Swinburne University of Technology

COS20007 Object Oriented Programming

6.2P - Key Object Oriented Concepts

PDF generated at 13:06 on Wednesday 29th March, 2023

Shah Nawaz Chowdhury (103830682)

Introduction

Object-Oriented Programming is a programming paradigm that is based on the concept of objects being able to hold data and have certain behaviour. These objects are instances of classes which interact with one another to build an application. OOP is often used due to its reusability and modularity; it is a powerful way to organize and manage complex applications.

In this document, we will explore the four key principles of OOP and use our previous assignments in this unit to relate these concepts directly. Further, we will create a concept map to visualize how these principles and other concepts of OOP are connected.

Four key principles

1. **Encapsulation:** This concept refers to the practice of having/encapsulating certain data and methods into an object, and only exposing the data to other objects through a public interface. This effectively hides the implementation details and ensures the data is protected, which makes it easier to maintain and modify the code without having it impact the rest of the codebase.

   Example: A bank account program would have an account object, here the balance would be stored, and methods such as deposit and withdraw would exist. The balance data would be hidden unless accessed through the public methods deposit/withdraw.

   Relation to previous program: In the drawing program, the data and methods for rectangle were encapsulated in the object MyRectangle, with private variables width and height which can only be accessed and modified through the property and not the variable directly.

2. **Inheritance:** This concept allows objects to inherit properties and behaviours of another object. The child class usually specializes certain behaviours that are generalized in the parent class. This allows code to be reused and organized into a hierarchy, improving readability and maintenance.

   Example: A vehicle and a car has a parent-child relationship, where the car is inheriting from the vehicle object. The car object will inherit common properties such as make and model, but modify or add onto other properties such as engine type or number of wheel. Other vehicles such as trucks can now use this blueprint.

   Relation to previous program: In SwinAdventure, we had an abstract class GameObject which outlined properties such as Name, FullDescription, etc. All types of GameObject such as Player, Item, etc, inherited those properties and added onto them to be more specific. For instance, Player object added onto the FullDescription to also include the items in the player's inventory.

3. **Polymorphism:** This is the ability of an object to take on multiple forms or behaviours. This concept can be implemented in many ways, some of which are interfaces, abstract classes or overload. This allows the object to have multiple implementations of the same method, and allows code to be more flexible and scalable.

Example: A program that has an animal object with subclasses of different types of animals. Its method “speak” can have different implements in each subclass, such as cat would have “meow” and dog would have “woof”.

Relation to previous program: In SwinAdventure, we had an interface IHaveInventory which defined a method Locate. Every object using that interface had their own implementation of that method, allowing Locate to take on multiple forms and be flexible in the codebase.

4. **Abstraction:** This is the process of reducing details by focusing on the essential features of an object. It is usually achieved through the use of abstract classes and interfaces which outline how objects should behave, without implementing the behaviour itself and leaving the implementation to those objects. Abstraction makes the codebase easier to understand and maintain due to its modularity and organized structure.

   Example: A game program can have an abstract class of Character which defines methods such as move, hp, etc, which its subclasses must have. the object with a type of character will be responsible of implementing those methods such as defining how it will move.

   Relation to previous program: In the drawing program, we had an abstract class Shape which defined methods such as DrawOutline, IsAt, etc, that all types of shapes must have. It did not define how to perform those methods as the implementation was upto the type of Shape. For instance, Circle implemented the IsAt method in a way that it looks for Point in circle, while Line implemented point in line.

Concept map