COS20007 - Object Oriented Programming

6.2P - Key Object Oriented Concepts and Self-Reflection

Student name: Nguyen Duc Manh
ID: 105547489

OOP Report

Name: Nguyen Duc Manh

ID: SWH02701

Object-Oriented Programming is a programing model that helps developers organize their work around objects rather than functions and logics. This paradigm can make it more efficient and easier to scale up when it comes to big and complex coding projects.

Throughout the COS20007 course, I've learned about the four core concepts of OOP are: **Encapsulation**, **Abstraction**, **Inheritance** and **Polymorphism**.

I. Encapsulation

Definition: Encapsulation is the concept of wrapping data (known as "fields") and the method into a "class". This restrics direct access from outside the class, giving controlled access to important data.

The main goal: The main goal of encapsulation is to control the access from the outside to the class's internal to make sure: Data integrity (prevent invalid inputs), Modularity (independent and scalable), Security (exposing only what's necessary).

Key component:

1. Access Modifiers: control field's visibility: private (-) and public (+) / also there are: protected and internal.
2. Getters and Setters: changing or view the value of a field indirectly.
3. Hide Implementation Details: change the internal implementation of a class without affecting other classes.

How I implemented this concept in my code:

```
public string FirstId
{
    get
    {
        if( _identifiers.Count == 0)
        {
            return "";
        } else { return _identifiers.First(); }
    }
}
```

```
//fields
private List<string> _identifiers;
string _myStudentID = "7489";
```

II. Abstraction

Definition: Abstraction focuses on hiding the implementation details and only show out the features (or more likely "what it does") of an object.

Main goal: To define a general picture of the behavior of that object and letting specific classes provide the actual implementation behind.

Purpose:

- Simplify interaction.
- Separate interface.
- Increase code readability, reusability, and maintainability.
- Enable extensibility via polymorphism.

How it is used: Through abstract classes, Interfaces.

How I implemented this concept in my code:

```
namespace SwinAdventure
{
    public abstract class GameObject : IdenObj
    {
```

III. Inheritance

Definition: Inheritance allows one class to inherit the members (fields, methods, properties) of another class (parent). It courages code reuse and supports polymorphism.

Main goal: Inheritance enables new classes to be built on top of existing ones, avoiding code duplication and making the system easier to extend.

How I implemented this concept in my code:

Parent's method:

```
public virtual string FullDescription { get { return
  _description; } }
```

Child class overrided it:

```
public override string FullDescription
{
    get { return $"In the {Name} you can see:\n{_inventory.ItemList
      ()}" ; }
}
```

IV. Polymorphism

Definition: Polymorphism is the ability of different objects to respond in their own way to the same method call.

Main goal: You can call the same method on different objects and get different results.

Benefits:

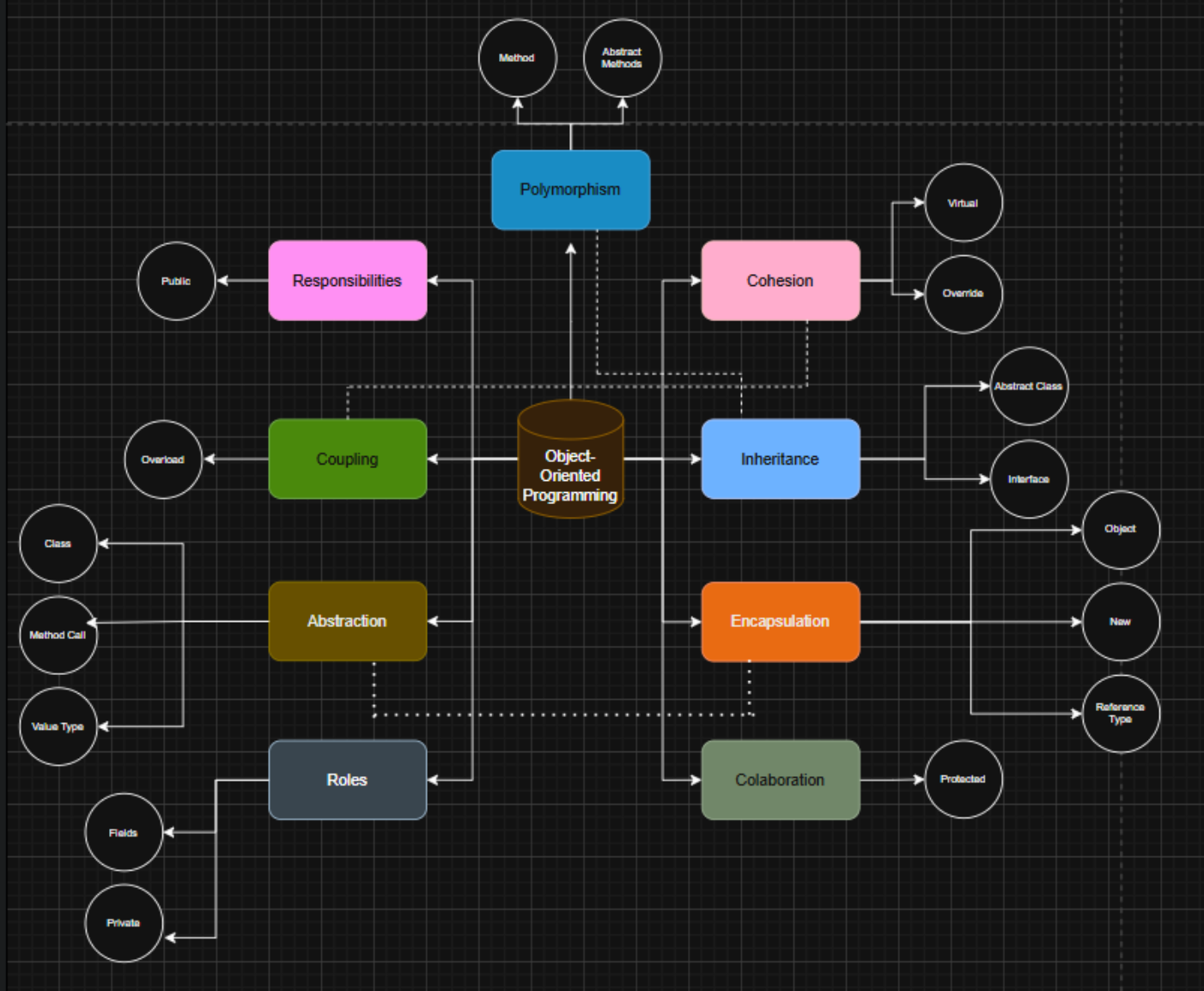
- Code Reusability
- Flexibility
- Maintainability
- Extensibility

How I implemented this concept in my code:

```
public override void Draw()
{
    if (Selected)
    {
        DrawOutLine();
    }
    SplashKit.DrawLine(FillColor, X, Y, _endX, _endY);
}
```

```
//method
public override void Draw()
{
    if (Selected)
    {
        DrawOutLine();
    }
    SplashKit.FillRectangle(FillColor, X, Y, Width, Height);
}
```

⇨ When the method Draw() is called, but depend on different classes, it will draw a Line or a Rectangle.


Method
Abstract Methods
Polymorphism
Virtual
Override
Public
Responsibilities
Cohesion
Abstract Class
Overload
Coupling
Object-Oriented Programming
Inheritance
Interface
Class
Object
Method Call
Abstraction
Encapsulation
New
Value Type
Reference Type
Roles
Colaboration
Protected
Fields
Private