Swinburne University of Technology

COS20007 Object Oriented Programming

7.2C - Case Study - Iteration 6 - Locations

PDF generated at 22:28 on Sunday 26th March, 2023

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

namespace SwinAdventure
{
    public class Location : GameObject, IHaveInventory
    {
        private Inventory _inventory;

        public Location(string name, string desc) : base(new string[] {"room", "here"}, name, desc)
        {
            _inventory = new Inventory();
        }

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

        public override string FullDescription
        {
            get
            {
                return $"You are in {Name}\n{Description}\nIn this room 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 TestSwinAdventure
{
    [TestFixture]
    public class TestLocation
    {
        //initialize variables
        Location location;
        Player player;
        Item sword;

        [SetUp]
        public void Setup ()
        {

            location = new Location ("a garden", "This is a garden");
            player = new Player("shah", "the student");
            sword = new Item(new string[] { "Sword" }, "a bronze sword", "This is a
↪   bronze sword");

            // add item to location, and set player's location
            location.Inventory.Put(sword);
            player.Location = location;
        }

        // test if location can identify itself
        [Test]
        public void TestIdentifyLocation ()
        {
            Assert.That(location.Locate("room"), Is.SameAs(location));
        }

        //test if location can identify an item in its inventory
        [Test]
        public void TestIdentifyItemsInLocation ()
        {
            Assert.That(location.Locate("sword"), Is.SameAs(sword));
        }

        // test that player can locate an item in its location
        [Test]
        public void TestIdentifyItemsInPlayerLocation()
        {
            Assert.That(player.Locate("sword"), Is.SameAs(sword));
        }

        //test location's full description
        [Test]
```

```
        public void TestLocationFullDescription()
        {
            string actual = location.FullDescription;
            string expected = "You are in a garden\nThis is a garden\nIn this room you can see:\na bronze sword (sword)\n";

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

```
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;
        private Location _location;

        //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 if (_location != null)
            {
                return _location.Locate(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;
        public Location Location
        {
            get => _location;
            set => _location = value;
        }
    }
}
```

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

namespace TestSwinAdventure
{
    [TestFixture]
    public class TestPlayer
    {
        Player player;
        Item sword;
        Location location;
        Item gem;

        [SetUp]
        public void Setup()
        {
            player = new Player("shah", "the student");
            sword = new Item(new string[] { "Sword" }, "a bronze sword", "This is a bronze sword");
            player.Inventory.Put(sword);

            gem = new Item(new string[] { "gem" }, "a gem", "a bright red crystal");
            location = new Location("garden", "This is a garden");
            location.Inventory.Put(gem);

            player.Location = location;
        }

        [Test]
        public void TestIsIdentifiable()
        {
            Assert.That(player.AreYou("me"), Is.True);
            Assert.That(player.AreYou("inventory"), Is.True);
        }

        [Test]
        public void TestLocateItems()
        {
            Assert.That(player.Locate("sword"), Is.SameAs(sword));
            Assert.That(player.Inventory.HasItem("sword"), Is.True);
        }

        [Test]
        public void TestLocateItself()
        {
            Assert.That(player.Locate("me"), Is.SameAs(player));
            Assert.That(player.Locate("inventory"), Is.SameAs(player));
        }

        [Test]
```

```
        public void TestLocateNothing()
        {
            Assert.That(player.Locate("scythe"), Is.SameAs(null));
        }

        [Test]
        public void TestLocateLocation()
        {
            Assert.That(player.Locate("room"), Is.SameAs(location));
        }

        [Test]
        public void TestLocateItemInLocation()
        {
            Assert.That(player.Locate("gem"), Is.SameAs(gem));
        }
        [Test]
        public void TestFullDescription()
        {
            Assert.That(player.FullDescription,
                Is.EqualTo("You are shah, the student.\nYou are carrying:\na bronze sword (sword)\n"));
        }
    }
}
```

```
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 = null;

            // 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 != 1 && 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.Length != 1 && 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)
                {
```

```
                    // if the input is just "look" then it will look at the room the player is in
                    case 1:
                        container = p;
                        itemId = "room";
                        break;
                    // player is the container if 3 inputs
                    // player object is also converted into IHaveInventory using safe type cast
                    case 3:
                        container = p;
                        itemId = text[2];
                        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]}";
                        }
                        itemId = text[2];
                        break;
                }
                // 3rd input will be the item
                // 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;
            }
        }
```

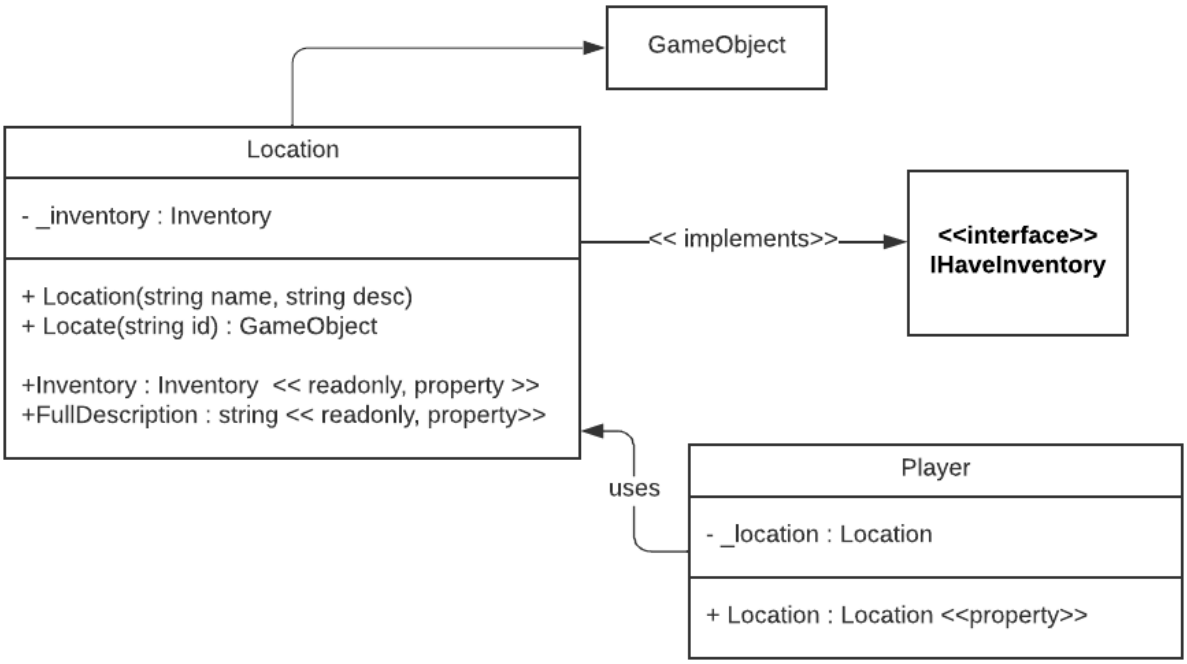
```
    }
}
```

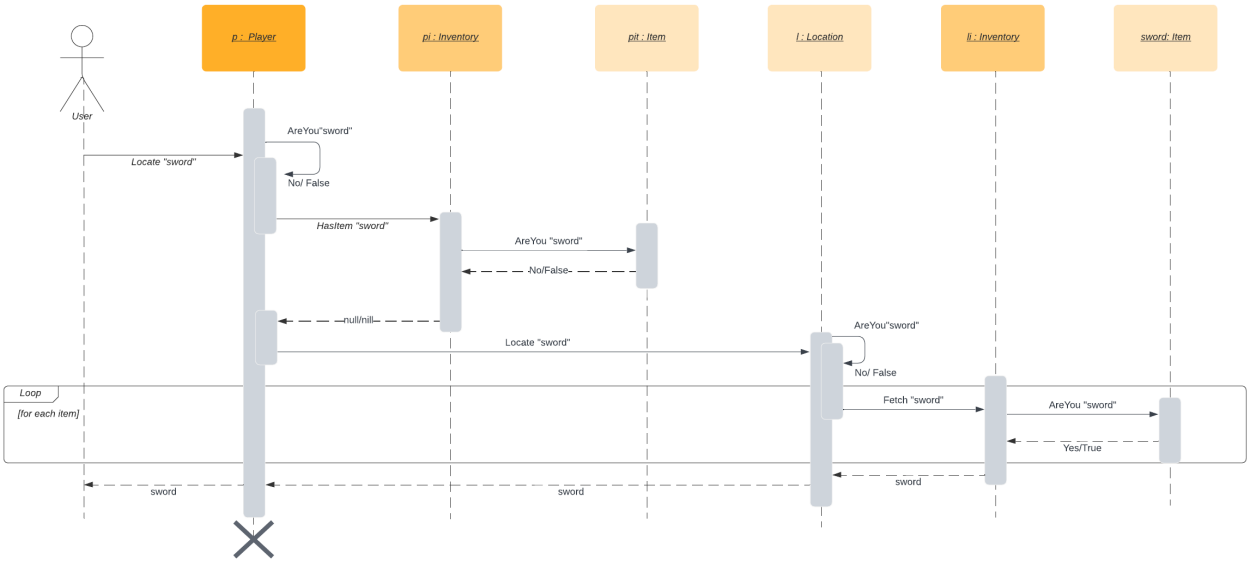
```
using SwinAdventure;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

        [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");
            location = new Location("garden", "This is a garden");
        }

        // 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
```

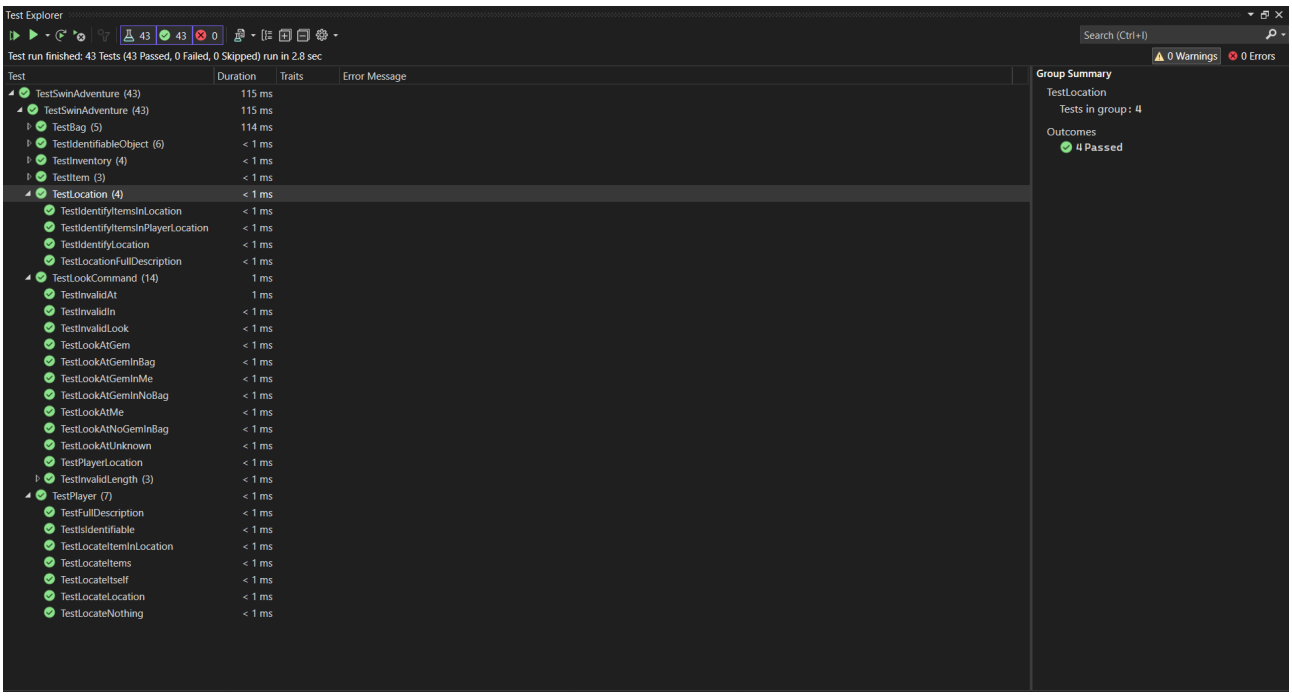
```
        [Test]
        public void TestLookAtUnknown()
        {
            string actual = look.Execute(player, new string[] { "look", "at", "gem" });
            string expected = "I can't find the gem";

            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 "look" to look at player's location
        [Test]
        public void TestPlayerLocation()
        {
            player.Location = location;
            string actual = look.Execute(player, new string[] { "look" });
            Assert.That(actual, Is.EqualTo(location.FullDescription));
        }

        //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", "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));
        }


    }
}
```


GameObject
Location
- _inventory : Inventory
+ Location(string name, string desc)
+ Locate(string id) : GameObject
+Inventory : Inventory << readonly, property >>
+FullDescription : string << readonly, property>>
<< implements>>
<<interface>>
IHaveInventory
uses
Player
- _location : Location
+ Location : Location <<property>>


p : Player
pi : Inventory
pit : Item
l : Location
li : Inventory
sword: Item
User
AreYou"sword"
Locate "sword"
No/ False
HasItem "sword"
AreYou "sword"
No/False
null/nill
AreYou"sword"
Locate "sword"
No/ False
Loop
[for each item]
Fetch "sword"
AreYou "sword"
Yes/True
sword
sword
sword


Test Explorer
43 43 0
Search (Ctrl+I)
Test run finished: 43 Tests (43 Passed, 0 Failed, 0 Skipped) run in 2.8 sec
0 Warnings
0 Errors
Test
Duration
Traits
Error Message
TestSwinAdventure (43) 115 ms
TestSwinAdventure (43) 115 ms
TestBag (5) 114 ms
TestIdentifiableObject (6) < 1 ms
TestInventory (4) < 1 ms
TestItem (3) < 1 ms
TestLocation (4) < 1 ms
TestIdentifyItemsInLocation < 1 ms
TestIdentifyItemsInPlayerLocation < 1 ms
TestIdentifyLocation < 1 ms
TestLocationFullDescription < 1 ms
TestLookCommand (14) 1 ms
TestInvalidAt 1 ms
TestInvalidIn < 1 ms
TestInvalidLook < 1 ms
TestLookAtGem < 1 ms
TestLookAtGemInBag < 1 ms
TestLookAtGemInMe < 1 ms
TestLookAtGemInNoBag < 1 ms
TestLookAtMe < 1 ms
TestLookAtNoGemInBag < 1 ms
TestLookAtUnknown < 1 ms
TestPlayerLocation < 1 ms
TestInvalidLength (3) < 1 ms
TestPlayer (7) < 1 ms
TestFullDescription < 1 ms
TestIsIdentifiable < 1 ms
TestLocateItemInLocation < 1 ms
TestLocateItems < 1 ms
TestLocateItself < 1 ms
TestLocateLocation < 1 ms
TestLocateNothing < 1 ms
Group Summary
TestLocation
Tests in group: 4
Outcomes
4 Passed

```
Microsoft Visual Studio Debu

Enter player name:
shah
Enter player description:
student

Command:
look
You are in the hallway
This is the main hallway
In this room you can see:
a key (key)


Command:
look at key
This is a key

Command:
quit

C:\Users\shahn\source\repos\oop\SwinAdventure\SwinAdve
th code 0.
```