Swinburne University of Technology

COS20007 Object Oriented Programming

9.2C - Case Study - Iteration 7 - Paths

PDF generated at 15:40 on Monday 15th May, 2023

```
namespace SwinAdventure
{
    public class Path : GameObject
    {
        private Location _startingLocation;
        private Location _endingLocation;
        private bool _isClosed;

        public Path(string[] ids, string name, string desc, Location
↪   startingLocation, Location endingLocation) : base(ids, name, desc)
        {
            _startingLocation = startingLocation;
            _endingLocation = endingLocation;

            Open();
        }

        public Location StartingLocation
        {
            get
            {
                return _startingLocation;
            }
        }
        public Location EndingLocation
        {
            get
            {
                return _endingLocation;
            }
        }

        public bool IsClosed
        {
            get
            {
                return _isClosed;
            }
        }

        public void Close()
        {
            _isClosed = true;
        }
        public void Open()
        {
            _isClosed = false;
        }

        public override string FullDescription
        {
            get
            {
```

```
                return $"{base.FullDescription}, which locates in the {this.FirstId} of {this.StartingLocation.Name}, leading to {this.EndingLocation.Name}.";
            }
        }
    }
}
```

```
namespace SwinAdventure
{
    public class PathTest
    {
        private Location _studio;
        private Location _closet;

        private Path _testPath;

        [SetUp] public void SetUp()
        {
            _studio = new Location(new string[] { "studio" }, "a studio", "A small, beautiful and fully-furnished studio");
            _closet = new Location(new string[] { "closet" }, "a closet", "A small dark closet, with an odd smell");

            _testPath = new Path(new string[] { "east", "e" }, "first door", "The first small door", _studio, _closet);
        }

        [Test] public void TestPathStartingLocation()
        {
            Assert.AreEqual(_studio, _testPath.StartingLocation);
        }
        [Test] public void TestPathEndingLocation()
        {
            Assert.AreEqual(_closet, _testPath.EndingLocation);
        }
        [Test] public void TestPathClose()
        {
            _testPath.Close();
            Assert.IsTrue(_testPath.IsClosed);
        }
        [Test]
        public void TestPathOpen()
        {
            _testPath.Close();
            _testPath.Open();
            Assert.IsFalse(_testPath.IsClosed);
        }
        [Test]
        public void TestPathFullDescription()
        {
            Assert.AreEqual(_testPath.FullDescription, $"The first small door, which locates in the {_testPath.FirstId} of {_testPath.StartingLocation.Name}, leading to {_testPath.EndingLocation.Name}.");
        }
    }
}
```

```
using System;

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

        public Location(string[] ids, string name, string desc) : base(ids, name, desc)
        {
            _inventory = new Inventory();
            _paths = new List<Path>();

            AddIdentifier("room");
            AddIdentifier("here");
        }

        public GameObject Locate(string id)
        {
            if (AreYou(id))
            {
                return this;
            }

            foreach (Path path in _paths)
            {
                if (path.AreYou(id))
                {
                    return path;
                }
            }

            return _inventory.Fetch(id);
        }

        // Locations will need to be identifiable and have a name, and description.
        public override string FullDescription
        {
            get
            {
                return $"You are in {Name}\n{base.FullDescription}\nIn this room you can see:\n{Inventory.ItemList}\n{PathList}";
            }
        }

        // Location can contain items
        public Inventory Inventory
        {
            get
            {
                return _inventory;
```

```
            }
        }

        public void AddPath(Path path)
        {
            _paths.Add(path);
        }

        public string PathList
        {
            get
            {
                string pList = "";

                if (_paths.Count > 0)
                {
                    if (_paths.Count == 1)
                    {
                        pList += $"There is an exit to the {_paths[0].FirstId}.";
                    }
                    else
                    {
                        pList += "There are exits to the ";
                        for (int i = 0; i < (_paths.Count - 1); i++)
                        {
                            pList += $"{_paths[i].FirstId}, ";
                        }
                        pList += $"and {_paths[_paths.Count - 1].FirstId}.";
                    }
                }
                else
                {
                    pList += "There is no exit from this room.";
                }

                return pList;
            }
        }
    }
}
```

```
namespace SwinAdventure
{
    public class LocationTest
    {
        private Location _testLocation;
        private Location _garden;
        private Location _closet;

        private Path _door1;
        private Path _door2;
        private Path _window;

        private Item _sword;
        private Item _shovel;
        private Item _pc;

        [SetUp]
        public void SetUp()
        {
            _testLocation = new Location(new string[] { "studio" }, "a studio", "A small, beautiful and fully-furnished studio.");

            _garden = new Location(new string[] { "garden" }, "a garden", "A small, beautiful and garden");
            _closet = new Location(new string[] { "studio" }, "a closet", "A small dark closet, with an odd smell");

            _door1 = new Path(new string[] { "north", "n" }, "first door", "The first small door", _testLocation, _garden);
            _door2 = new Path(new string[] { "southeast", "se" }, "second door", "The second medium door", _testLocation, _closet);
            _window = new Path(new string[] { "east", "e" }, "window", "The tiny window", _testLocation, _closet);

            _testLocation.AddPath(_door1);
            _testLocation.AddPath(_door2);

            _sword = new Item(new string[] { "sword", "bronze" }, "a bronze sword", "This is a bronze sword");
            _shovel = new Item(new string[] { "shovel" }, "a shovel", "This is a shovel");
            _pc = new Item(new string[] { "pc", "computer" }, "a small computer", "This is a small computer");

            _testLocation.Inventory.Put(_shovel);
            _testLocation.Inventory.Put(_pc);
        }

        // Locations can identify themselves
        [Test]
        public void TestLocationLocateItself()
        {
            Assert.AreEqual(_testLocation.Locate("room"), _testLocation);
```

```
            Assert.AreEqual(_testLocation.Locate("here"), _testLocation);
        }

        // Locations can locate items they have
        [Test]
        public void TestLocationLocateItems()
        {
            // Locate items that are in the location inventory
            Assert.AreEqual(_testLocation.Locate("shovel"), _shovel);
            Assert.AreEqual(_testLocation.Locate("pc"), _pc);

            // Locate item that is not in the location inventory
            Assert.AreEqual(_testLocation.Locate("sword"), null);
        }

        [Test]
        public void TestLocationFullDescription()
        {
            Assert.AreEqual(_testLocation.FullDescription, $"You are in {_testLocation.Name}\nA small, beautiful and fully-furnished studio.\nIn this room you can see:\n{_testLocation.Inventory.ItemList}\n{_testLocation.PathList}");
        }

        [Test]
        public void TestLocationPathList()
        {
            Assert.AreEqual(_testLocation.PathList, $"There are exits to the {_door1.FirstId}, and {_door2.FirstId}.");
        }

        [Test]
        public void TestLocationLocatePaths()
        {
            // Test if there are two doors in the test room
            Assert.AreEqual(_testLocation.Locate("north"), _door1);
            Assert.AreEqual(_testLocation.Locate("se"), _door2);

            // Test if there is no way to the east of the test room
            Assert.AreEqual(_testLocation.Locate("east"), null);
        }

        [Test]
        public void TestLocationAddsPath()
        {
            // Test if there is no way to the east of the test room
            Assert.AreEqual(_testLocation.Locate("east"), null);

            // Adding window to the east of the test room
            _testLocation.AddPath(_window);
            Assert.AreEqual(_testLocation.Locate("east"), _window);
        }
    }
```

```
}
```

```
using System;

namespace SwinAdventure
{
    public class MoveCommand : Command
    {
        public MoveCommand() : base(new string[] { "move", "go" })
        {
        }

        public override string Execute(Player p, string[] text)
        {
            if ((text.Length == 0) || (text.Length > 2))
            {
                return "Error in move input.";
            }

            if ((text[0] != "move") && (text[0] != "go"))
            {
                return "Error in move input.";
            }

            if (text.Length == 1)
            {
                return "Which direction do you want to move to?";
            }

            GameObject gameObject = p.Location.Locate(text[1]);
            if (gameObject != null)
            {
                if (gameObject.GetType() == typeof(Path))
                {
                    Path path = (Path)gameObject;

                    if (path.StartingLocation != p.Location)
                    {
                        return $"Could not move from {path.StartingLocation.Name}.";
                    }
                    else if (path.EndingLocation == null)
                    {
                        return $"Could not move.";
                    }
                    else if (path.IsClosed)
                    {
                        return $"The path {path.Name} is closed";
                    }

                    p.Move(path);
                    /* The "Move" method is in the class "Player":
                     *      public void Move(Path path)
                     *      {
                     *          this.Location = path.EndingLocation;
                     *      }
```

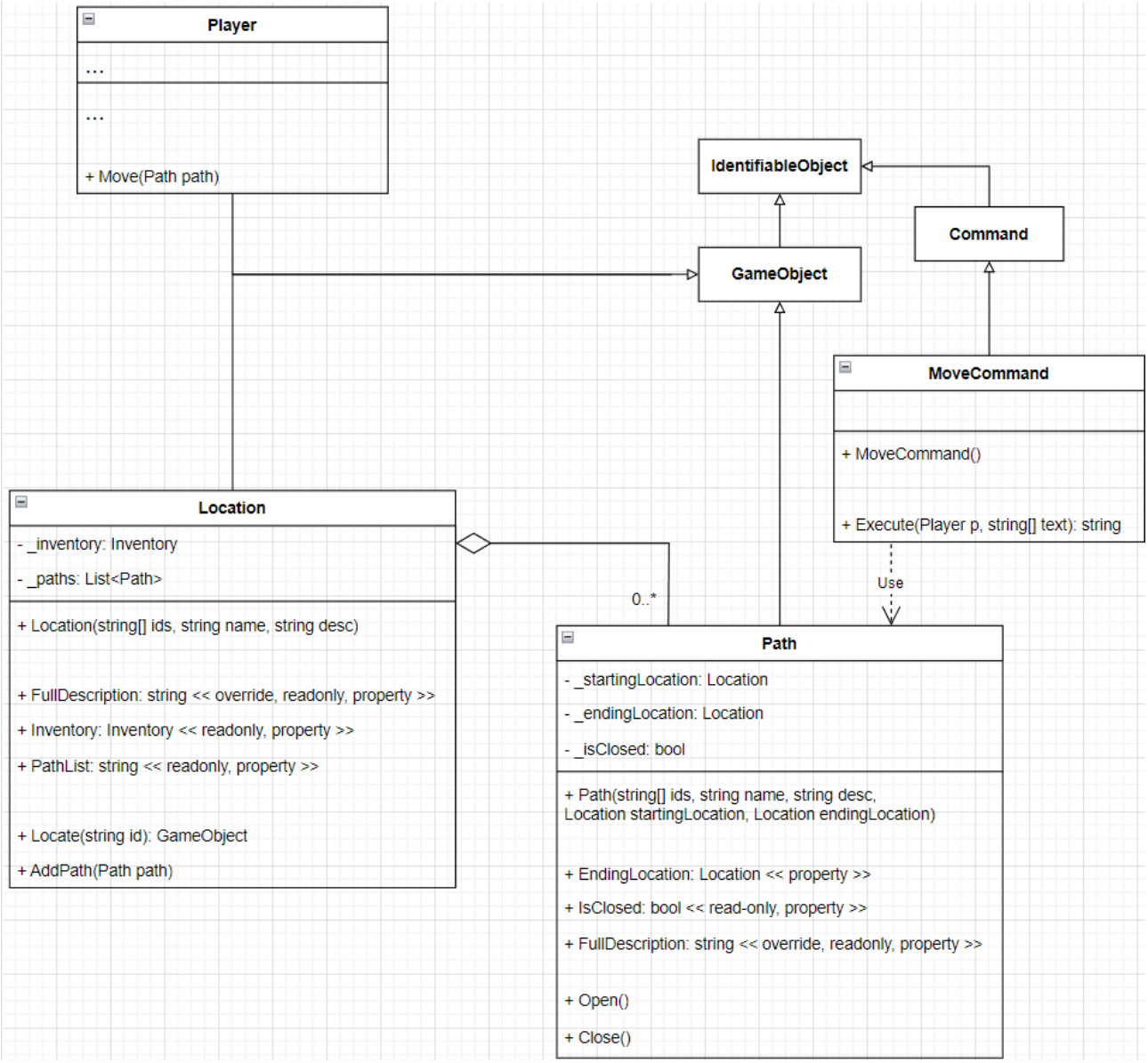
```
                    */
                    return $"You head {path.FirstId}\nYou go through {path.FullDescription}\nYou have arrived {path.EndingLocation.Name}.";
                }

                return $"Could not find the {gameObject.Name}.";
            }

            return "Could not find the path.";
        }
    }
}
```

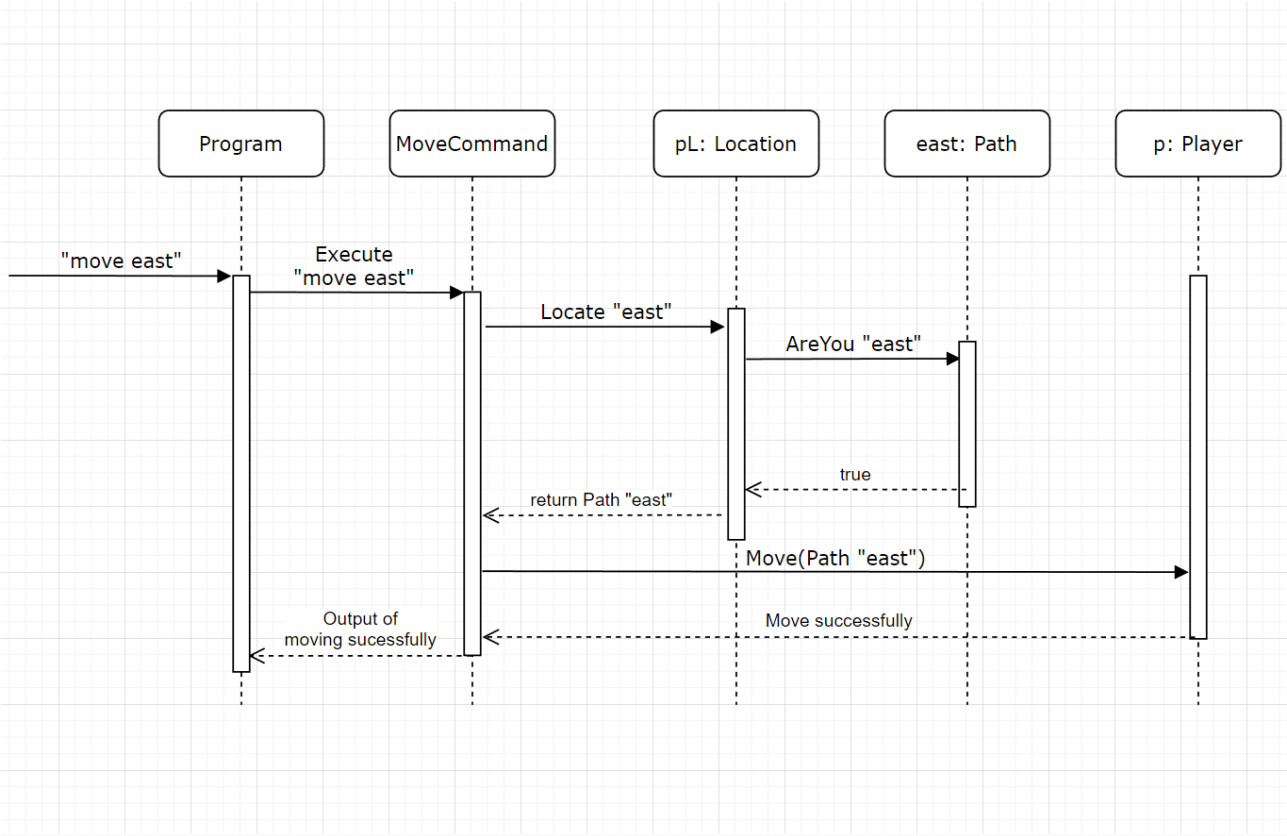
```
using System.IO;

namespace SwinAdventure
{
    public class MoveCommandTest
    {
        private MoveCommand _testMoveCommand;
        private Player _testPlayer;

        private Location _studio, _closet, _garden;
        private Path _studioDoor1, _studioDoor2, _studioWindow, _closetDoor, _closetWindow;

        [SetUp]
        public void SetUp()
        {
            _testMoveCommand = new MoveCommand();

            _testPlayer = new Player("Trung Kien Nguyen", "I am the player");

            _studio = new Location(new string[] { "studio" }, "a studio", "A small, beautiful and fully-furnished studio");
            _closet = new Location(new string[] { "closet" }, "a closet", "A small dark closet, with an odd smell");
            _garden = new Location(new string[] { "garden" }, "a garden", "A large and beautiful garden");

            _studioDoor1 = new Path(new string[] { "east", "e" }, "first door", "The first small door", _studio, _closet);
            _studioDoor2 = new Path(new string[] { "south", "s" }, "second door", "The second large door", _studio, _garden);
            _studioWindow = new Path(new string[] { "north", "n" }, "window", "The large window", _studio, null);
            _closetDoor = new Path(new string[] { "west", "w" }, "door", "The small door", _closet, _studio);
            _closetWindow = new Path(new string[] { "southwest", "sw" }, "window", "The large window", _closet, _garden);

            _studioDoor2.Close();

            _studio.AddPath(_studioDoor1);
            _studio.AddPath(_studioDoor2);
            _studio.AddPath(_studioWindow);
            _closet.AddPath(_closetDoor);
            _closet.AddPath(_closetWindow);
        }

        [Test]
        public void TestMoveInputError()
        {
            _testPlayer.Location = _studio;

            string errorMessage = "Error in move input.";
```

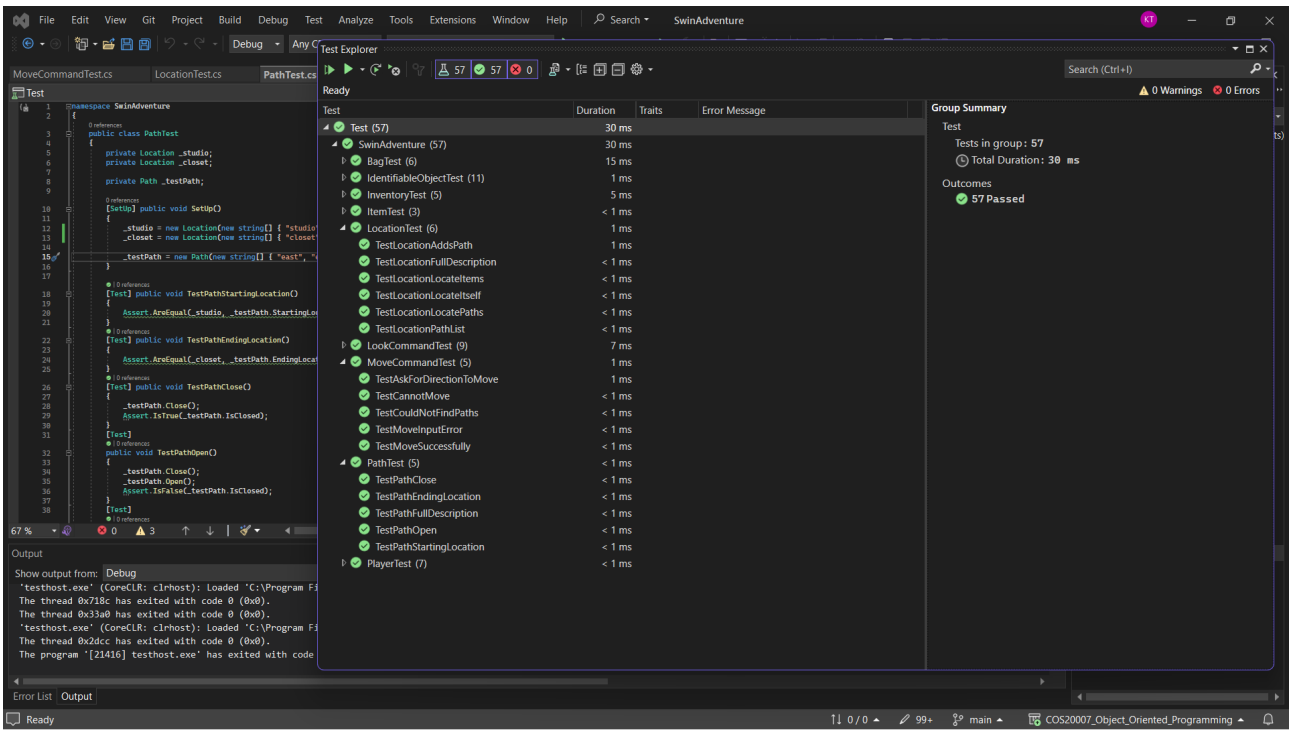
```

            // Invalid input's number of keywords
            Assert.AreEqual(errorMessage, _testMoveCommand.Execute(_testPlayer, new string[] { "" }));
            Assert.AreEqual(errorMessage, _testMoveCommand.Execute(_testPlayer, new string[] { "move", "to", "west" }));

            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);

            // Invalid first keyword
            Assert.AreEqual(errorMessage, _testMoveCommand.Execute(_testPlayer, new string[] { "movee" }));
            Assert.AreEqual(errorMessage, _testMoveCommand.Execute(_testPlayer, new string[] { "gooo" }));

            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);
        }

        [Test]
        public void TestAskForDirectionToMove()
        {
            _testPlayer.Location = _studio;

            string askingMessage = "Which direction do you want to move to?";

            // No direction
            Assert.AreEqual(askingMessage, _testMoveCommand.Execute(_testPlayer, new string[] { "move" }));
            Assert.AreEqual(askingMessage, _testMoveCommand.Execute(_testPlayer, new string[] { "go" }));

            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);
        }

        [Test]
        public void TestCannotMove()
        {
            _testPlayer.Location = _studio;

            // Path is closed
            Assert.AreEqual("The path second door is closed", _testMoveCommand.Execute(_testPlayer, new string[] { "move", "south" }));

            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);

            // Path does not belongs to the current room (studio)
            _studio.AddPath(_closetWindow);
            Assert.AreEqual($"Could not move from {_closetDoor.StartingLocation.Name}.", _testMoveCommand.Execute(_testPlayer, new string[] { "move", "southwest" }));
```

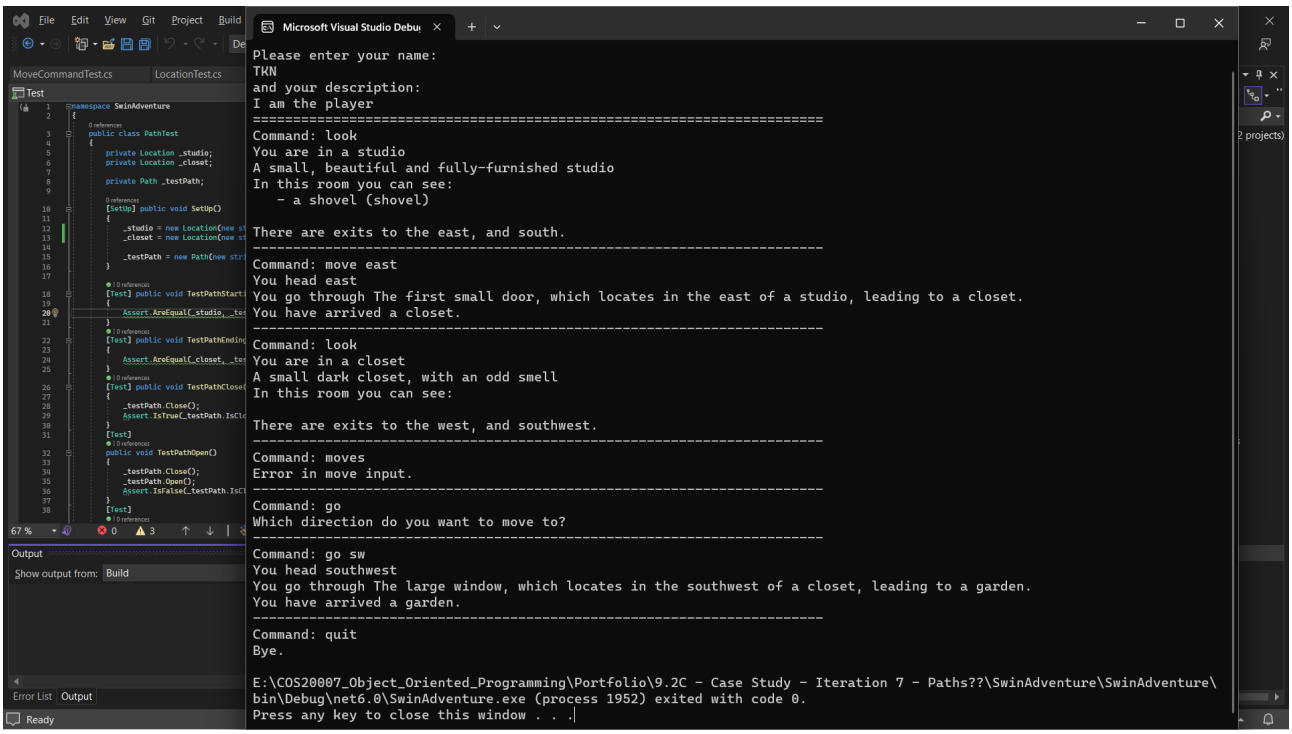
```
            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);

            // Path has no destinations
            Assert.AreEqual("Could not move.", _testMoveCommand.Execute(_testPlayer, new string[] { "go", "north" }));

            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);
        }

        [Test]
        public void TestCouldNotFindPaths()
        {
            _testPlayer.Location = _studio;

            // Location locates a GameObject that is not a Path
            Item sword = new Item(new string[] { "sword" }, "a bronze sword", "This is a bronze sword");
            _studio.Inventory.Put(sword);
            Assert.AreEqual($"Could not find the {sword.Name}.", _testMoveCommand.Execute(_testPlayer, new string[] { "move", "sword" }));

            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);

            // Location locates nothing
            Assert.AreEqual("Could not find the path.", _testMoveCommand.Execute(_testPlayer, new string[] { "go", "west" }));
            Assert.AreEqual("Could not find the path.", _testMoveCommand.Execute(_testPlayer, new string[] { "go", "southwest" }));
            Assert.AreEqual("Could not find the path.", _testMoveCommand.Execute(_testPlayer, new string[] { "go", "gun" }));

            // and the player remains in the same location
            Assert.AreEqual(_testPlayer.Location, _studio);
        }

        [Test]
        public void TestMoveSuccessfully()
        {
            _testPlayer.Location = _studio;

            // Move east, from the studio to the closet through the small door
            Assert.AreEqual($"You head {_studioDoor1.FirstId}\nYou go through {_studioDoor1.FullDescription}\nYou have arrived {_closet.Name}.", _testMoveCommand.Execute(_testPlayer, new string[] { "move", "east" }));
            Assert.AreEqual(_testPlayer.Location, _closet);

            // Move southwest, from the closet to the garden through the window
            Assert.AreEqual($"You head {_closetWindow.FirstId}\nYou go through {_closetWindow.FullDescription}\nYou have arrived {_garden.Name}.", _testMoveCommand.Execute(_testPlayer, new string[] { "go", "sw" }));
```

```
            Assert.AreEqual(_testPlayer.Location, _garden);
        }
    }
}
```


Player
...
...
+ Move(Path path)
IdentifiableObject
Command
GameObject
MoveCommand
+ MoveCommand()
+ Execute(Player p, string[] text): string
Use
Location
- _inventory: Inventory
- _paths: List<Path>
+ Location(string[] ids, string name, string desc)
+ FullDescription: string << override, readonly, property >>
+ Inventory: Inventory << readonly, property >>
+ PathList: string << readonly, property >>
+ Locate(string id): GameObject
+ AddPath(Path path)
0..*
Path
- _startingLocation: Location
- _endingLocation: Location
- _isClosed: bool
+ Path(string[] ids, string name, string desc,
Location startingLocation, Location endingLocation)
+ EndingLocation: Location << property >>
+ IsClosed: bool << read-only, property >>
+ FullDescription: string << override, readonly, property >>
+ Open()
+ Close()


Program
MoveCommand
pL: Location
east: Path
p: Player
"move east"
Execute
"move east"
Locate "east"
AreYou "east"
true
return Path "east"
Move(Path "east")
Output of
moving sucessfully
Move successfully


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
SwinAdventure
Debug
MoveCommandTest.cs
LocationTest.cs
PathTest.cs
Test
namespace SwinAdventure
{
public class PathTest
{
private Location _studio;
private Location _closet;
private Path _testPath;
[SetUp] public void SetUp()
{
_studio = new Location(new string[] { "studio"
_closet = new Location(new string[] { "closet"
_testPath = new Path(new string[] { "east", "
}
[Test] public void TestPathStartingLocation()
{
Assert.AreEqual(_studio, _testPath.StartingLo
}
[Test] public void TestPathEndingLocation()
{
Assert.AreEqual(_closet, _testPath.EndingLoca
}
[Test] public void TestPathClose()
{
_testPath.Close();
Assert.IsTrue(_testPath.IsClosed);
}
[Test]
public void TestPathOpen()
{
_testPath.Close();
_testPath.Open();
Assert.IsFalse(_testPath.IsClosed);
}
[Test]
67 %
Output
Show output from: Debug
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Fi
The thread 0x718c has exited with code 0 (0x0).
The thread 0x33a0 has exited with code 0 (0x0).
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Fi
The thread 0x2dcc has exited with code 0 (0x0).
The program '[21416] testhost.exe' has exited with code
Error List Output
Test Explorer
57 57 0
Search (Ctrl+I)
Ready
0 Warnings 0 Errors
Test Duration Traits Error Message
Test (57) 30 ms
SwinAdventure (57) 30 ms
BagTest (6) 15 ms
IdentifiableObjectTest (11) 1 ms
InventoryTest (5) 5 ms
ItemTest (3) < 1 ms
LocationTest (6) 1 ms
TestLocationAddsPath 1 ms
TestLocationFullDescription < 1 ms
TestLocationLocateItems < 1 ms
TestLocationLocateItself < 1 ms
TestLocationLocatePaths < 1 ms
TestLocationPathList < 1 ms
LookCommandTest (9) 7 ms
MoveCommandTest (5) 1 ms
TestAskForDirectionToMove 1 ms
TestCannotMove < 1 ms
TestCouldNotFindPaths < 1 ms
TestMoveInputError < 1 ms
TestMoveSuccessfully < 1 ms
PathTest (5) < 1 ms
TestPathClose < 1 ms
TestPathEndingLocation < 1 ms
TestPathFullDescription < 1 ms
TestPathOpen < 1 ms
TestPathStartingLocation < 1 ms
PlayerTest (7) < 1 ms
Group Summary
Test
Tests in group: 57
Total Duration: 30 ms
Outcomes
57 Passed
Ready
0 / 0
99+
main
COS20007_Object_Oriented_Programming


Microsoft Visual Studio Debu
Please enter your name:
TKN
and your description:
I am the player
==========
Command: look
You are in a studio
A small, beautiful and fully-furnished studio
In this room you can see:
- a shovel (shovel)
There are exits to the east, and south.

Command: move east
You head east
You go through The first small door, which locates in the east of a studio, leading to a closet.
You have arrived a closet.

Command: look
You are in a closet
A small dark closet, with an odd smell
In this room you can see:
There are exits to the west, and southwest.

Command: moves
Error in move input.

Command: go
Which direction do you want to move to?

Command: go sw
You head southwest
You go through The large window, which locates in the southwest of a closet, leading to a garden.
You have arrived a garden.

Command: quit
Bye.
E:\COS20007_Object_Oriented_Programming\Portfolio\9.2C - Case Study - Iteration 7 - Paths??\SwinAdventure\SwinAdventure\bin\Debug\net6.0\SwinAdventure.exe (process 1952) exited with code 0.
Press any key to close this window . . .