COS20007-OBJECT ORIENTED PROGRAMMING

Written Report

Abstract

This is the written report to show my understanding about the Iteration 2 - Players, Items, and Inventory related to OO Principles

HAI NAM NGO

103488515

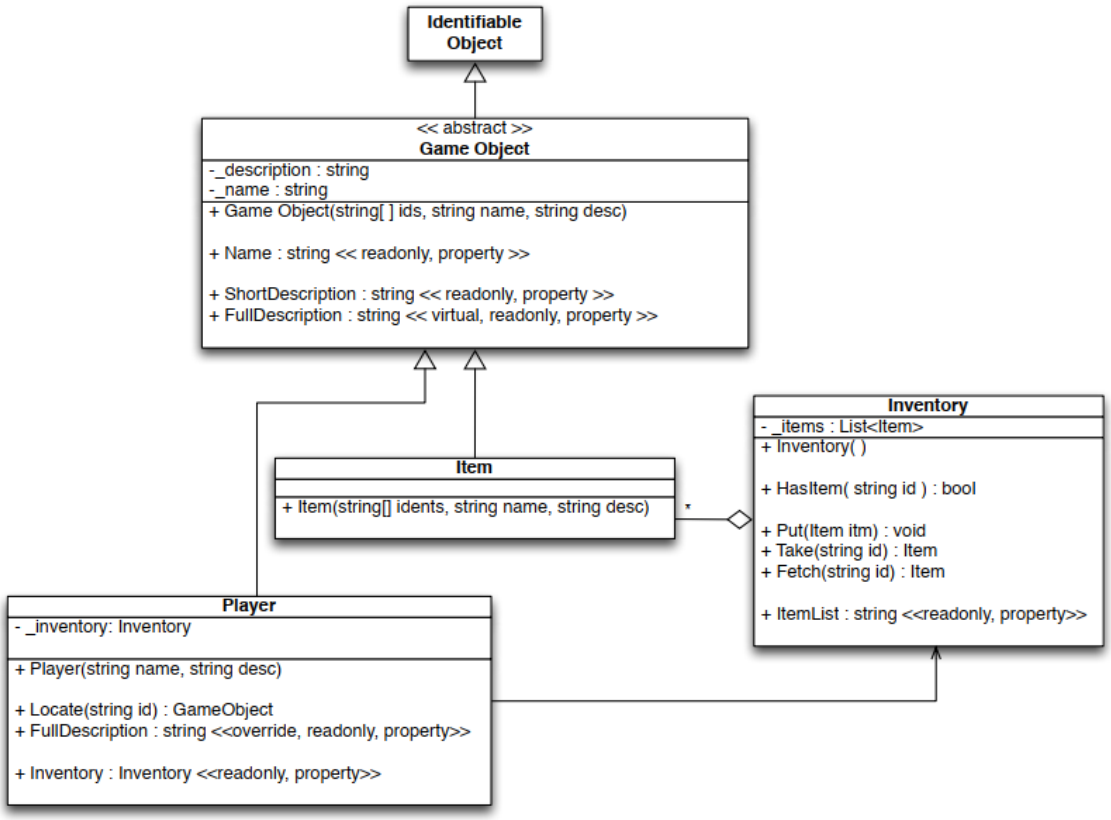

Figure 1 UML Diagram

I will explain what is going on with the UML Diagram first.

GameObject class:

This is an abstract class; it is more like a template for other classes such as Player and Item. As far as I have learned. The Name is just readonly property, so it cannot be given a different value (just “get” without “set”.

The ShortDescription is also the same has Name, but now it returns “Short Description”, so that means it just includes the name and the first ID of the object, nothing more, nothing less.

The FullDescription can be overridden by other class (GameObject’s children’s classes), and they will have the own way to of description.

```
    // It initializes the base class (IdentifiableObject) with the provided IDs
    2 references
    public GameObject(string[] ids, string name, string desc) : base(ids)
    {
        // Initialize the private fields with the provided name and description
        _name = name;
        _description = desc;
    }

    // Define a public property Name to get the object's name
    1 reference
    public string Name
    {
        get
        {
            return _name;
        }
    }

    // Define a public property ShortDescription to get a short description of the object
    5 references | 1/1 passing
    public string ShortDescription
    {
        get
        {
            return $"{_name} ({FirstID})"; // It combines the name and the first ID of the object
        }
    }

    // Define a virtual property FullDescription to get the full description of the object
    7 references | 2/2 passing
    public virtual string FullDescription  // Subclasses can override this property to provide a custom full description
    {
        get
        {
            return _description;
        }
    }
}
```

Figure 2 A part of code in GameObject class

Item class:

So this class is basically a “child” of GameObject class, item is a kind of GameObject, which can inherit the properties of GameObject, I have created an instructor for Item class, including ids, name, desc as parameter and the base is also the same because I take them from GameObject class.

```
1   using System;
2   using System.Collections.Generic;
3
4   namespace SwinAdventure
5   {
        44 references
6       public class Item : GameObject //item is a kind of game object
7       {
8           //create a default constructor of item
            14 references | 5/5 passing
9           public Item(string[] idents, string name, string desc) : base(idents, name, desc)
10          {
11
12          }
13      }
14  }
15
16
```

Figure 3 default constructor for Item class

Inventory class:

It contains a private attribute _items, which is a list of Item objects.

The class provides several public methods:

- HasItem: Checks if a specified item exists in the inventory by its ID.
- Put: Adds an item to the inventory.
- Take: Removes and returns an item from the inventory using its ID.
- Fetch: Locates an item by its ID and returns it.

The class also has a read-only property Item List that likely provides a list or collection of all the items in the inventory. I have provided the code logic directly in my code.

Player Class:
As I mentioned, it is also inherited from the GameObject class.
It has a private attribute _inventory, which is an instance of the Inventory class, indicating that each player has an inventory.
The class provides several public methods and properties:

- A constructor (Player) that accepts a name and a description.
- Locate: Finds a GameObject by its ID.
- FullDescription: This is an overridden read-only property from the GameObject class to provide a detailed description of the player.
- A read-only property Inventory that provides access to the player's inventory.

Moreover, in the UML diagram, we can see a line connecting the Player class to the Inventory class suggests an association, indicating that the Player class uses or has an instance of the Inventory class.

HOW DO OO PRINCIPLES RELATE TO THIS ITERATION 2?

Inheritance:
In the UML diagram, inheritance is symbolized by arrows with hollow arrowheads, directing from the child class towards the parent class. Both the Item and Player classes derive from the GameObject class, signifying they are specialized forms of GameObject, inheriting its attributes and methods. This kind of inheritance promotes code reusability and forms a structured hierarchy. Any general attributes or behaviors applicable to game objects are defined once in the GameObject class and inherited by other specialized subclasses.

Abstraction:
The GameObject class, marked as abstract, indicates it's not meant for direct instantiation. Instead, it sets the foundation for other classes to build upon. Abstraction emphasizes simplifying complex systems by showcasing only essential characteristics. Here, the GameObject class offers a high-level perspective of any interactable in-game object.

Polymorphism:
Polymorphism enables treating objects from various classes as if they belonged to a shared parent class. Method overriding is a common approach to achieve this. The FullDescription property in the GameObject class, denoted as virtual, suggests that its subclasses can redefine it. For instance, the Player class does just that by overriding the FullDescription property to give its tailored description. This exemplifies polymorphism in action: when a game object's FullDescription property is invoked (be it a generic game object, an item, or a player), the system retrieves the correct description corresponding to the object's actual class.

Encapsulation:
Encapsulation revolves around bundling the data (attributes) and the methods (functions) that operate on the data into a single unit or class and restricting the access to some of the object's components. In the provided UML diagram, classes like Inventory and Player have private attributes (e.g., _items and _inventory), which are not directly accessible from outside the class. Instead, these classes provide public methods and properties, like HasItem, Put, and Inventory, to interact with these private attributes. This ensures that the internal state of these objects is shielded from unauthorized external modifications, promoting data integrity.

CONCLUSION
This is my understanding about iteration 2 itself. I have used this for the further and I have everything done for the case study until command processor section.
The application of object-oriented principles such as inheritance, abstraction, polymorphism, and encapsulation is evident throughout the program's design. By adhering to these principles, the program ensures structured, maintainable, and scalable code, paving the way for future iterations and extensions.