Swinburne University of Technology

COS20007 Object Oriented Programming

Key Object Oriented Concepts

PDF generated at 06:30 on Thursday 12th October, 2023

Produced by OOP Submission Formatter

Name: Phan Vu – ID: 104222099

Object-Oriented Programming Report

Object-oriented programming (OOP) is a conceptual framework used to structure artifacts, commonly referred to as objects, based on their distinctive attributes, data, and functionalities. In object-oriented programming (OOP), the efficacy of interaction among objects is heightened due to the subdivision of objects into smaller groups, hence facilitating ease of management for developers. When discussing Object-Oriented Programming (OOP), the initial concept that is introduced is the notion of a "object" or, in OOP terminology, a "class." Each class possesses its own distinct attributes and functionalities.

E.g: In the context of a class called "Car," there are attributes like "make," "model," and "year," as well as methods called "start" and "stop." In daily life, an automobile has a make, a model, a production year, and the capacity to start and stop. As a result, it is clear that the entity known as a "car" may be studied via the lens of Object-Oriented Programming (OOP).

There exist 4 fundamental principles in computer science: Abstraction, Encapsulation, Inheritance, and Polymorphism.

1. **Abstraction**: posits that objects ought to possess limited knowledge and perform tasks while disregarding the intricacies of their acquisition and operation. This process is commonly referred to as "item design." To achieve more precision, it is essential to note that the establishment of classifications, roles, responsibilities, and collaborations are the four fundamental aspects of objects facilitated by the concept of abstraction. The term "classification" denotes the definition of the object in question. Every software object must possess a function, purpose, and responsibility to fulfil its designated job. The concept of abstraction plays a crucial role in defining the relationship between elements, exemplified by the notion of partnerships.

E.g: "Shape" is an abstract class defining standard methods for shapes, and "Circle" and "Rectangle" are concrete classes that implement those methods. Abstraction simplifies working with different shapes by providing a unified interface.

2. **Encapsulation:** refers to the practice of controlling the accessibility of information by encapsulating both data and functionality into objects. While methods and attributes facilitate the utilisation of objects, ensuring the security of their internal mechanisms is imperative. This is due to the fact that utilising objects necessitates comprehension of their purpose, and inadvertently interfering with them might result in severe ramifications. The modification of access to programmed classes can be achieved by utilising access modifiers.

3. **Inheritance**: is a fundamental concept in object-oriented programming where a class is derived from another type, hence establishing a hierarchical relationship between them.

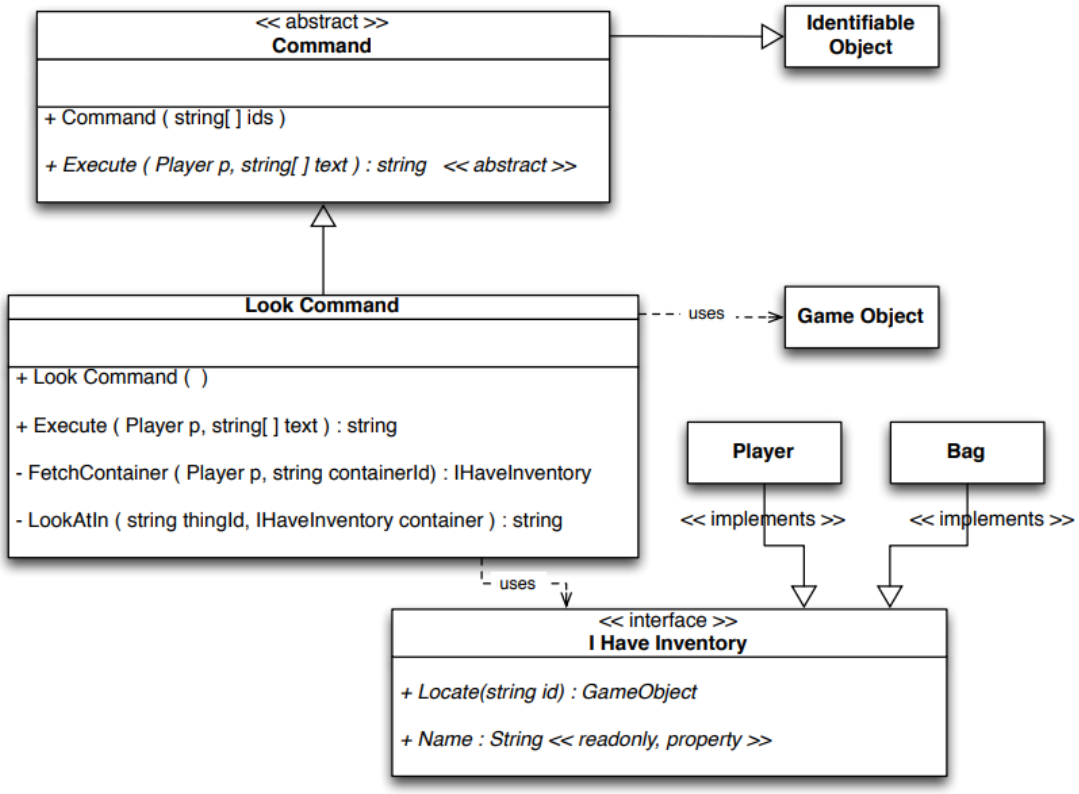
The child-derived class can inherit all members of the parent base class. To enhance code reusability, developers employ the inheritance concept to design a class hierarchy. Abstract classes and interfaces are introduced within the context of inheritance. Abstract courses are designed to be inherited due to their partial inclusion of members without specific reasoning. While classes can inherit from interfaces, the primary purpose of utilising interfaces is to provide additional functionality to classes that do not possess the same set of features as the class hierarchy. While an interface can own many base classes, a class is limited to just one base class.

E.g: a class "Car" inherits from a class called "Vehicle." This means that the "Car" class naturally inherits the characteristic of being a "Vehicle" since a car is a type of vehicle.

4. **Polymorphism**: is a fundamental notion in object-oriented programming that pertains to the ability of a single object to exhibit multiple forms. This versatility is achieved through the mechanism of inheritance. This implies that the child class possesses both its own type as well as the type of its parent, allowing developers to modify its type throughout the program dynamically.

E.g: The "Animal" class serves as a base class, while "Dog" and "Cat" are subclasses. Even though they have different implementations of the "speak" method, they can be treated interchangeably in a list, demonstrating polymorphism in action.

Previous weekly activities have used these principles to execute simple yet effective programs. The case study titled "SwinAdventure" serves as an illustrative example:



SwinAdventure's UML diagram uses abstraction to organize classes, notably for Command class command execution. Encapsulation is used in the LookCommand, which executes, obtains a container, and examines things. Due to the Look Command's execution role, only one method is visible to the public. Through inheritance and polymorphism, a family of objects built from the Identifiable Object class has made object use more flexible and adaptable. When detected, the person interacts with other items like a game object with the "IHaveInventory" feature.

Roles: refer to the functions, assignments, expectations, and requirements associated with an object. They serve as references to the system object. The term "role" is commonly used to refer to a set of interconnected responsibilities that can be exchanged or substituted.

Responsibility: refers to the duties or obligations that an individual, such as an employee, is expected to fulfill. The obligations imposed on the behaviour of an entity are intricately linked to its corresponding duties. These encompass the execution of a certain task or possessing particular information.

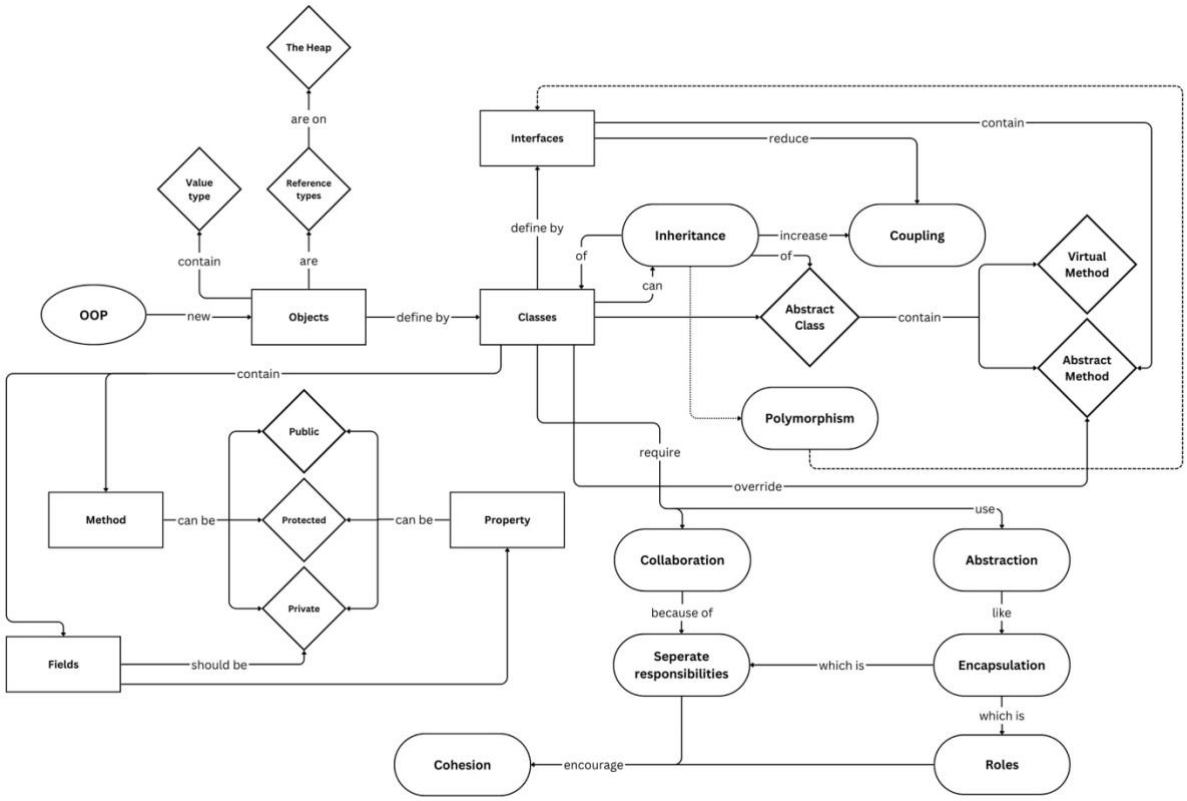
Coupling: refers to the interdependence and interconnectedness between two distinct groups. In the event of little interdependence between the two classes, modifying the code in one class will not have an impact on the other class. Conversely, it is advisable to minimize tight coupling in software development and object-oriented programming (OOP) architecture due to its detrimental effects on code maintainability and modifiability arising from the close interdependence between two classes.

Cohesion: refers to the degree of functional integration within a class. A lack of coherence inside a class indicates a diverse array of methods and functions being employed without a clear sense of purpose or direction. Conversely, a high level of cohesiveness suggests that the class is primarily dedicated to achieving its intended objectives, aligning with the initial goal of the class.

There are some important keywords in the OOP concept:

- private: Attributes can only be accessed by the class itself, requiring a property from that class to interact with them.
- public: This attribute enables interaction between other classes and the current class, in addition to private attributes.
- override: Override a changed technique by developers. Classes that inherit from abstract classes usually have override methods.
- virtual: Virtual methods can be updated in the derived class but can be seen in the base class.
- interface: Is like an abstract class without a body. It cannot generate an object, but a class can inherit from numerous interfaces (note: a class can only inherit from one class, making the interface and abstract class separate).

The Concept Map:


The Heap
are on
Value type
Reference types
contain
are
OOP
new
Objects
define by
Classes
Interfaces
define by
contain
reduce
Inheritance
increase
Coupling
of
of
can
Abstract Class
contain
Virtual Method
Abstract Method
contain
Polymorphism
require
override
use
Public
Protected
Private
Method
can be
can be
Property
Fields
should be
Collaboration
Abstraction
because of
like
Seperate responsibilities
which is
Encapsulation
which is
Cohesion
encourage
Roles