Object Oriented Principles

Written by: Marella Morad – Student ID: 103076428

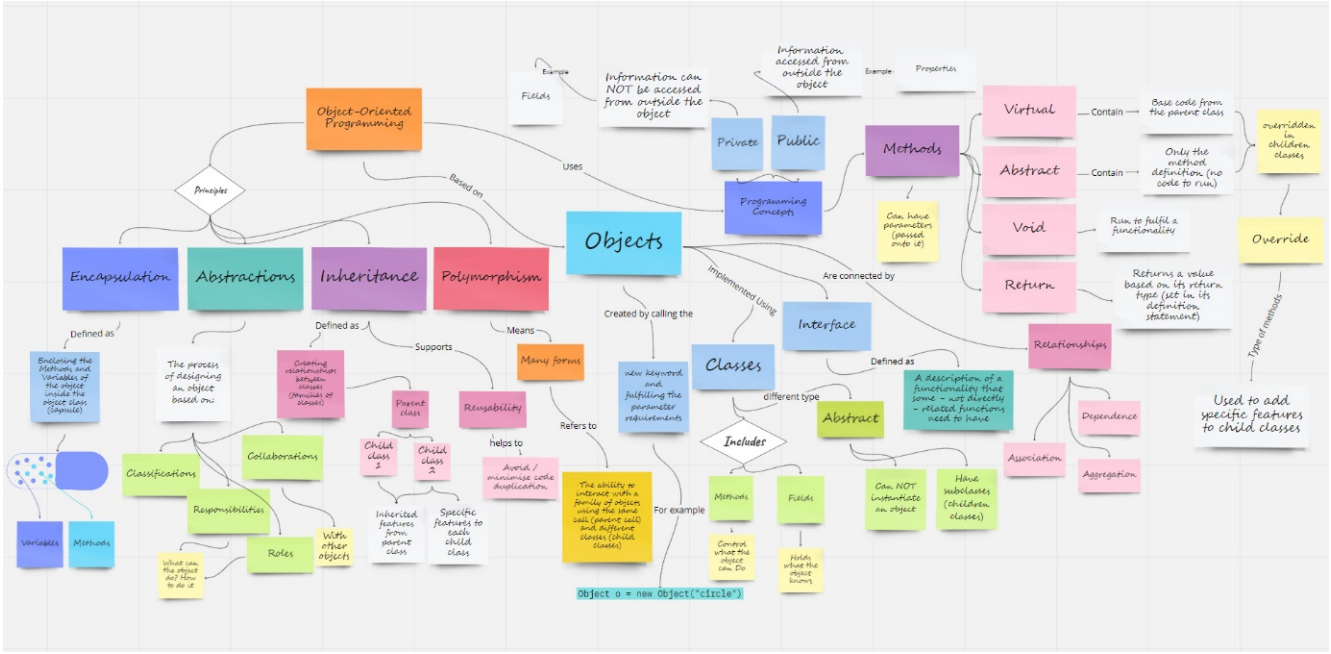

Table of Contents

1 Introduction:

Object Oriented Programming is a style of programming which organises a program/software into objects rather than separating the code into functions. The benefit of OOP becomes clearer with large and more complex designs. The use of objects is particularly useful to decrease the timeliness of a program and increase its efficiency (minimum code duplication)

2 OOP Principles:

Object Oriented Programming is based on four main principles.

2.1 Encapsulation

One of the key principles about OOP is **encapsulation**. This is basically the idea of enclosing the methods and fields of the object inside the object class – as a capsule – where the capsule cover represents the curly brackets at the start and end of the object (see Figure 1).

```
public class Message
{
        private string text;
        public Message(string txt)
        {
            text = txt;
        }
        public void Print()
        {
            Console.WriteLine(text);
        }
}
```

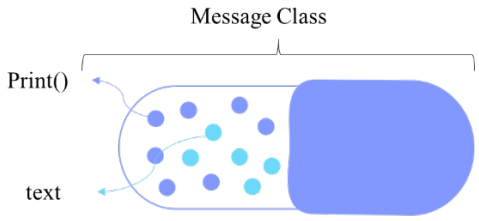

Figure 1 Encapsulation Principle explained with an example

2.2 Abstractions

Another key principle in OOP is **abstractions**. This is defined as the process of designing an object in relations to its Roles, Responsibilities, Classifications and Collaborations with other objects. Basically, defining what the object can do and how it can do it (with the implementation of methods and some relationships with other object which will be discussed later).

2.2.1 Cohesion

Cohesion is also an important concept in OOP. This concept is responsible for defining the objective of an object, or in other words, making sure that the object does not have a lot of different roles tied to it and is only responsible for the one single goal that it was made for. For example, in the shape drawer activity, although the Circle and Rectangle classes are both shapes, but we can argue that the goal of a Circle class is to draw a circle only not a rectangle, not a line.

2.3 Inheritance

The third main principle in OOP is **inheritance**. As the word suggests, a child inherits something from their parents. The same is applied in OOP. Inheritance is used to create families of classes. These classes will have a parent class and children's classes which inherit functionalities from their parent class. For example, from SwinAdventure, there are two types of commands that the game requires. See, two types of commands (command is the parent class here with functionalities that it passes on to its children's classes), see Figure 2 for code example.

```
namespace SwinAdventure
{
    public abstract class Command : IdentifiableObject
    {
        public Command(string[] ids) : base(ids)
        {

        }

        public abstract string Execute(Player p, string[] text);
    }
}
```

```
namespace SwinAdventure
{
    public class LookCommand : Command
    {
        public LookCommand() : base(new string[] { "look" })
        {

        }

        public override string Execute(Player p, string[] text)
        {
            if(text.Length !=1 && text.Length != 3 && text.Length != 5)
            {
                return "I don't know how to look like that";
            }
```

Figure 2 Inheritance LookCommand inherits Execute functionality from Command class

2.3.1 Reusability

The use of inheritance increases the reusability of the program as it groups objects into family which can call each other instead of having to run all the individual classes to get to a functionality (skips through the line).

2.4 Polymorphism

The final major principle of OOP is **polymorphism**. Polymorphism translates into "many forms", which refers to the ability to interact with a family of objects using the same call (parent call) and specifying the type (children class) in the call (See Figure 3).

```
//Polymorphism
//Creating a ResigsterShape method in the (parent class) Shape
//and registering the different types of shapes (child classes)
Shape.RegisterShape("Rectangle", typeof(MyRectangle));
Shape.RegisterShape("Circle", typeof(MyCircle));
Shape.RegisterShape("Line", typeof(MyLine));
```

Figure 3 Polymorphism of the Shape family

3 Objects

Now that we've overview OOP's principles, we can move to talking about OOP's backbone – Objects. At the end of the day OOP is a style of programming and so it obeys the basic programming principles.

3.1 Instantiation

Objects are instantiated only by calling the new keyword and fulfilling their constructor's parameter requirements as shown in Figure 4.

```
Item goggles = new Item(new string[] { "goggles" }, "IR goggles", "These are the
best night vision goggles");
```

Figure 4 Instantiating an Item by calling the new keyword with its constructor

3.2 Class

A class is how the object is constructed. It contains what the object knows and what it can do (fields and methods respectively). There are different types of classes, one of which is an Abstract class. Abstract classes are different as that they cannot instantiate an object, however, they have subclasses which can. If you look back at Figure 3, you can see that Shape class is an abstract class, so it only contains the information that it will pass on to its children classes.

3.3 Interface

Interface is only a description of a functionality that some – not directly related – classes need to have. For example the `IHaveInventory` interface from SwinAdventure, is implemented in both Player and Location classes to add the Locate() functionalities to both of them.

3.4 Relationships

After objects have been created, now is the time to define the relationships between them in order to have a fully functional program at the end. There are three types of relationships in OOP and these relationships are best described using a UML diagram (see Figure 5).

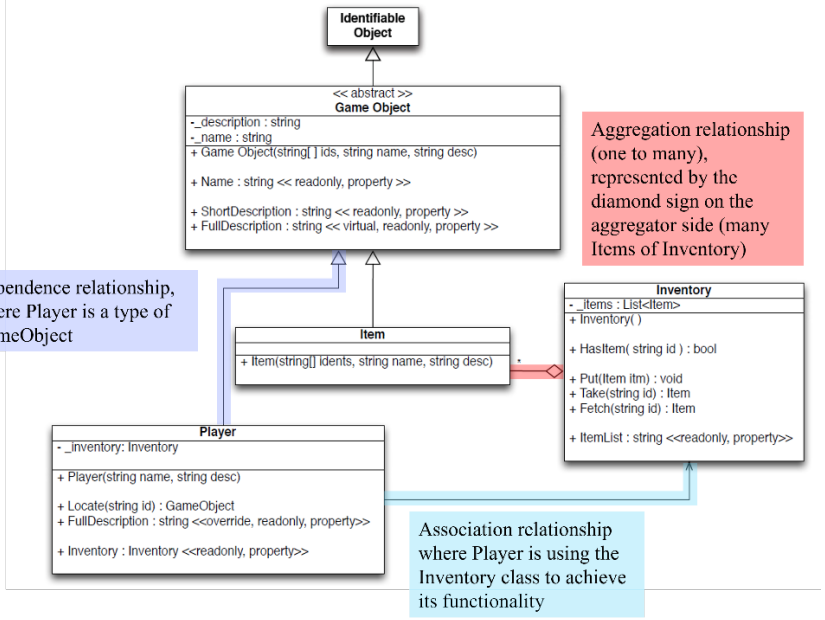

Figure 5 UML diagram with the three types of relationships

4 Programming Concepts / Terminology

4.1 Methods

There are a few types of methods used in OOP and all of them can have parameters passed onto them. A summary of the common types is in Table 1.

Table 1 Summary of Method Types used in OOP

Method Type	Definitions	Example
Void	Basic type of methods, does not return a value	`public void Draw()`
Return	Must have a return statement which returns a value based on the specified return type (in the method definition)	`static string GetKey(Type type)`
Abstract	Contains only the method definitions with no code to run	`public abstract string Execute(Player p, string[] text);`
Virtual	Contains the base code from the parent class	`public virtual void SaveTo(StreamWriter writer)`
Override	Used to override abstract and virtual methods in children's classes to add specific features.	`public override string Execute(Player p, string[] text)`

4.2 Scope

The last thing to talk about in this brief introduction to OOP is the scope of an element of a program.

Table 2 Scopes in Programming (used in OOP)

Type	**Scope**
Public	Can be accessed from anywhere in the program
Private	Can only be accessed from inside the object
Protected	Can be accessed from children classes which inherit it from the parent class
Static	Is shared through the entire class scope

Please refer to 7.2C Concept map which also outlines the relationships between the discussed concepts in a graphical manner.

5 References:

danijar 2012, “What is the difference between protected and private?,” *Stack Overflow*, viewed 21 September, 2021, <https://stackoverflow.com/questions/12784083/what-is-the-difference-between-protected-and-private>.

Francesco Lelli 2019, “Object Oriented Programming: A curated set of resources,” *Francesco Lelli*, viewed 21 September, 2021, <https://francescolelli.info/tutorial/object-oriented-programming-a-curated-set-of-resources/>.

Pavan Podila 2016, “10 Applications of Object Oriented Programming,” *Quickstart.com*, QuickStart, viewed 21 September, 2021, <https://www.quickstart.com/blog/10-applications-of-object-oriented-programming/>.

Gillis, AS & Lewis, S 2017, “object-oriented programming (OOP),” *SearchAppArchitecture*, TechTarget, viewed 21 September, 2021, <https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP>.

Object Oriented Programming Lectures 2021, “Lecture Slides and Materials: 2021-HS2-COS20007-Object Oriented Programming,” *Instructure.com*, viewed 21 September, 2021, <https://swinburne.instructure.com/courses/38543/pages/lecture-slides-and-materials?module_item_id=2306990>.