Swinburne University of Technology

COS20007 Object Oriented Programming

Case Study - Iteration 7 - Paths

PDF generated at 18:47 on Sunday 19th November, 2023

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace SwinAdventure
{
    public class Path : GameObject
    {
        private Location _destination;

        public Path(string[] ids, string name, string desc, Location destination) :
    ↪   base(ids, name, desc)
        {
            _destination = destination;
        }

        public Location SetDestination()
        {
            return _destination;
        }

        public override string FullDescription
        {
            get
            {
                return "You head " + this.Name +
                       "\nYou travel through a " + base.FullDescription +
                       "\nYou have arrived in the " + _destination.Name;
            }
        }
    }
}
```

```
using System;
using SwinAdventure;

namespace SwinAdventureTest
{
    [TestFixture]
    public class PathTests
    {
        Player player;
        Location hall, garden, lab;
        SwinAdventure.Path pathHtoL, pathHtoG, pathGtoH, pathGtoL, pathLtoH,
    ↪   pathLtoG;
        MoveCommand move;

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

            player = new Player("Vu", "The strong player");
            hall = new Location(new string[] { "hallway" }, "Hallway", "This is a
    ↪   long well lit Hallway");
            garden = new Location(new string[] { "garden" }, "Garden", "This is a big
    ↪   garden with a lot of secret spots");
            lab = new Location(new string[] { "lab" }, "Laboratory", "This is where
    ↪   the magic is created");

            pathHtoL = new SwinAdventure.Path(new string[] { "south", "s", "down" },
    ↪   "South", "slide", lab);
            pathHtoG = new SwinAdventure.Path(new string[] { "east", "e" }, "East",
    ↪   "small door", garden);

            pathGtoH = new SwinAdventure.Path(new string[] { "west", "w" }, "West",
    ↪   "small door", hall);
            pathGtoL = new SwinAdventure.Path(new string[] { "sw", "south west" },
    ↪   "South West", "roller coaster", lab);

            pathLtoH = new SwinAdventure.Path(new string[] { "n", "north", "up" },
    ↪   "North", "ladder", hall);
            pathLtoG = new SwinAdventure.Path(new string[] { "ne", "north west" },
    ↪   "North West", "roller coaster", garden);

            hall.AddPath(pathHtoG);
            hall.AddPath(pathHtoL);

            garden.AddPath(pathGtoL);
            garden.AddPath(pathGtoH);

            lab.AddPath(pathLtoG);
            lab.AddPath(pathLtoH);

            player.Location = hall;
        }

        [Test]
        public void TestMovePlayer()
        {
            string expected = "You head South\nYou travel through a slide\nYou have
    ↪   arrived in the Laboratory";
            Assert.AreEqual(expected, move.Execute(player, new string[] { "move",
    ↪   "south" }), "Player cannot be moved south");
        }

        [Test]
        public void TestGetPath()
        {
            Assert.AreEqual(pathGtoL, garden.GetPath("sw"));
        }

        [Test]
        public void TestLeaveLocation()
        {

            move.Execute(player, new string[] { "move", "south" }); //player is now
    ↪   in the lab
            Assert.AreEqual("You have headed back to the Hallway",
    ↪   move.Execute(player, new string[] { "leave" }));
        }

        [Test]
        public void TestInvalidMove()
        {
            player.Location = hall;

            Assert.AreEqual("This location has no path in the somewhere direction",
    ↪   move.Execute(player, new string[] { "move", "somewhere" }));
            Assert.AreEqual(hall, player.Location, "Player's location changed after
    ↪   an invalid move");

            Assert.AreEqual("Where do you want to go?", move.Execute(player, new
    ↪   string[] { "move" }));
            Assert.AreEqual(hall, player.Location, "Player's location changed after
    ↪   an invalid move");

            move.Execute(player, new string[] { "go", "east" });
            move.Execute(player, new string[] { "head", "south west" });

            Assert.AreEqual("This location has no path in the south direction",
    ↪   move.Execute(player, new string[] { "move", "south" }));
            Assert.AreEqual(lab, player.Location, "Player's location changed after an
    ↪   invalid move");

            Assert.AreEqual("Where do you want to go?", move.Execute(player, new
    ↪   string[] { "move" }));
            Assert.AreEqual(lab, player.Location, "Player's location changed after an
    ↪   invalid move");
```

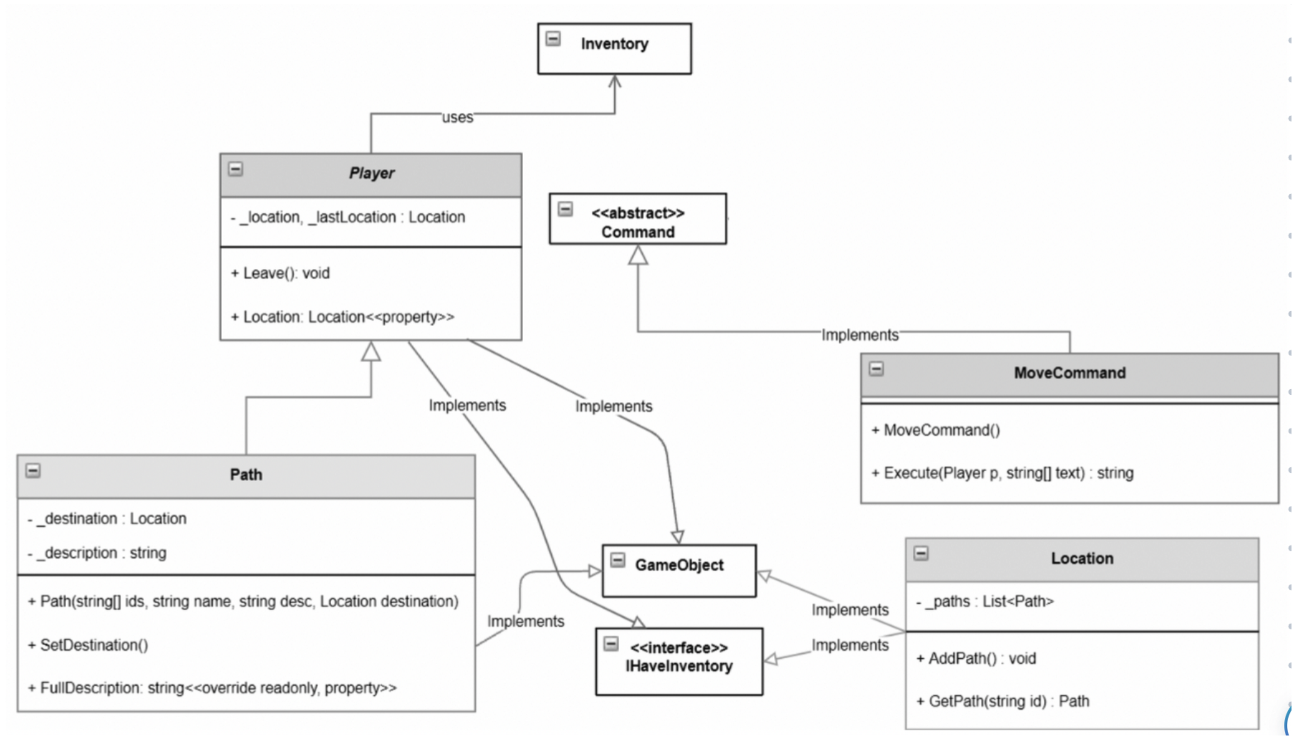
```
        }
    }
}
```

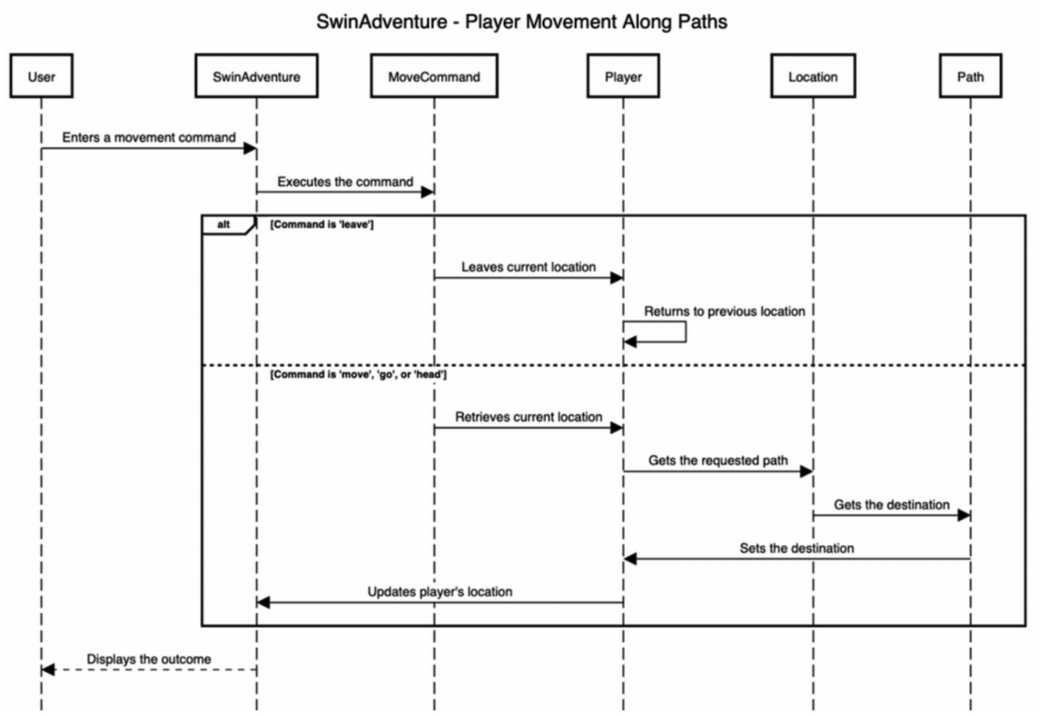
```
using System;
using System.Collections.Generic;
using System.Linq;

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

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

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

            GameObject item = _inventory.Fetch(id);
            if (item != null)
            {
                return item;
            }

            return _inventory.Locate(id);
        }

        public Inventory Inventory
        {
            get { return _inventory; }
        }

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

        public Path GetPath(string id)
        {
            foreach (Path p in _paths)
            {
                if (p.AreYou(id))
                {
                    return p;
                }
            }

            return null;
        }

        public override string FullDescription
        {
            get
            {
                string paths = "";
                int count = 0;
                foreach (Path p in _paths)
                {
                    if (count == 0)
                    {
                        paths = p.Name.ToLower();
                    }
                    else
                    {
                        paths = paths + " and " + p.Name.ToLower();
                    }
                    count++;
                }

                string fullDesc = "You are in the " + Name + "\n" +
    ↪ base.FullDescription
                                + "\nThere are exits to the " + paths //addition of
    ↪ exits available
                                + "\nIn this room you can see:\n" +
    ↪ _inventory.ItemList;

                return fullDesc;
            }
        }
    }
}

```

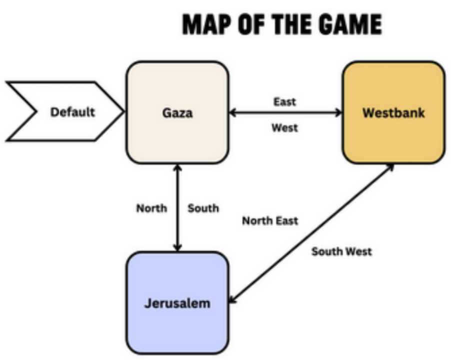
```
using System;
using SwinAdventure;

namespace SwinAdventureTest
{
    [TestFixture]
    public class PathTests
    {
        Player player;
        Location hall, garden, lab;
        SwinAdventure.Path pathHtoL, pathHtoG, pathGtoH, pathGtoL, pathLtoH, pathLtoG;
        MoveCommand move;

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

            player = new Player("Vu", "The strong player");
            hall = new Location(new string[] { "hallway" }, "Hallway", "This is a long well lit Hallway");
            garden = new Location(new string[] { "garden" }, "Garden", "This is a big garden with a lot of secret spots");
            lab = new Location(new string[] { "lab" }, "Laboratory", "This is where the magic is created");

            pathHtoL = new SwinAdventure.Path(new string[] { "south", "s", "down" }, "South", "slide", lab);
            pathHtoG = new SwinAdventure.Path(new string[] { "east", "e" }, "East", "small door", garden);

            pathGtoH = new SwinAdventure.Path(new string[] { "west", "w" }, "West", "small door", hall);
            pathGtoL = new SwinAdventure.Path(new string[] { "sw", "south west" }, "South West", "roller coaster", lab);

            pathLtoH = new SwinAdventure.Path(new string[] { "n", "north", "up" }, "North", "ladder", hall);
            pathLtoG = new SwinAdventure.Path(new string[] { "ne", "north west" }, "North West", "roller coaster", garden);

            hall.AddPath(pathHtoG);
            hall.AddPath(pathHtoL);

            garden.AddPath(pathGtoL);
            garden.AddPath(pathGtoH);

            lab.AddPath(pathLtoG);
            lab.AddPath(pathLtoH);

            player.Location = hall;
        }

        [Test]
        public void TestMovePlayer()
        {
            string expected = "You head South\nYou travel through a slide\nYou have arrived in the Laboratory";
            Assert.AreEqual(expected, move.Execute(player, new string[] { "move", "south" }), "Player cannot be moved south");
        }

        [Test]
        public void TestGetPath()
        {
            Assert.AreEqual(pathGtoL, garden.GetPath("sw"));
        }

        [Test]
        public void TestLeaveLocation()
        {

            move.Execute(player, new string[] { "move", "south" }); //player is now in the lab
            Assert.AreEqual("You have headed back to the Hallway", move.Execute(player, new string[] { "leave" }));
        }

        [Test]
        public void TestInvalidMove()
        {
            player.Location = hall;

            Assert.AreEqual("This location has no path in the somewhere direction", move.Execute(player, new string[] { "move", "somewhere" }));
            Assert.AreEqual(hall, player.Location, "Player's location changed after an invalid move");

            Assert.AreEqual("Where do you want to go?", move.Execute(player, new string[] { "move" }));
            Assert.AreEqual(hall, player.Location, "Player's location changed after an invalid move");

            move.Execute(player, new string[] { "go", "east" });
            move.Execute(player, new string[] { "head", "south west" });

            Assert.AreEqual("This location has no path in the south direction", move.Execute(player, new string[] { "move", "south" }));
            Assert.AreEqual(lab, player.Location, "Player's location changed after an invalid move");

            Assert.AreEqual("Where do you want to go?", move.Execute(player, new string[] { "move" }));
            Assert.AreEqual(lab, player.Location, "Player's location changed after an invalid move");
        }
    }
}
```

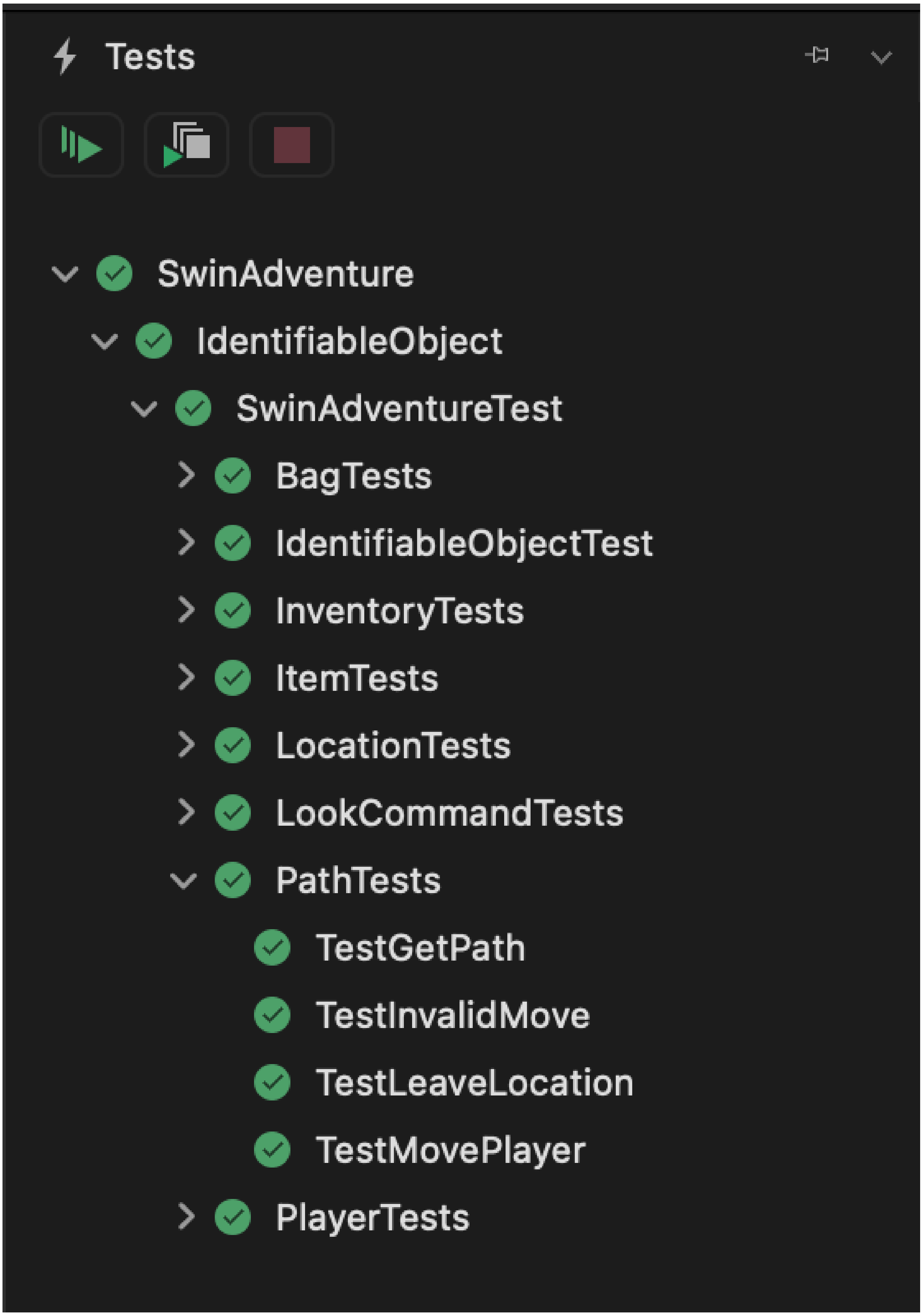
```
using System;
using System.Collections.Generic;
using System.Linq;

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

        }

        public override string Execute(Player p, string[] text)
        {
            if (text.Length != 2 && text.Length != 1)
            {
                return "I don't know how to move like that";
            }

            if (text[0] != "leave" && text.Length == 1)
            {
                return "Where do you want to go?";
            }

            if (text[0] == "leave")
            {
                p.Leave();
                string message;
                message = "You have headed back to the " + p.Location.Name;
                return message;
            }
            else
            {
                Location _location = p.Location;
                Path _path = _location.GetPath(text[1]);

                if (_path == null)
                {
                    return "This location has no path in the " + text[1] + "
↪   direction";
                }

                _location = _path.SetDestination();

                if (_location == null)
                {
                    return "Location not found";
                }

                p.Location = _location;

                return _path.FullDescription;
                }
            }
        }
}
```

```
using System;
using SwinAdventure;

namespace SwinAdventureTest
{
    [TestFixture]
    public class PathTests
    {
        Player player;
        Location hall, garden, lab;
        SwinAdventure.Path pathHtoL, pathHtoG, pathGtoH, pathGtoL, pathLtoH,
    ↪   pathLtoG;
        MoveCommand move;

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

            player = new Player("Vu", "The strong player");
            hall = new Location(new string[] { "hallway" }, "Hallway", "This is a
    ↪   long well lit Hallway");
            garden = new Location(new string[] { "garden" }, "Garden", "This is a big
    ↪   garden with a lot of secret spots");
            lab = new Location(new string[] { "lab" }, "Laboratory", "This is where
    ↪   the magic is created");

            pathHtoL = new SwinAdventure.Path(new string[] { "south", "s", "down" },
    ↪   "South", "slide", lab);
            pathHtoG = new SwinAdventure.Path(new string[] { "east", "e" }, "East",
    ↪   "small door", garden);

            pathGtoH = new SwinAdventure.Path(new string[] { "west", "w" }, "West",
    ↪   "small door", hall);
            pathGtoL = new SwinAdventure.Path(new string[] { "sw", "south west" },
    ↪   "South West", "roller coaster", lab);

            pathLtoH = new SwinAdventure.Path(new string[] { "n", "north", "up" },
    ↪   "North", "ladder", hall);
            pathLtoG = new SwinAdventure.Path(new string[] { "ne", "north west" },
    ↪   "North West", "roller coaster", garden);

            hall.AddPath(pathHtoG);
            hall.AddPath(pathHtoL);

            garden.AddPath(pathGtoL);
            garden.AddPath(pathGtoH);

            lab.AddPath(pathLtoG);
            lab.AddPath(pathLtoH);

            player.Location = hall;
        }

        [Test]
        public void TestMovePlayer()
        {
            string expected = "You head South\nYou travel through a slide\nYou have arrived in the Laboratory";
            Assert.AreEqual(expected, move.Execute(player, new string[] { "move", "south" }), "Player cannot be moved south");
        }

        [Test]
        public void TestGetPath()
        {
            Assert.AreEqual(pathGtoL, garden.GetPath("sw"));
        }

        [Test]
        public void TestLeaveLocation()
        {

            move.Execute(player, new string[] { "move", "south" }); //player is now in the lab
            Assert.AreEqual("You have headed back to the Hallway", move.Execute(player, new string[] { "leave" }));
        }

        [Test]
        public void TestInvalidMove()
        {
            player.Location = hall;

            Assert.AreEqual("This location has no path in the somewhere direction", move.Execute(player, new string[] { "move", "somewhere" }));
            Assert.AreEqual(hall, player.Location, "Player's location changed after an invalid move");

            Assert.AreEqual("Where do you want to go?", move.Execute(player, new string[] { "move" }));
            Assert.AreEqual(hall, player.Location, "Player's location changed after an invalid move");

            move.Execute(player, new string[] { "go", "east" });
            move.Execute(player, new string[] { "head", "south west" });

            Assert.AreEqual("This location has no path in the south direction", move.Execute(player, new string[] { "move", "south" }));
            Assert.AreEqual(lab, player.Location, "Player's location changed after an invalid move");

            Assert.AreEqual("Where do you want to go?", move.Execute(player, new string[] { "move" }));
            Assert.AreEqual(lab, player.Location, "Player's location changed after an invalid move");
        }
    }
}
```

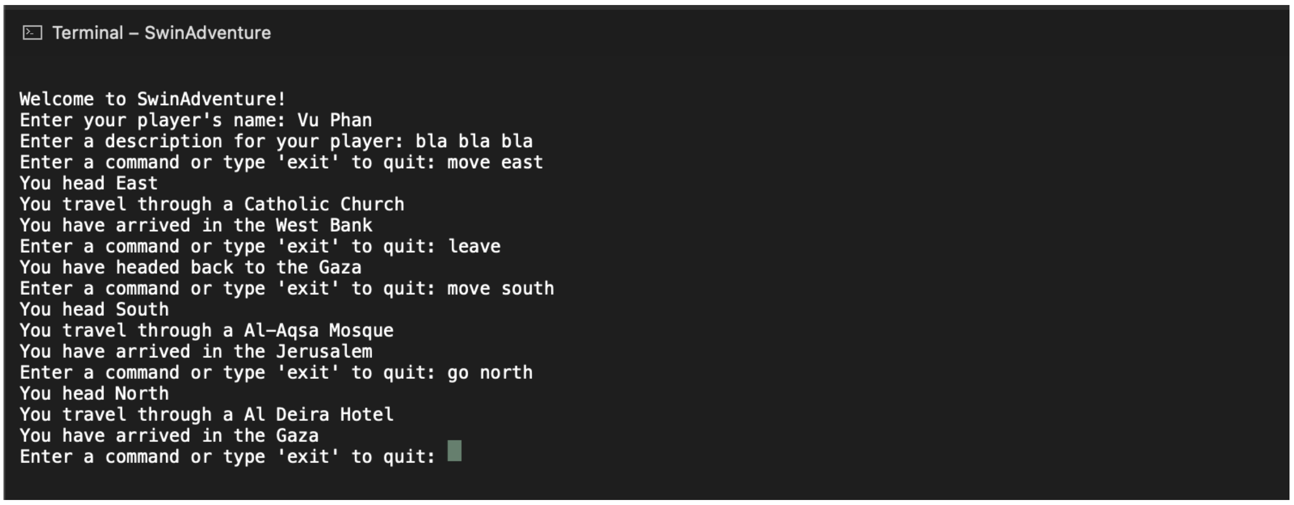

Inventory
uses
Player
- _location, _lastLocation : Location
+ Leave(): void
+ Location: Location<<property>>
<<abstract>> Command
Implements
MoveCommand
+ MoveCommand()
+ Execute(Player p, string[] text) : string
Implements
Implements
Path
- _destination : Location
- _description : string
+ Path(string[] ids, string name, string desc, Location destination)
+ SetDestination()
+ FullDescription: string<<override readonly, property>>
Implements
GameObject
<<interface>> IHaveInventory
Implements
Implements
Location
- _paths : List<Path>
+ AddPath() : void
+ GetPath(string id) : Path


SwinAdventure - Player Movement Along Paths
User
SwinAdventure
MoveCommand
Player
Location
Path
Enters a movement command
Executes the command
alt
[Command is 'leave']
Leaves current location
Returns to previous location
[Command is 'move', 'go', or 'head']
Retrieves current location
Gets the requested path
Gets the destination
Sets the destination
Updates player's location
Displays the outcome


MAP OF THE GAME
Default
Gaza
East
West
Westbank
North
South
North East
South West
Jerusalem


Tests

✓ SwinAdventure
  ✓ IdentifiableObject
    ✓ SwinAdventureTest
      ✓ BagTests
      ✓ IdentifiableObjectTest
      ✓ InventoryTests
      ✓ ItemTests
      ✓ LocationTests
      ✓ LookCommandTests
      ✓ PathTests
        ✓ TestGetPath
        ✓ TestInvalidMove
        ✓ TestLeaveLocation
        ✓ TestMovePlayer
      ✓ PlayerTests


Terminal – SwinAdventure

```
Welcome to SwinAdventure!
Enter your player's name: Vu Phan
Enter a description for your player: bla bla bla
Enter a command or type 'exit' to quit: move east
You head East
You travel through a Catholic Church
You have arrived in the West Bank
Enter a command or type 'exit' to quit: leave
You have headed back to the Gaza
Enter a command or type 'exit' to quit: move south
You head South
You travel through a Al-Aqsa Mosque
You have arrived in the Jerusalem
Enter a command or type 'exit' to quit: go north
You head North
You travel through a Al Deira Hotel
You have arrived in the Gaza
Enter a command or type 'exit' to quit:
```