Swinburne University of Technology

COS20007 Object Oriented Programming

6.1P - Case Study - Iteration 4 - Look Command

PDF generated at 18:51 on Saturday 25th March, 2023

Produced by OOP Submission Formatter

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public interface IHaveInventory
    {
        public GameObject Locate(string id);

        public string Name { get; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Player : GameObject, IHaveInventory
    {
        //local variables
        // inventory of items of the player
        private Inventory _inventory;

        //comstructor
        public Player(string name, string desc) : base(new string[] {"me",
        "inventory"}, name, desc)
        {
            _inventory = new Inventory();
        }

        //methods

        //locate method that returns a gameobject based on the id.
        // for now it only returns the player itself if any of the above identifiers
        are entereed
        // or returns items that exists in its inventory
        public GameObject Locate(string id)
        {
            if (AreYou(id))
            {
                return this;
            }
            else if (_inventory.HasItem(id))
            {
                return _inventory.Fetch(id);
            }
            else return null;
        }

        //properties

        // override FullDescription property to include the player's name, and the
        shortdescription of themselves and their items in their inventory
        public override string FullDescription
        {
            get
            {
                return $"You are {Name}, {Description}.\nYou are
        carrying:\n{_inventory.ItemList}";
            }
        }

        public Inventory Inventory => _inventory;
    }
}
```

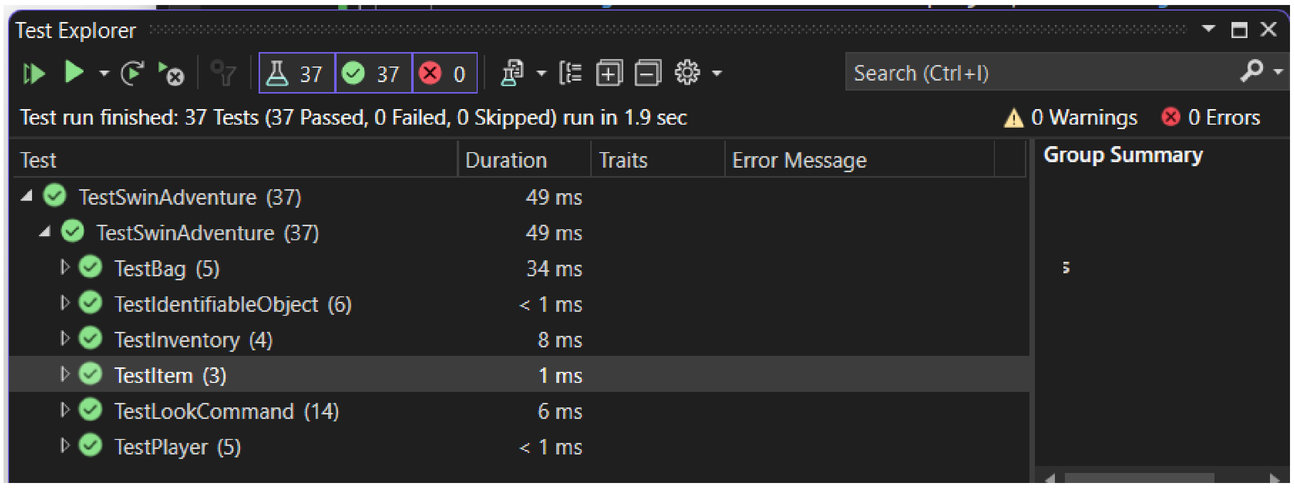
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Bag : Item, IHaveInventory
    {
        //local variables
        private Inventory _inventory;

        //constructor
        public Bag(string[] ids, string name, string desc):base(ids, name, desc)
        {
            _inventory = new Inventory();
        }

        //methods
        public GameObject Locate(string id)
        {
            if (AreYou(id))
            {
                return this;
            }
            else if (_inventory.HasItem(id))
            {
                return _inventory.Fetch(id);
            }
            else return null;
        }

        //properties
        public override string FullDescription
        {
            get
            {
                return $"In the {Name} you can see:\n" + _inventory.ItemList;
            }
        }

        public Inventory Inventory => _inventory;
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public abstract class Command : IdentifiableObject
    {
        public Command(string[] ids) : base(ids) { }

        // the commnad recieved will split each string and store it in an array
        public abstract string Execute(Player p, string[] text);

    }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class LookCommand : Command
    {
        //constructor with default identifier "look"
        public LookCommand() : base(new string[] { "look" })
        {

        }

        //main property that returns the output based on the command
        public override string Execute(Player p, string[] text)
        {
            //initialize container and item to hold the values of the input
            IHaveInventory container = null;
            string itemId;

            // first check the length of the input (each string stored in an array)
            // if its not equal to 3 and 5 return an error because look command can
↪   only process those inputs
            if (text.Length != 3 && text.Length != 5 )
            {
                return "I don't know how to look like that";
            }
            else
            {
                //if the first text is not "look" then there is command error
                if (text[0] != "look")
                {
                    return "Error in look input";
                }
                // same with if second text is not "at"
                if (text[1] != "at")
                {
                    return "What do you want to look at?";
                }
                // same with if 4th text is not "in"
                // because if there are 5 inputs we will be look for the item inside
↪   a bag
                if (text.Length == 5 && text[3] != "in")
                {
                    return "What do you want to look in?";
                }
                //if none of the errors above occur, assign the conatiner value
↪   based on the inputs recieved
                switch(text.Length)
                {
                    // player is the container if 3 inputs
                    // player object is also converted into IHaveInventory using safe type cast
                    case 3:
                        container = p;
                        break;
                    // bag is the container if 5 inputs
                    case 5:
                        // the last input would be the name of the bag
                        // so here a method is called that returns the bag object, the safe type cast is performed in the method
                        container = FetchContainer(p, text[4]);
                        // if object is null then return an error
                        if (container == null)
                        {
                            return $"I can't find the {text[4]}";
                        }

                        break;
                }
                // 3rd input will be the item
                itemId = text[2];
                // lastly, return the full description of the item if no errors encountered
                return LookAtIn(itemId, container);
            }
        }
        // method to fetch a bag that the player has, if asked to locate an item inside a bag
        public IHaveInventory FetchContainer(Player p, string containerId)
        {
            return p.Locate(containerId) as IHaveInventory;
        }

        //return the full description of the item being looked
        public string LookAtIn(string thingId, IHaveInventory container)
        {
            //return an error if the item doesnt exist in the container
            if (container.Locate(thingId) == null)
            {
                return $"I can't find the {thingId}";
            }
            else
            {
                return container.Locate(thingId).FullDescription;
            }
        }
    }
}
```

```
using SwinAdventure;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TestSwinAdventure
{
    [TestFixture]
    public class TestLookCommand
    {
        LookCommand look;
        Player player;
        Bag bag;
        Item gem;

        [SetUp]
        public void Setup()
        {
            look = new LookCommand();
            player = new Player("shah", "the student");
            bag = new Bag(new string[] { "bag" }, "bag", "This is a bag");
            gem = new Item(new string[] { "gem" }, "a gem", "a bright red crystal");
        }

        // test looking at your own inventory
        [Test]
        public void TestLookAtMe()
        {
            string actual = look.Execute(player, new string[] { "look", "at",
 ↪  "inventory" });
            string expected = "You are shah, the student.\nYou are carrying:\n";

            Assert.That(actual, Is.EqualTo(expected));
        }

        // test looking at a gem
        [Test]
        public void TestLookAtGem()
        {
            player.Inventory.Put(gem);

            string actual = look.Execute(player, new string[] { "look", "at", "gem"
 ↪  });
            string expected = "a bright red crystal";

            Assert.That(actual, Is.EqualTo(expected));
        }

        //test looking at a non existent item in your inventory
        [Test]
        public void TestLookAtUnknown()
        {
            string actual = look.Execute(player, new string[] { "look", "at", "gem"
 ↪  });
            string expected = "I can't find the gem";

            Assert.That(actual, Is.EqualTo(expected));
        }

        // test looking at gem in your own inventory
        [Test]
        public void TestLookAtGemInMe()
        {
            player.Inventory.Put(gem);

            string actual = look.Execute(player, new string[] { "look", "at", "gem",
 ↪  "in", "inventory" });
            string expected = "a bright red crystal";

            Assert.That(actual, Is.EqualTo(expected));
        }

        //test looking at gem in your bag
        [Test]
        public void TestLookAtGemInBag()
        {
            bag.Inventory.Put(gem);
            player.Inventory.Put(bag);

            string actual = look.Execute(player, new string[] { "look", "at", "gem",
 ↪  "in", "bag" });
            string expected = "a bright red crystal";

            Assert.That(actual, Is.EqualTo(expected));
        }

        //test looking at gem in a bag that you dont have
        [Test]
        public void TestLookAtGemInNoBag()
        {
            player.Inventory.Put(gem);

            string actual = look.Execute(player, new string[] { "look", "at", "gem",
 ↪  "in", "bag" });
            string expected = "I can't find the bag";

            Assert.That(actual, Is.EqualTo(expected));
        }

        // test looking at non existent item in your bag
        [Test]
        public void TestLookAtNoGemInBag()
        {
            player.Inventory.Put(bag);

            string actual = look.Execute(player, new string[] { "look", "at", "gem",
 ↪  "in", "bag" });
            string expected = "I can't find the gem";

            Assert.That(actual, Is.EqualTo(expected));
        }

        //test error with invalid input

        //invalid look command
        [Test]
        public void TestInvalidLook()
        {
            string actual = look.Execute(player, new string[] { "find", "the", "gem"
 ↪  });
            string expected = "Error in look input";

            Assert.That(actual, Is.EqualTo(expected));
        }

        // invalid number of inputs
        [TestCaseSource(nameof(InvalidLengthTestCases))]
        public void TestInvalidLength(string[] toTest )
        {
            Assert.That(look.Execute(player, toTest),
                Is.EqualTo("I don't know how to look like that"));
        }
        private static IEnumerable<string[]> InvalidLengthTestCases()
        {
            yield return new string[] { "look","bag" };
            yield return new string[] { "look" };
            yield return new string[] { "look", "at", "gem", "in", "the", "bag" };
            yield return new string[] { "look", "at", "big", "bag" };
        }

        // invalid command for 2nd input "at"
        [Test]
        public void TestInvalidAt()
        {
            string actual = look.Execute(player, new string[] { "look", "in", "gem"
 ↪  });
            string expected = "What do you want to look at?";

            Assert.That(actual, Is.EqualTo(expected));
        }

        //invalid command for 4th input "in"
        [Test]
        public void TestInvalidIn()
        {
            string actual = look.Execute(player, new string[] { "look", "at", "gem",
 ↪  "at", "bag" });
```

```
            string expected = "What do you want to look in?";

            Assert.That(actual, Is.EqualTo(expected));
        }


    }
}
```



Test run finished: 37 Tests (37 Passed, 0 Failed, 0 Skipped) run in 1.9 sec

0 Warnings 0 Errors

Test	Duration	Traits	Error Message
TestSwinAdventure (37)	49 ms		
TestSwinAdventure (37)	49 ms		
TestBag (5)	34 ms		
TestIdentifiableObject (6)	< 1 ms		
TestInventory (4)	8 ms		
TestItem (3)	1 ms		
TestLookCommand (14)	6 ms		
TestPlayer (5)	< 1 ms		

Group Summary