1. Abstraction

Definition: Abstraction is the concept of hiding the complex implementation details of a system and exposing only the essential features to the user. It allows programmers to focus on what an object does rather than how it does it.

Example: In a program simulating a zoo, an **Animal** class might have methods like **AnimalSound()** and **AnimalSpecies()**. Users of the class do not need to know the internal workings of these methods; they only need to know that animal have different sounds and species.

Relation to Programs: In our assignments, referring to Iteration 4, abstraction was used when the **Execute** method of **LookCommand** abstracts the process of parsing the command and determining whether to look at an item directly or within a container. It hides the detailed logic of how to perform these actions from the user.

2. Encapsulation

Definition: Encapsulation is the bundling of data and methods that operate on the data within a single unit or class, and restricting access to some of the object's components. This is done to protect the internal state of the object from unintended interference and misuse.

Example: In the **Animal** class, the **_food** might be a private field, meaning it cannot be accessed directly from outside the class. Instead, methods like **nutrition_source()** are provided to interact with the object's data safely.

Relation to Programs: Encapsulation was applied in our projects by defining classes with private fields and public getter and setter methods. For example, the **GameObject** class with private fields **_name** and **_description**, accessed through **Name()** and **FullDescription()** methods.

3. Inheritance

Definition: Inheritance is a mechanism where a new class inherits properties and behaviors (methods) from an existing class. This allows for code reusability and the creation of a hierarchical relationship between classes.

Example: If we have an **Animal** class, we could create a **Cat** class that inherits from **Animal**. The **Cat** class would have all the attributes and methods of **Animal** such as leg number or with fur, plus any additional attributes or methods specific to a cat.

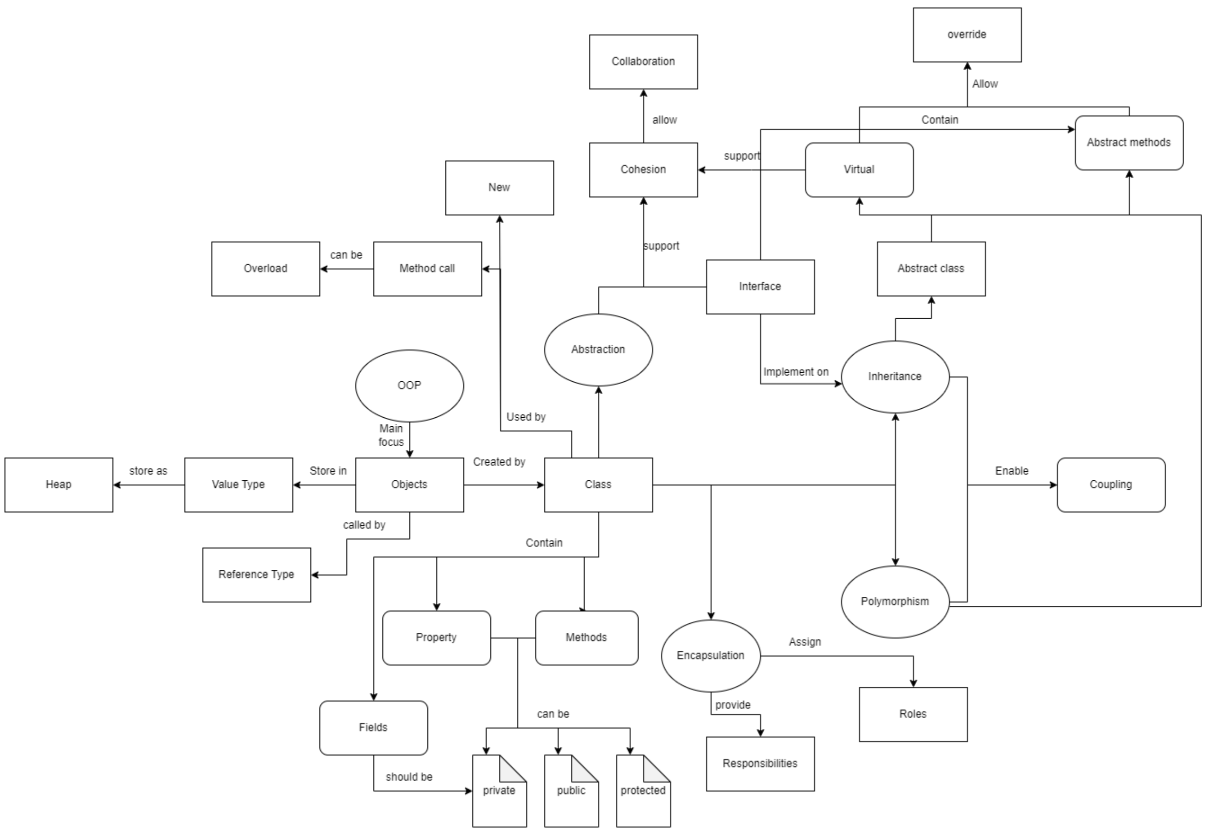
Relation to Programs: In our assignments, inheritance was used to create specialized classes from general ones, such as a **Bag** class inheriting from a **Item** class, adding new attributes like **Iventory**.

4. Polymorphism

Definition: Polymorphism allows objects of different classes to be treated as objects of a common super class. It is the ability to redefine methods in derived classes, and it supports method overriding and method overloading.

Example: If Animal is a superclass with a method **makeSound()**, and Dog and Cat are subclasses, both can override **makeSound()** to provide their specific implementations. A function that takes an **Animal** object can thus accept both Dog and Cat objects.

Relation to Programs: Polymorphism was utilized in our projects when we had methods in interfaces or base classes that were implemented differently in derived classes. For instance, an interface **IHaveInventory** with a method **Locate()**, implemented by **Bag** and **Player** classes.



override
Allow
Collaboration
allow
Contain
Abstract methods
support
Cohesion
Virtual
New
support
Overload
can be
Method call
Interface
Abstract class
Abstraction
Implement on
Inheritance
OOP
Used by
Main focus
Heap
store as
Value Type
Store in
Objects
Created by
Class
Enable
Coupling
called by
Reference Type
Contain
Polymorphism
Property
Methods
Encapsulation
Assign
Roles
provide
can be
Fields
Responsibilities
should be
private
public
protected