Swinburne University of Technology

COS20007 Object Oriented Programming

9.2C - Case Study - Iteration 7 - Paths

PDF generated at 22:48 on Wednesday 29th March, 2023

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

namespace SwinAdventure
{
    public class Path : GameObject
    {
        private Location _destination;
        private bool _isLocked;
        public Path(string[] ids, string name, string desc, Location destination) : base(ids, name, desc)
        {
            _destination = destination;
        }

        public Location Destination => _destination;

        public override string FullDescription => $"At {Name} lies {Destination.Name}";

        public bool IsLocked
        {
            get => _isLocked;
            set => _isLocked = value;
        }
    }
}
```

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

namespace TestSwinAdventure
{
    [TestFixture]
    public class TestPath
    {
        Path path;
        Location garden;

        [SetUp]
        public void Setup()
        {
            garden = new Location("a garden", "This is a garden");
            path = new Path(new string[] { "south", "s" }, "south", "this is south", garden);

        }
        [Test]
        public void TestPathIsIdentifiable()
        {
            Assert.That(path.AreYou("south"), Is.True);
            Assert.That(path.AreYou("s"), Is.True);
            Assert.That(path.AreYou("north"), Is.False);
        }

        [Test]
        public void TestFullDescription()
        {
            string actual = path.FullDescription;
            string expected = "At south lies a garden";

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

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

namespace SwinAdventure
{
    public class Location : GameObject, IHaveInventory
    {
        //local variables
        private Inventory _inventory;
        private List<Path> _paths;

        //constructor
        public Location(string name, string desc) : base(new string[] {"room", "here"}, name, desc)
        {
            _inventory = new Inventory();
            _paths = new List<Path>();
        }

        //methods
        public GameObject Locate(string id)
        {
            //identify itself
            if (AreYou(id))
            {
                return this;
            }
            //identify the items in its inventory
            else if (_inventory.HasItem(id))
            {
                return _inventory.Fetch(id);
            }
            //identify paths in this location
            else if (_paths.Count >= 1)
            {
                foreach (Path p in _paths)
                {
                    if (p.AreYou(id))
                    {
                        return p;
                    }
                }
                return null;
            }
            else return null;
        }

        public void AddPath(Path path)
        {
```

```
            _paths.Add(path);
        }

        //properties
        public string PathList
        {
            get
            {
                string pathList = "";

                if (_paths.Count > 0)
                {
                    for (int i = 0; i < _paths.Count; i++)
                    {
                        if (_paths.Count == 1)
                        {
                            pathList += $"{_paths[i].Name}";
                        }
                        else if (i == _paths.Count - 1)
                        {
                            pathList += $"and {_paths[i].Name}";
                        }
                        else
                        {
                            if (_paths.Count == 1)
                            {
                                pathList += $"{_paths[i].Name}, ";
                            }
                        }
                    }
                    return $"There are exists to the {pathList}.";
                }
                else
                {
                    return "There are no exits.";
                }

            }
        }
        public override string FullDescription
        {
            get
            {
                return $"You are in {Name}\n{Description}\n{PathList}\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;
using Path = SwinAdventure.Path;

namespace TestSwinAdventure
{
    [TestFixture]
    public class TestLocation
    {
        //initialize variables
        Location location;
        Location destination;
        Path path;
        Player player;
        Item sword;

        [SetUp]
        public void Setup ()
        {

            location = new Location ("a garden", "This is a garden");
            destination = new Location("a house", "This is a house");
            path = new Path(new string[] { "south" }, "south", "this is south", destination);
            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 that location can locate its paths
        [Test]
        public void TestIdentifyPath()
        {
            location.AddPath(path);
            Assert.That(player.Locate("south"), Is.SameAs(path));
        }

        //test location's full description
        [Test]
        public void TestLocationFullDescription()
        {
            string actual = location.FullDescription;
            string expected = "You are in a garden\nThis is a garden\nThere are no exits.\nIn this room you can see:\na bronze sword (sword)\n";

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

        [Test]
        public void TestLocationFullDescriptionWithPath()
        {
            location.AddPath(path);
            string actual = location.FullDescription;
            string expected = "You are in a garden\nThis is a garden\nThere are exists to the south.\nIn this room you can see:\na bronze sword (sword)\n";

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

    }
}
```

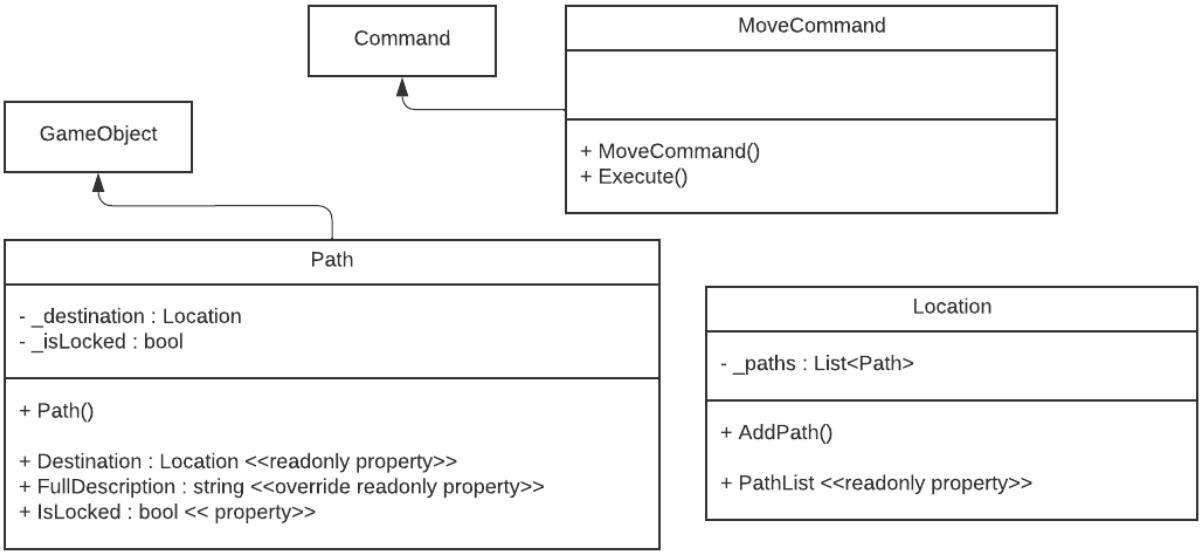
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class MoveCommand : Command
    {
        //constructor with ids for the move command
        public MoveCommand() : base(new string[] { "move", "go", "head", "leave" })
        {

        }

        //methods
        public override string Execute(Player p, string[] text)
        {
            // to store the direction
            string moveTo;
            //array of valid move commands
            string[] moveIds = new string[] { "move", "go", "head", "leave" };

            //error if 3 or more input strings
            if (text.Length >= 3)
            {
                return "I don't know how to move like that";
            }
            //error if first string does not match any move commands
            else if (!moveIds.Contains(text[0]))
            {
                return "Error in move input";
            }
            //error if only 1 string input
            else if (text.Length == 1)
            {
                return "Where do you want to move?";
            }
            else
            {
                // set second string to direction
                moveTo = text[1];

                //check if direction( should be a path identifier) exists in location
                //only check the location the player is in
                //check if the result is a Path object
                if (p.Locate(moveTo) is Path path)
                {
                    p.Move(path);

                    return $"You head {path.Name}\nYou have arrived in
↪ {path.Destination.Name}";
```

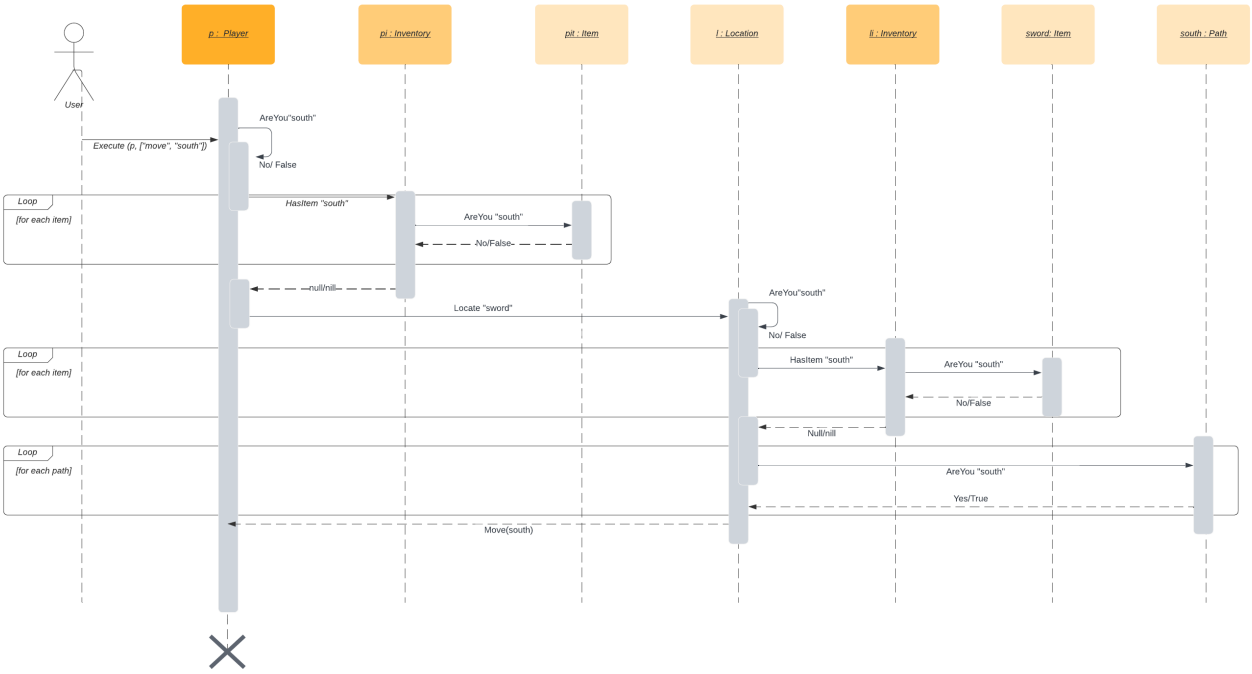
```
                }
                //error if method returns null or returns a different type of GameObject
                else
                {
                    return "Error in direction!";
                }
            }


        }
    }
}
```

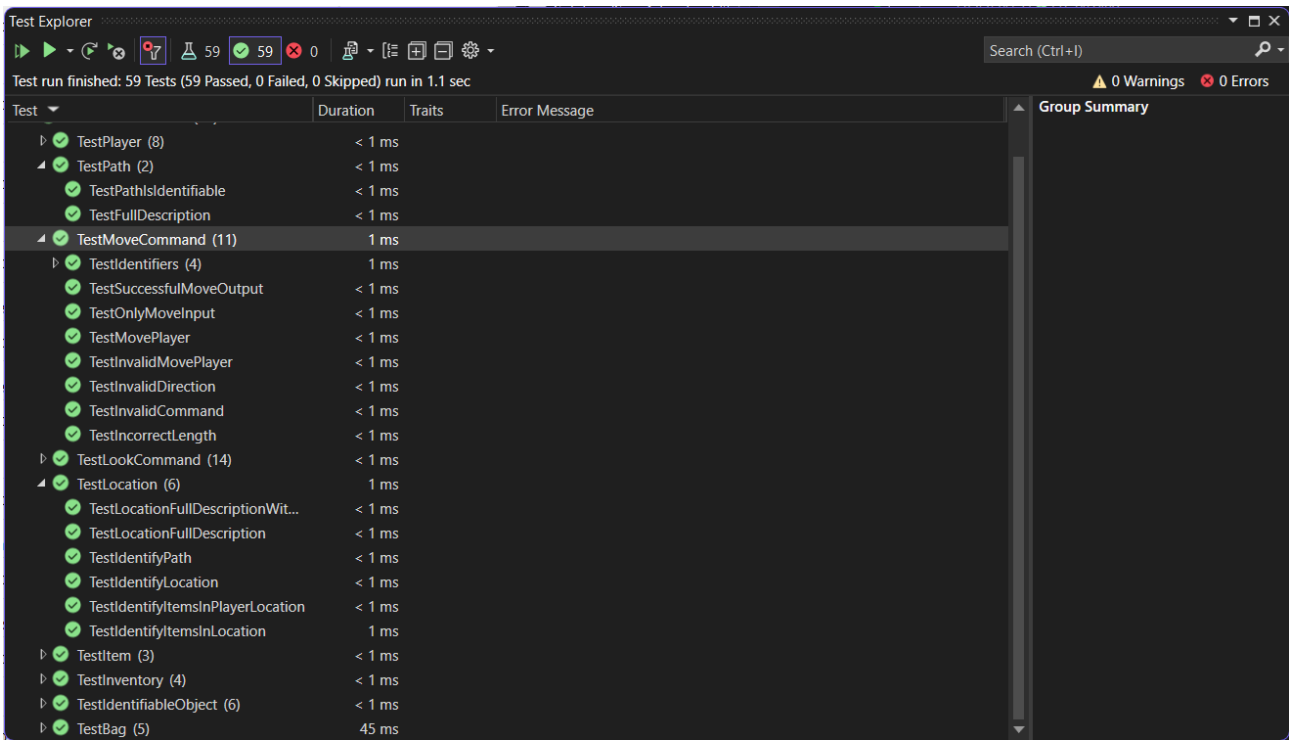
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Path = SwinAdventure.Path;

namespace TestSwinAdventure
{
    public class TestMoveCommand
    {
        MoveCommand move;
        Location location;
        Location destination;
        Path path;
        Player player;

        [SetUp]
        public void Setup()
        {
            move = new MoveCommand();
            location = new Location("a garden", "This is a garden");
            destination = new Location("a house", "This is a house");
            path = new Path(new string[] { "south" }, "south", "this is south",
↪   destination);
            player = new Player("shah", "the student");

            player.Location = location;
            location.AddPath(path);
        }

        [Test]
        public void TestMovePlayer()
        {
            Assert.That(player.Location, Is.SameAs(location));
            move.Execute(player, new string[] { "move", "south" });
            Assert.That(player.Location, Is.SameAs(destination));
        }

        [Test]
        public void TestInvalidMovePlayer()
        {
            Assert.That(player.Location, Is.SameAs(location));
            move.Execute(player, new string[] { "move", "north" });
            Assert.That(player.Location, Is.SameAs(location));
        }

        [TestCase("move")]
        [TestCase("head")]
        [TestCase("go")]
        [TestCase("leave")]
        public void TestIdentifiers(string toTest)
        {
```

```
            move.Execute(player, new string[] { toTest, "south" });
            Assert.That(player.Location, Is.SameAs(destination));
        }

        [Test]
        public void TestSuccessfulMoveOutput()
        {
            string actual = move.Execute(player, new string[] { "move", "south" });
            string expected = "You head south\nYou have arrived in a house";

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

        //errors

        [Test]
        public void TestIncorrectLength()
        {
            string actual = move.Execute(player, new string[] { "move", "at", "north" });
            string expected = "I don't know how to move like that";

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

        [Test]
        public void TestInvalidCommand()
        {
            string actual = move.Execute(player, new string[] { "letsgo", "south" });
            string expected = "Error in move input";

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

        }

        [Test]
        public void TestOnlyMoveInput()
        {
            string actual = move.Execute(player, new string[] { "move" });
            string expected = "Where do you want to move?";

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

        [Test]
        public void TestInvalidDirection()
        {
            string actual = move.Execute(player, new string[] { "move", "north" });
            string expected = "Error in direction!";

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

```
}
```


Command
MoveCommand
+ MoveCommand()
+ Execute()
GameObject
Path
- _destination : Location
- _isLocked : bool
+ Path()
+ Destination : Location <<readonly property>>
+ FullDescription : string <<override readonly property>>
+ IsLocked : bool << property>>
Location
- _paths : List<Path>
+ AddPath()
+ PathList <<readonly property>>


p : Player
pi : Inventory
pit : Item
l : Location
li : Inventory
sword: Item
south : Path
User
AreYou"south"
Execute (p, ["move", "south"])
No/ False
Loop
[for each item]
HasItem "south"
AreYou "south"
No/False
null/nill
AreYou"south"
Locate "sword"
No/ False
Loop
[for each item]
HasItem "south"
AreYou "south"
No/False
Null/nill
Loop
[for each path]
AreYou "south"
Yes/True
Move(south)


Test Explorer
59
59
0
Search (Ctrl+I)
Test run finished: 59 Tests (59 Passed, 0 Failed, 0 Skipped) run in 1.1 sec
0 Warnings
0 Errors
Test
Duration
Traits
Error Message
Group Summary
TestPlayer (8) < 1 ms
TestPath (2) < 1 ms
TestPathIsIdentifiable < 1 ms
TestFullDescription < 1 ms
TestMoveCommand (11) 1 ms
TestIdentifiers (4) 1 ms
TestSuccessfulMoveOutput < 1 ms
TestOnlyMoveInput < 1 ms
TestMovePlayer < 1 ms
TestInvalidMovePlayer < 1 ms
TestInvalidDirection < 1 ms
TestInvalidCommand < 1 ms
TestIncorrectLength < 1 ms
TestLookCommand (14) < 1 ms
TestLocation (6) 1 ms
TestLocationFullDescriptionWit... < 1 ms
TestLocationFullDescription < 1 ms
TestIdentifyPath < 1 ms
TestIdentifyLocation < 1 ms
TestIdentifyItemsInPlayerLocation < 1 ms
TestIdentifyItemsInLocation 1 ms
TestItem (3) < 1 ms
TestInventory (4) < 1 ms
TestIdentifiableObject (6) < 1 ms
TestBag (5) 45 ms

```
C:\Users\shahn\source\repos\

Enter player name:
shah
Enter player description:
the student

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


Command:
move
Where do you want to move?

Command:
move to south
I don't know how to move like that

Command:
move soth
Error in direction!

Command:
move south
You head south
You have arrived in the garden

Command:
look
You are in the garden
This is a garden
There are no exits.
In this room you can see:
```