Name: Luong Trac Duc Anh

ID: 103488117

# Question 1

*<The answer to this question, including the UML diagram and code files will be submitted separately>*

# Question 2

## Abstraction

In OOP design, programs are usually made of big code bases, with items communicating with each other. Therefore, handling the complexity of maintaining and adding changes to many code files is not an easy task. The solution to this is using abstraction. The concept is basically to only review the essential attributes of an object and hide other redundant information.

For reference, abstraction has been implemented in question 1, specifically in the **AddResource()** method. In the **Program** class, to add a new item to the library, we only need to call **library.AddResource(…)** instead of having to worry about the **_resources** field that adds new items to the list **List<LibraryResource>**. The property **OnLoan** is another instance of the abstraction concept. To change an item's availability in the library, we only need to call **item.OnLoan** and set it to either **true** or **false**, without worrying about getting and setting the value of the **_onLoan** field.

## Polymorphism

Let's say that in a program, we have a parent class and some children classes that inherit the properties that are contained in their parent class. If we have a method in the parent that we want to implement in the child without mixing types, we will need a concept called polymorphism. Polymorphism provides a solution to where a method can be used in the children classes exactly like their parent class, and every child will be able to keep their unique methods. For this to work, we will need to define an interface so that it can be reused. The parent will have some methods, and its children will implement their own versions of those methods.

For reference, polymorphism has been implemented in question 1, specifically in the **AddResource()** and **HasResource()** methods. Properties like **Name** and **Author** can be used for both **Book** and **Game** through the **LibraryResource** interface, even though **Book** and **Game** are two different classes.