School of Science, Computing and Engineering Technologies

Object Oriented Programming

Week 10: SwinAdventure — Iteration 7

Overview

In this week, there are two tasks 10.1 and 10.2. 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 9.

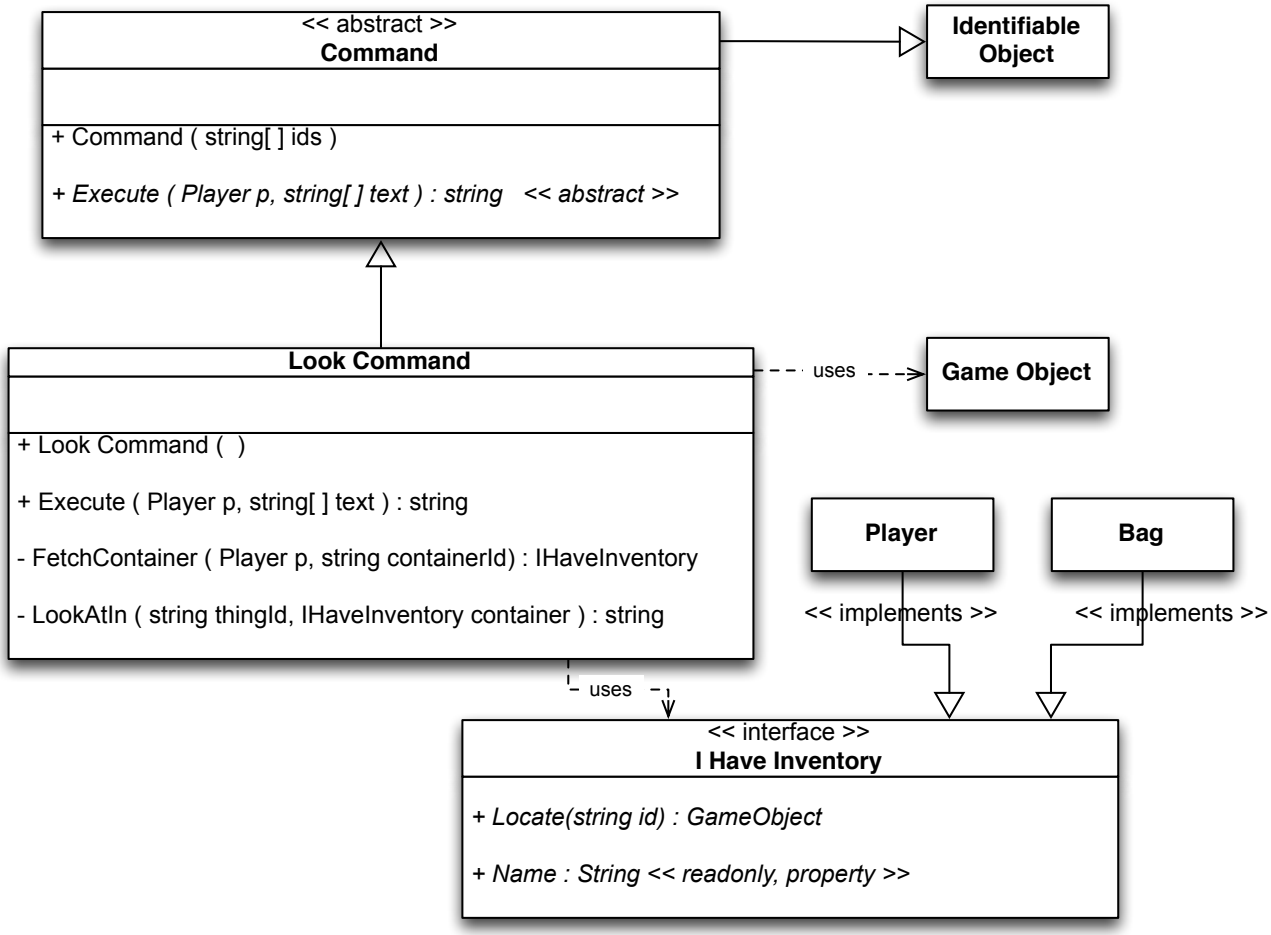
Purpose

Task 10.1 This task aims to develop the first of the commands, the look command in the SwinAdventure game. You will then create unit test cases for this class.

Task 10.2 This task aims to create a console application using the above look command and previously developed classes including Item, Bag, and Player.

Instruction for Task 10.1

1. Review the ***Case Study Requirements*** document. It outlines what you need to create.
2. In this iteration you will add the first command of the SwinAdventure game, called the Look command. At a high level, the command allows to create a small application where the user can look at items they have.
3. We provide you with an UML class diagram for the Look command class as follows.



4. As there will be a number of Commands (we will implement in the following weeks), an abstract **Command** class has also been added.

This abstract class inherits from **Identifiable Object**, as each of the commands needs to be identifi-able. When data is entered by the user, each Command object will be asked "Are You" and the first work of the command. For example, with "look at pen" each Command would be asked "Are You 'look' " to locate the Command to process the text. As a result the Look Command should be identified by "look".

The Look Command will handle the instructions like:

```
look at pen in bag
look at bag in inventory
look at pen
look at bag
```

To handle this the Look Command will need to perform a set of action:

1. Locate the "container" where the item resides. For example the *bag* in "look at pen in bag", or the *player* in "look at pen".
   - Return "I cannot find the <<container text>>" if the container cannot be found. For example, "look at pen in bag" may return "I cannot find the bag".
2. Locate the item from the "container"
   - Return "I cannot find the <<item text>> in the <<container name>>" if the item cannot be found. For example, "look at pen in bag" may return "I cannot find the pen in the small bag" (in this case "small bag" is the name of the Bag container).
3. Return the full description of the Game Object found.

For this to work the code needs a way of treating Bags and Players in the same way: as objects that container other items. To handle this the **I Have Inventory** interface/protocol has been added. Objects that implement this must be able to perform the following tasks for the Look Command:

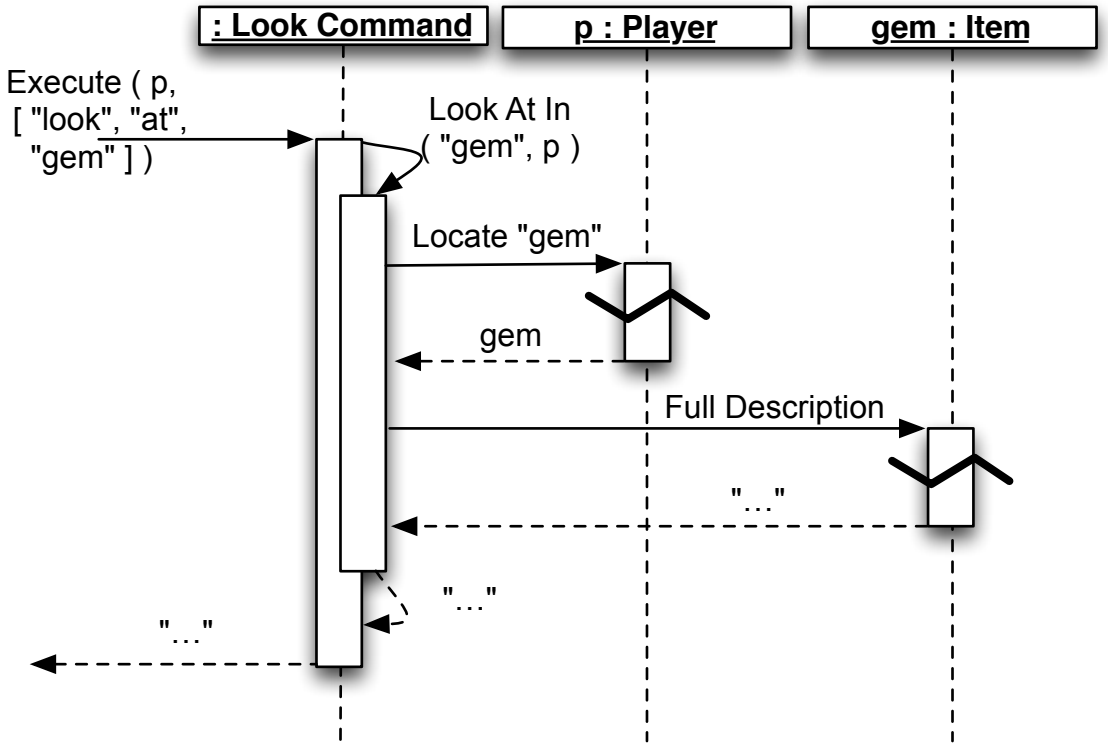
- **Locate** and item given its id. Once the container is located from the Player, it is then asked to locate the item from the text id the user entered.
- The container needs to be able to return its **Name**. This can then be used when the item cannot be found in the container.

The command input will be supplied as a list of individual words. For example: "look at pen in bag" will be passed in as the list ["look", "at", "pen", "in", "bag"]. The conversion of the text from a single string to a list will be handled elsewhere.

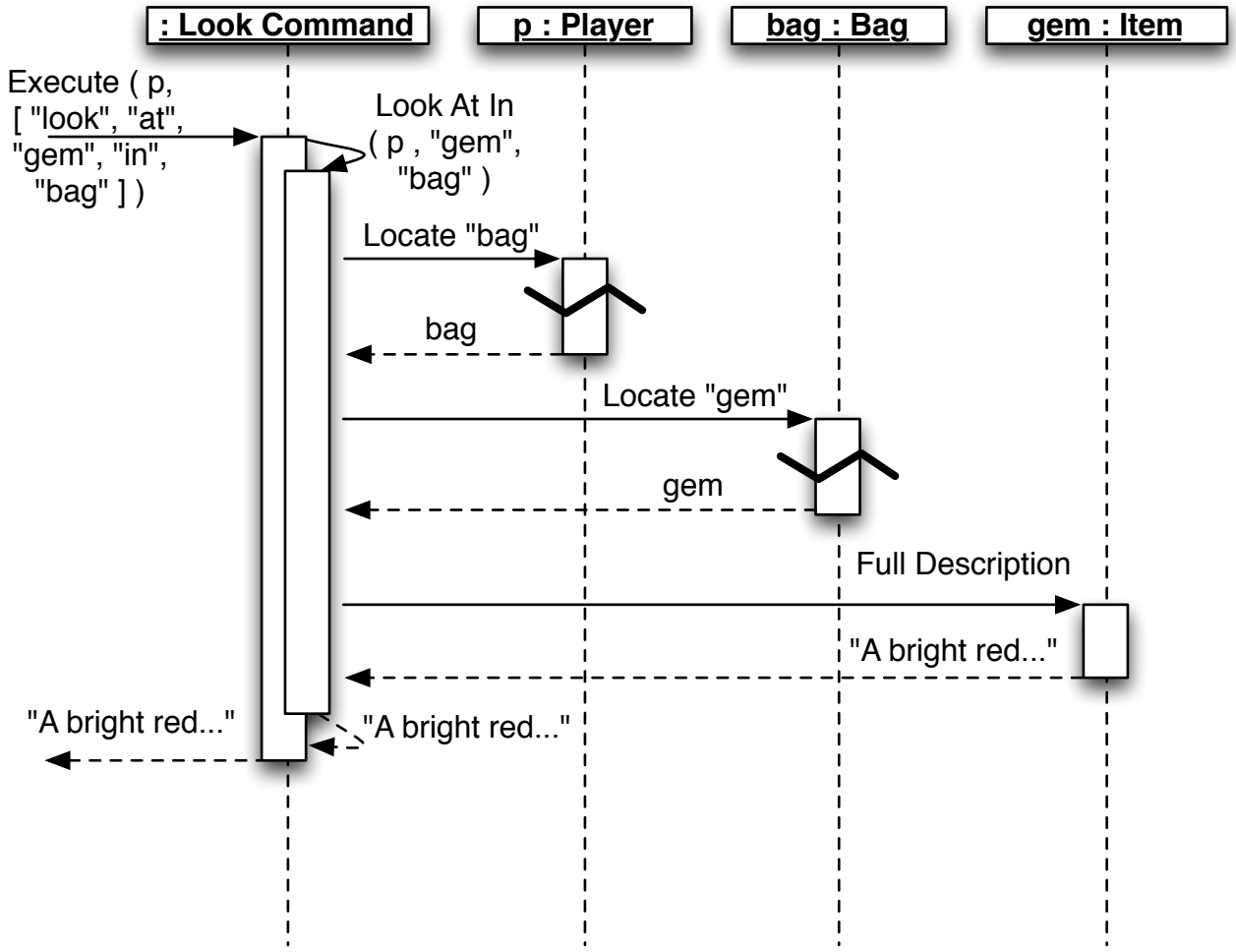
Processing the array will be performed as follows:

1. There must be either 3 or 5 elements in the array, otherwise return "I don't know how to look like that"
2. The first word must be "look", otherwise return "Error in look input"
3. The second word must be "at", otherwise return "What do you want to look at?"
4. If there are 5 elements, then the 4th word must be "in", otherwise return "What do you want to look in?"
5. If there are 3 elements, the container is the player
6. If there are 5 elements, then the container id is the 5th word
   1. Call **FetchContainer**, and have this method retrieve the container from the Player.
7. The item id is the 3rd word
8. Perform the method **LookAtIn**, with the container and the item id

The following shows the sequence of messages involved in performing the command "look at gem". In this case the container is the player themselves, though the logic is the same as when it is another object that matches the I Have Inventory interface/protocol.



The following sequence diagrams shows the command "look at gem in bag". When the command is "look at x in y" then the **Locate Container** method asks the player to locate "y", and returns this as the container to locate "x".



Notes:

- You will need to cast the `GameObject` to `IHaveInventory` use the following to perform a safe type cast. This will set container to null if obj is not an object that implements the I Have Inventory interface/protocol.

```
container = obj as IHaveInventory;
```

Below is the implementation of the abstract class Command.

```
public abstract class Command : IdentifiableObject
{
    0 references
    public Command(string[] ids) : base(ids)
    {
    }

    0 references
    public abstract string Execute(Player p, string[] text);
}
```

The coding template of the Look command class is as follows.

```
using System;
namespace SwinAdventure
{
    1 reference
    public class LookCommand : Command
    {
        0 references
        public LookCommand(string[] ids) : base(ids)
        {
        }
        0 references
        public override string Execute(Player p, string[] text)…
        1 reference
        private IHaveInventory FetchContainer(Player p, string containerId)…
        1 reference
        private string LookAtIn(string thingId, IHaveInventory container)
        {
            GameObject itm = container.Locate(thingId);
            if (itm == null)
            {
                return "I can't find the " + thingId;
            }
            return itm.FullDescription;
        }
    }
}
```

You need to complete the Execute method and the FetchContainer method.

Then, open the SwinAdventure test project and create the following test cases for the Look command.

Look Command Unit Tests	
Test Look At Me	Returns players description when looking at "inventory"
Test Look At Gem	Returns the gems description when looking at a gem in the player's inventory.
Test Look At Unk	Returns "I can't find the gem" when the the player does not have a gem in their inventory.
Test Look At Gem In Me	Returns the gem's description when looking at a gem in the player's inventory. "look at gem in inventory"
Test Look At Gem in Bag	Returns the gems description when looking at a gem in a bag that is in the player's inventory.
Test Look at Gem in No Bag	Returns "I can't find the bag" when the player does not have a bag in their inventory.
Test Look at No Gem in Bag	Returns "I can't find the gem" when the bag does not have a gem in its inventory.
Test Invalid Look	Test look options to check all error conditions. This include "**look around**", or "**hello your student ID**", "**look at your name**".

The coding template for the LookCommandTests.cs is as below (continue in the next page).

```
public class LookCommandTests
{
    3 references
    private Item _testItem;
    15 references
    private Player _testPlayer;
    4 references
    private Bag _testMoneyBag;
    12 references
    private LookCommand _testLookCommand;

    [SetUp]
    0 references
    public void Setup()
    {
        _testLookCommand = new LookCommand(new string[] { "" });
        _testPlayer = new Player("HarryPotter", "a student");

        _testItem = new Item(new string[] { "gem", "Ruby" }, "A Ruby", "A bright Pink ruby");
        _testMoneyBag = new Bag(new string[] { "bag", "money" }, "Money Bag", "A bag that contains Valuables");

        _testPlayer.Inventory.Put(_testItem);
    }
```

```
        [Test]
        0 references
        public void LookAtPlayer() ⋯
        [Test]
        0 references
        public void LookAtItem() ⋯
        [Test]
        0 references
        public void LookAtNothing() ⋯
        [Test]
        0 references
        public void LookAtItemInPlayer() ⋯

        [Test]
        0 references
        public void LookAtItemInBag() ⋯
        [Test]
        0 references
        public void LookAtItemInNoBag() ⋯
        [Test]
        0 references
        public void LookAtNothingInBag() ⋯
        [Test]
        0 references
        public void InvalidLook() ⋯

    }
}
```

Instruction for task 10.2

Extend the Program.cs in your SwinAdventure project to meet the following requirements.

1. Get the player's name and description from the user, and use these details to create a Player object.
2. Create two items and add them to the the player's inventory
3. Create a bag and add it to the player's inventory
4. Create another item and add it to the bag
5. Loop reading commands from the user, and getting the look command to execute them.

   Loop command could be developed as follows.

```
bool finished = false;
LookCommand cmd = new LookCommand();

while (!finished)
{
    Console.WriteLine("Enter a command");
    string command = Console.ReadLine();

    if (command.ToLower() == "exit")
    {
        finished = true;
        break;
    }

    string[] split = command.Split(" ");

    Console.WriteLine(cmd.Execute(player, split));
}
```