School of Science, Computing and Engineering Technologies

Object Oriented Programming

Week 11: SwinAdventure — Iteration 8

Overview

In this week, there are two tasks 11.1 and 11.2. Each task contributes 2% to your final grade. Noting that you need to complete this task before coming to your allocated lab. In the lab, there will be a verification task and short interview to verify your understanding.

These tasks extend from your current SwinAdventure application that you created in Week 10.

Purpose

Task 11.1

This task aims to implement a feature that allows the player to locate items in their current location, in addition to the inventory of the player. You will then modify the implementation of the LookCommand class (from Week 9) to enable this functionality.

Task 11.2
****challenging task****

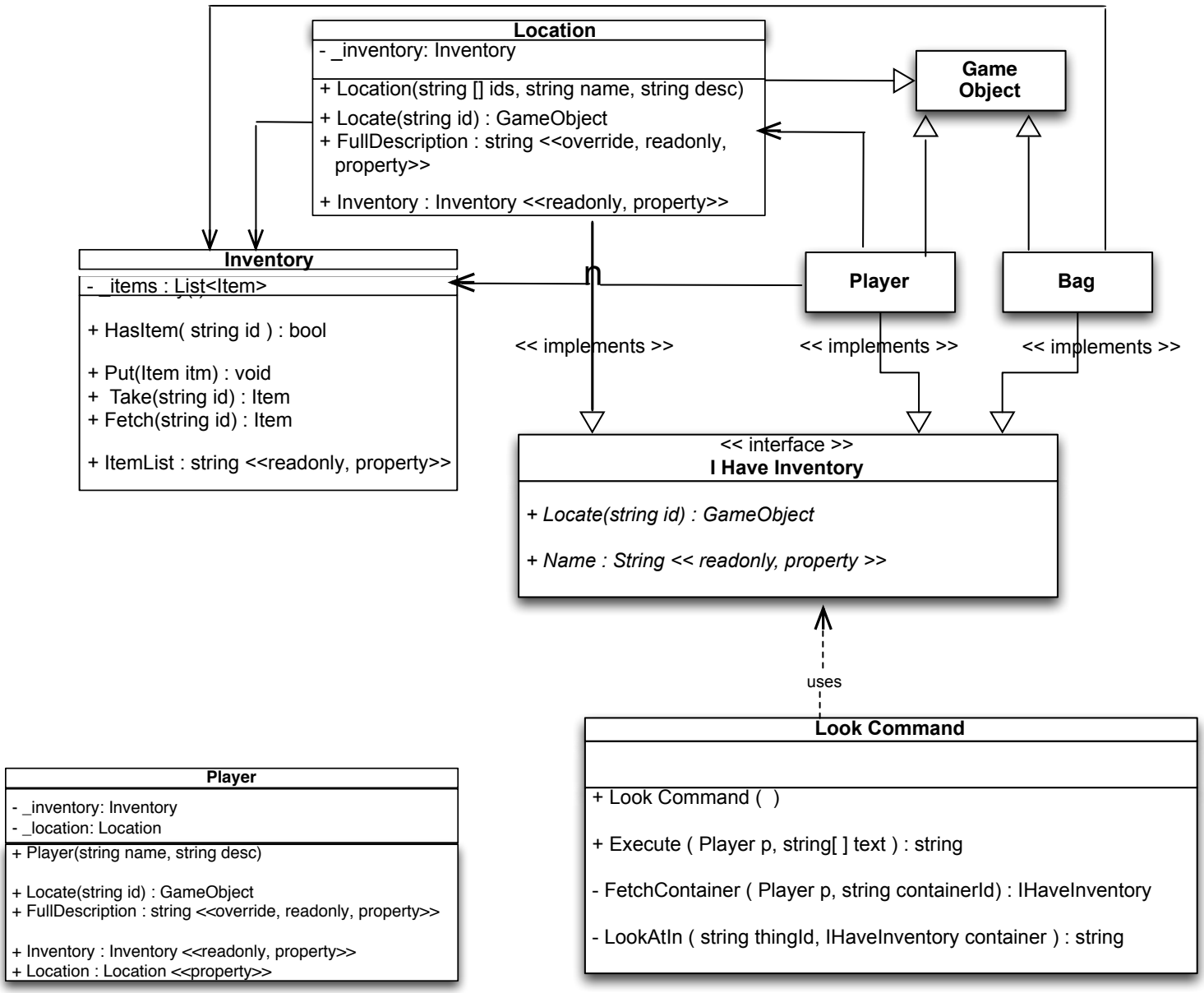
This task also involves developing a new Move command that allows the player to move to a new location. You will then modify the implementation of Program.cs in the SwinAdventure game to process new instructions via the console.

This task requires you to use your own critical thinking to develop a UML design and a sequence diagram for implementing the new command.

Instruction for Task 11.1

1. Review the ***Case Study Requirements*** document. It outlines what you need to create.

2. In this iteration, you will add the Location class, which enables the player to locate not only items in the inventory but also items in the location where the player is currently standing.
3. We provide you with an extended UML class diagram for the Location class as follows. Note that this UML only presents the new implementation that should be added. It should not remove any existing implementations (fields/properties) from previous iterations you have developed.



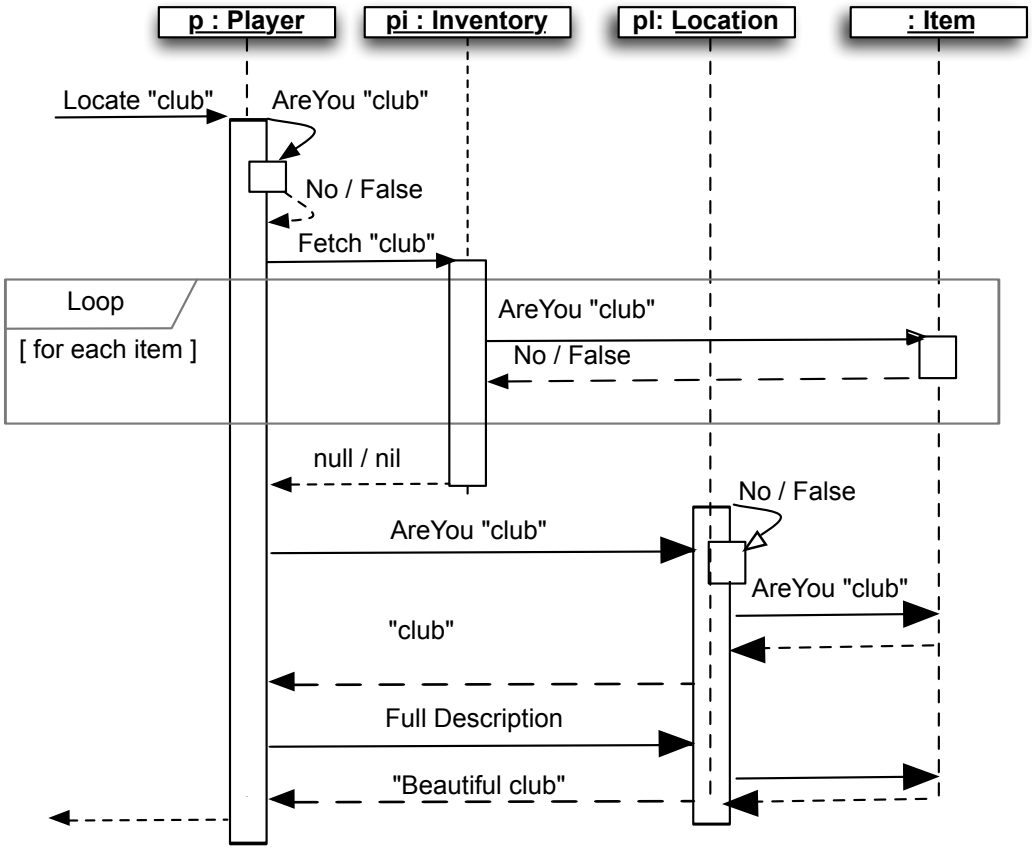
Use the following information to help you design the additions necessary to add locations to the Swin-Adventure.

- Locations:
  - Will need to be identifiable and have a name, and description.
  - Can contain items.
- Players have a location.
- Players "locate" items by checking three things (in order):
  - First checking if they are what is to be located (locate "inventory")
  - Second, checking if they have what is being located (_inventory fetch "gem")
  - Lastly, checking if the item can be located where they are (_location, locate "gem")
- This will change the look command to also include "look" to look at the player's location. Specifically, if the command input only has one word 'look', the full description of the location should be displayed.

Here are some hints for things you will need to test for:

- Locations can identify themselves
- Locations can locate items they have
- Players can locate items in their location

- The following UML sequence diagram explains how locate works in the Player, with the newly added Location aspect to the search.



Below is the coding skeleton for the Location class

```
public class Location : GameObject, IHaveInventory
{
    6 references
    private Inventory _inventory;
    0 references
    public Location(string[] ids, string name, string desc) : base(ids, name, desc)
    {
        _inventory = new Inventory();
    }
    1 reference
    public Inventory Inventory
    {
        get {
            return _inventory;
        }
    }

    0 references
    public GameObject Locate(string id)
    {
        if (AreYou(id))
        {
            return this;
        }
        else
        {
            return Inventory.Fetch(id);
        }
    }
    public override string FullDescription
    {
        get
        {
            string nameDescription;
            string inventoryDescription;

            if(Name !=null && Name !=""){
                nameDescription = Name;
            }
            else{
                nameDescription = " an unknown location";
            }

            if(_inventory !=null && _inventory.ItemList !=null)
            {
                inventoryDescription = _inventory.ItemList;
            } else {
                inventoryDescription = "there are no items at this location. ";
            }

            return "You are in " + nameDescription + ". " +
                    base.FullDescription +
                    "\n Here, you can see :\n" + inventoryDescription;
        }
    }
```

Tasks.

- Update the Player class using the UML diagrams provided
- Update the Look command class:
    - Display the location's description if the command only has one keyword 'look'
    - Allow to fetch the items in the location. **Tips**. Modify the FetchContainer method.
- Test your implementation to make sure players can locate items in their location
- Optional. You can create the unit test for the Location class.

Instruction for Task 11.2

Implement Path, Direction, and the Move Command.

1. This task is a challenging custom task. It is up to your custom design and implementation to meet the following requirements.
   - Have the Path objects move the player to the new location. This will allow for flexibility like lockable paths etc.
   - Make Path's identifiable. The identifiers indicate the direction, and can be used to locate the path from the location.
   - The Move Command is identified by the words "move", "go", "head", "leave", ...

Here are some hints for things you will need to test for:

- Path can move player to the Path's destination
- You can get a Path from a Location given one of the path's identifiers
- Players can leave a location, when given a valid path identifier
- Players remain in the same location when they leave with an invalid path identifier

Tasks:

- Draw a UML class diagram to show what needs to be added (hand drawing on paper). Present to your lab instructor.
- Draw a sequence diagram to explore how moving will work. For example: execute "move north" is sent to a Move Command (hand drawing on paper). Present to your lab instructor.
- Implement the unit tests, and features to support them.
- Demonstrate the unit tests and the console application to your lab.