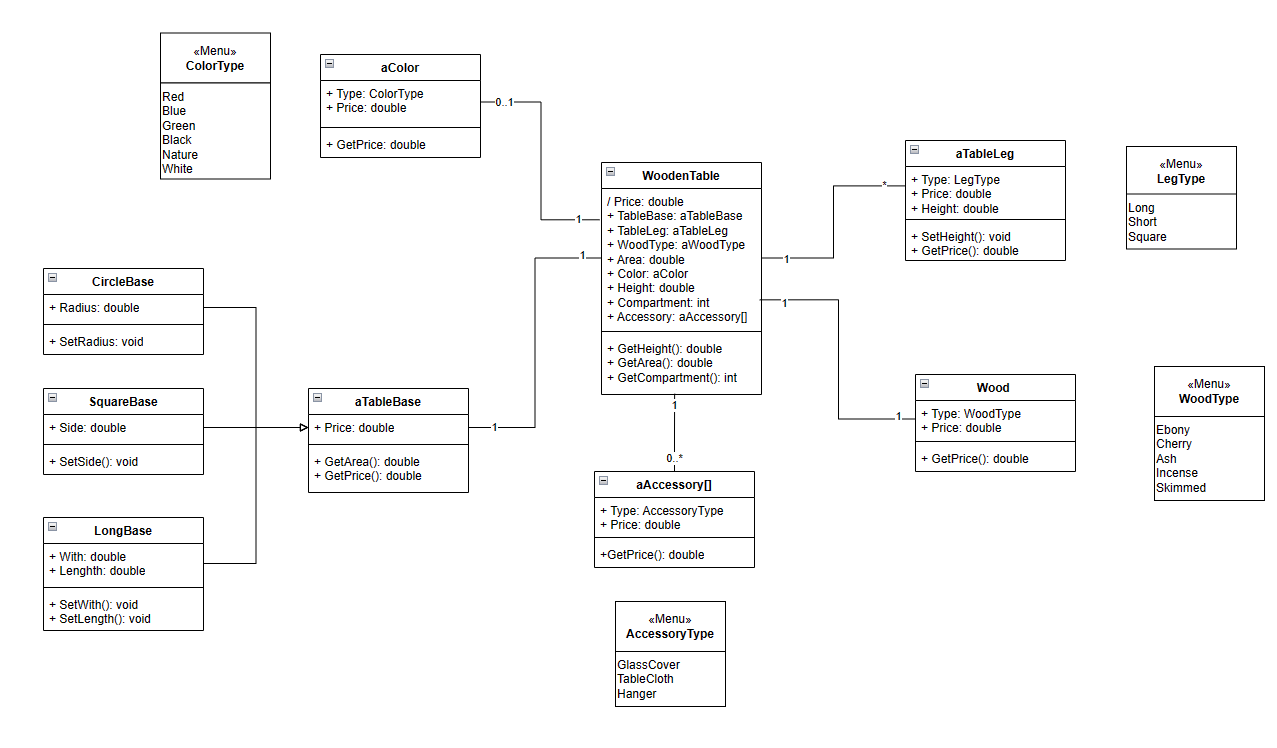


Figure : class diagram

Explain:

This class diagram represents the 3.1 scenario. A class WoodenTable and a total of 5 other classes and 1 interface (whose property and methods are implemented in 3 other classes) represent all the parts and attributes of the wooden table and the table itself. The WoodenTable has relation to all these classes. First, every class has the property Price in double. This represent its cost, and can be calculated with GetPrice() method in each class. A wooden table has 1 table base, so the relation to interface aTableBase is one-to-one. The interface aTableBase has the mentioned property Price, and methods GetArea() and GetPrice(). GetArea(), which return a double, calculates the area of the table base, and because there are diverse types of table bases, it will be overridden when implemented in other classes. For example, the CircleBase class, which implements the interface, calculates the area with the overridden GetArea() based on the Radius. To set the Radius, SetRadius() lets customers choose their wanted size. Similarly, SquareBase also overrides GetArea() to calculate the area based on measurement of Sides. SetSide() gives the customers the ability to choose the length of such sides. Finally, the LongBase is a rectangular base, so to calculate its area, customer must choose Width and Length with SetWidth() and SetLength(), respectively. The price of table base is determined by the calculated area.

A wooden table has multiple legs, so the relationship between WoodenTable and aTableLeg is one-to-many. aTableLeg has many types of legs, represented by the enumeration LegType that has the following predefined constants of Long, Short, and Square. Same as every other class, aTableLeg has Price that is different for each type of leg and how many legs there are. Another property of aTableLeg is Height, and customers can freely choose the height of the legs with SetHeight(). The GetPrice() will calculate the price based on the type of table legs, the height, and the number of table legs.

Customers can choose the material for their wooden table. One WoodenTable can only have one Wood (wood type). The several Types are represented by enumeration WoodType, which has the following: Ironwood, Hardwood, Redwood, and Oakwood. The Price will therefore differ for each type of wood. The wooden table can be painted (optional) with Brown, Black, or White color. These colors are the constants of ColorType, which is a property of PaintColor. WoodenTable has one-to-zero or one relationship with PaintColor. The types of colors are differentiated because the Price varies. Though there are no calculation for the price, GetPrice() will still return the price of painting the wooden table. As an option, customers can order some accessories for their wooden table. These include Galass Cover, Table Cloth, Hanger, which are the predefined constants of AccessoryType, a property of Accessory. Each has different cost, represented by Price. The price can be accessed with GetPrice().

The Area property in WoodenTable class has the same value as the area of chosen table base. The Height property is the sum of the table leg height. GetArea() and GetHeight() will be able to show the former and latter values. The property Price in WoodenTable is the sum of all components and can be calculated with GetPrice() method

# 4. Conclusion

After doing this exercise, I have a very clear understanding of object-oriented programming, how to analyze a real-life problem and convert it into a diagram. This makes the problem easier to understand and implement. I have presented a problem and in the next exercise I will turn it into code so that I can interact as desired.

# References

Khanna, M., 2021. *Scount APM.* [Online]   
Available at: https://scoutapm.com/blog/what-is-encapsulation

Nyakundi, H., 2023. *freeCodeCamp.* [Online]   
Available at: https://www.freecodecamp.org/news/what-is-object-oriented-programming/

Pankaj, 2022. *DigitalOcean.* [Online]   
Available at: https://www.digitalocean.com/community/tutorials/what-is-abstraction-in-oops

Singh, H., 2020. *AnalyticsVidhya.* [Online]   
Available at: https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/

Thorben, 2021. *Stackify.* [Online]   
Available at: https://stackify.com/oop-concept-polymorphism/

TiEo, 2020. *VietLinhTinh.* [Online]   
Available at: https://votanlanh.wordpress.com/2017/08/08/mot-so-quan-he-giua-cac-class-trong-uml/