COS20007: Object Oriented Programming

Credit Task 9.2C: Case Study — Iteration 7: Paths

Show Wai Yan/105293041

Location.cs

```
namespace SwinAdventure
{
    public class Location : GameObject, IHaveInventory
    {
        // Fields
        private readonly Inventory _inventory;
        private List<Path> _exits; // to indicate 10 directions

        // Constructor
        public Location(string[] ids, string name, string description)
            : base(ids.Concat(new string[] { "location", "place" }).ToArray(), name, description)
        {
            _inventory = new Inventory();
            _exits = new List<Path>();
        }

        // Properties
        public Inventory Inventory
        {
            get { return _inventory; } // Readonly properties
        }

        public override string ShortDescription
        {
            // need to make base's properties as virtual to make specific for location
            get { return FirstId; }
        }

        public override string FullDescription
        {
            get
            {
                return $"{base.FullDescription}\n{GetExits()}\nIn this room you can
see\n{Inventory.ItemList}";
            }
        }

        public List<Path> Exits
        {
            get { return _exits; }
            set { _exits = value; }
        }

        // Methods
        public GameObject Locate(string id)
        {
            if (AreYou(id))
                return this;
            return _inventory.Fetch(id);
        }

        public string GetExits()
        {
            if (_exits.Count == 0)
                return "There is no exist from here.";

            string exitsString = "There are exits to ";

            if (_exits.Count == 1)
                return exitsString + $"{_exits[0].FirstId}.";

            foreach (Path p in _exits)
            {
                if (p == _exits.Last())
                    exitsString += $"and {p.FirstId}.";
                else
                    exitsString += $"{p.FirstId}, ";
            }
            return exitsString;
        }
```

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

        public Path FindExits(string direction)
        {
            foreach (Path p in _exits)
            {
                if (p.AreYou(direction))
                    return p;
            }
            return null;
        }
    }
}
```

Player.cs

```
namespace SwinAdventure
{
    public class Player : GameObject, IHaveInventory
    {
        // Field
        private Inventory _inventory = new Inventory();
        private Location _currentLocation;

        // Constructor
        public Player(string name, string desc, Location spawnLocatoin)
            : base(new string[] { "me", "inventory" }, name, desc)
        {
            _currentLocation = spawnLocatoin;
        }

        // Properties
        public override string FullDescription
        {
            get
            {
                return $"You are {Name} {base.FullDescription}\nYou are carrying\n {Inventory.ItemList}";
            }
        }

        public Inventory Inventory
        {
            get { return _inventory; }
        }

        public Location CurrentLocation
        {
            get { return _currentLocation; }
        }

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

            GameObject obj = Inventory.Fetch(id);
            if (obj != null)
                return obj;

            obj = CurrentLocation.Locate(id);
            if (obj != null)
                return obj;

            return null;
        }

        public string WhereAmI()
        {
            return $"You are in {CurrentLocation.ShortDescription}";
        }

        public string Arrive()
        {
            return $"You have arrived in {CurrentLocation.ShortDescription}";
        }

        public string Exit(string direction)
```

```
        {
            // Only first letter Capitalized
            return $"You head {char.ToUpper(direction[0]) + direction.Substring(1)}";
        }

        public string Travel(Path p)
        {
            return p.FullDescription;
        }

        public string Move(Path p)
        {
          _currentLocation = p.EndLocation;
          return $"{Exit(p.FirstId)}\n{Travel(p)}\n{Arrive()}";
        }
    }
}
```

Path.cs

```
namespace SwinAdventure
{
    public class Path : GameObject
    {
        // Fields
        private Location _endLocation;
        private bool _lookable; // To indicate path is blocked or not

        // Constructor
        public Path(string[] ids, string name, string description, Location endLocation)
            : base(ids.Concat(new string[] { "path" }).ToArray(), name, description)
        {
            _endLocation = endLocation;
            _lookable = true;
        }

        // Properties
        public Location EndLocation
        {
            get { return _endLocation; }
            set { _endLocation = value; }
        }

        public bool Lookable
        {
            get { return _lookable; }
            set { _lookable = value; }
        }

        // Methods
    }
}
```

MoveCommand.cs

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

        // Methods
        public override string Execute(Player p, string[] text)
        {
            string[] validMoveCommand = { "move", "go", "head", "leave" };
            string[] validDirection =
            {
                "east",
                "e",
                "south east",
                "southeast",
                "se",
                "south",
                "s",
                "south west",
                "southwest",
                "sw",
                "west",
```

```
                "w",
                "north west",
                "northwest",
                "nw",
                "north",
                "n",
                "north east",
                "northeast",
                "ne",
                "up",
                "u",
                "down",
                "d"
            };

            // Check text has no more 3 word
            if (text.Length > 3 || text.Length < 2)
                return "I don\'t know how to move like that";

            // Check first letter of text is valid
            if (!validMoveCommand.Contains(text[0].ToLower()))
                return "Error in move input";

            // Check for direction validation
            string direction = String.Join(" ", text[1..]).ToLower();
            if (!validDirection.Contains(direction))
                return "Where are you heading to?";

            // Check path exist
            Path travelPath = p.CurrentLocation.FindExits(direction);
            if (travelPath == null)
                return $"Traveller, there is no exist in {direction}, try another way!";

            // Check path is travelable
            if (!travelPath.Lookable)
                return $"Traveller, {travelPath.Name} is currently blocked, please try in another time!";

            // Player Travel
            return p.Move(travelPath);
        }
    }
}
```

TestPath.cs

```
using NUnit.Framework;
using NUnit.Framework.Legacy;
using SwinAdventure;

namespace UnitTests
{
    [TestFixture]
    public class TestPath
    {
        private SwinAdventure.Path testPath;
        private Location l1;

        [SetUp]
        public void Setup()
        {
            l1 = new Location(
                new string[] { "a small tant", "tant" },
                "Small Tant",
                "This a rest place for traveller."
            );
            testPath = new SwinAdventure.Path(
                new string[] { "east", "e" },
                "road",
                "You are walking down the road from the east.",
                l1
            );
        }

        [Test]
        public void TestPathIsIdentifiable()
        {
          ClassicAssert.True(testPath.AreYou("east"));
          ClassicAssert.True(testPath.AreYou("e"));
          ClassicAssert.True(testPath.AreYou("path"));
          ClassicAssert.False(testPath.AreYou("north west"));
        }
```

```
        [Test]
        public void TestPathEndLocation()
        {
          ClassicAssert.That(testPath.EndLocation, Is.EqualTo(l1));
        }
    }
}
```

TestLocation.cs

```
using NUnit.Framework;
using NUnit.Framework.Legacy;
using SwinAdventure;

namespace UnitTests
{
    [TestFixture]
    public class TestLocation
    {
        private Location testLocation;
        private Bag bag;
        private Player player;
        private Item gem = new Item(new string[] { "gem" }, "a gem", "This is a gem");
        private Item shovel = new Item(new string[] { "shovel" }, "a shovel", "This is a shovel");
        private Item diamond = new Item(
            new string[] { "diamond" },
            "a diamond",
            "This is a diamond"
        );

        [SetUp]
        public void Setup()
        {
            testLocation = new Location(
                new string[] { "a small tant", "tant" },
                "Small Tant",
                "This a rest place for traveller."
            );
            bag = new Bag(
                new string[] { "bag", "backpack", "leather bag" },
                "Leather Bag",
                "A sturdy leather bag to carry items"
            );
            player = new Player("Show", "The Programmer", testLocation);
            bag.Inventory.Put(gem);
            bag.Inventory.Put(diamond);
            testLocation.Inventory.Put(shovel);
            testLocation.Inventory.Put(bag);
        }

        [Test]
        public void TestLocationIsIdentifiable()
        {
            ClassicAssert.True(testLocation.AreYou("location"));
            ClassicAssert.True(testLocation.AreYou("place"));
        }

        [Test]
        public void TestLocationCanLocateItem()
        {
            string bagId = bag.FirstId;
            ClassicAssert.That(bag, Is.EqualTo(testLocation.Locate(bagId)));

            string shovelId = shovel.FirstId;
            ClassicAssert.That(shovel, Is.EqualTo(testLocation.Locate(shovelId)));
        }

        [Test]
        public void TestPlayerCanLocateItemInLocation()
        {
            string bagId = bag.FirstId;
            ClassicAssert.That(bag, Is.EqualTo(player.Locate(bagId)));

            string shovelId = shovel.FirstId;
            ClassicAssert.That(shovel, Is.EqualTo(player.Locate(shovelId)));
        }

        [Test]
        public void TestGetAllExits()
        {
            SwinAdventure.Path p1 = new SwinAdventure.Path(
                new string[] { "north", "n" },
```

```
                "forest",
                "You are entering a dense forest from the north.",
                null
            );

            SwinAdventure.Path p2 = new SwinAdventure.Path(
                new string[] { "south", "s" },
                "valley",
                "You descend into a quiet valley from the south.",
                null
            );

            SwinAdventure.Path p3 = new SwinAdventure.Path(
                new string[] { "west", "w" },
                "bridge",
                "You cross a narrow bridge from the west.",
                null
            );
            testLocation.AddPath(p1);
            testLocation.AddPath(p2);
            testLocation.AddPath(p3);

            string exceptedString = "There are exits to north, south, and west.";
            ClassicAssert.That(testLocation.GetExits(), Is.EqualTo(exceptedString));
        }

        [Test]
        public void TestFindExits()
        {
            SwinAdventure.Path p1 = new SwinAdventure.Path(
                new string[] { "north", "n" },
                "forest",
                "You are entering a dense forest from the north.",
                null
            );
            testLocation.AddPath(p1);
            ClassicAssert.That(testLocation.FindExits("north"),Is.EqualTo(p1));
            ClassicAssert.That(testLocation.FindExits("east"),Is.EqualTo(null));
        }
    }
}
```

TestMoveCommand.cs

```
using NUnit.Framework;
using NUnit.Framework.Legacy;
using SwinAdventure;

namespace UnitTests
{
    [TestFixture]
    public class TestMoveCommand
    {
        private MoveCommand move;
        private Player player;
        private Location l1;
        private Location l2;
        private SwinAdventure.Path p1;
        private SwinAdventure.Path p2;

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

            l1 = new Location(
                new string[] { "a small tent", "tent" },
                "Small Tent",
                "This is a resting place for travelers."
            );

            l2 = new Location(
                new string[] { "a dark cave", "cave" },
                "Dark Cave",
                "A damp, echoing cave stretches into the darkness."
            );

            player = new Player("Show", "The Programmer", l1);

            p1 = new SwinAdventure.Path(
                new string[] { "north", "n" },
                "forest",
```

```
            "You are entering a dense forest from the north.",
            l2
        );

        p2 = new SwinAdventure.Path(
            new string[] { "south", "s" },
            "valley",
            "You descend into a quiet valley from the south.",
            l1
        );

        l1.AddPath(p1);

        l2.AddPath(p2);
    }

    [Test]
    public void TestLengthValidation()
    {
        string exceptedString = "I don\'t know how to move like that";
        ClassicAssert.That(
            move.Execute(player, new string[] { "move" }),
            Is.EqualTo(exceptedString)
        );
        ClassicAssert.That(
            move.Execute(player, new string[] { "move", "to", "east", "ok?" }),
            Is.EqualTo(exceptedString)
        );
    }

    [Test]
    public void TestCommandValidation()
    {
        string exceptedString = "Error in move input";
        ClassicAssert.That(
            move.Execute(player, new string[] { "mover", "west" }),
            Is.EqualTo(exceptedString)
        );
        ClassicAssert.That(
            move.Execute(player, new string[] { "moving", "wast" }),
            Is.EqualTo(exceptedString)
        );
        ClassicAssert.That(
            move.Execute(player, new string[] { "evom", "north", "wast" }),
            Is.EqualTo(exceptedString)
        );
    }

    [Test]
    public void TestDirectionValidation()
    {
        string exceptedString = "Where are you heading to?";
        ClassicAssert.That(
            move.Execute(player, new string[] { "move", "western" }),
            Is.EqualTo(exceptedString)
        );
        ClassicAssert.That(
            move.Execute(player, new string[] { "move", "eastern" }),
            Is.EqualTo(exceptedString)
        );
        ClassicAssert.That(
            move.Execute(player, new string[] { "move", "over", "there" }),
            Is.EqualTo(exceptedString)
        );
    }

    [Test]
    public void TestPathExist()
    {
        string exceptedString = "Traveller, there is no exist in west, try another way!";
        ClassicAssert.That(
            move.Execute(player, new string[] { "move", "west" }),
            Is.EqualTo(exceptedString)
        );
    }

    [Test]
    public void TestPathTravelable()
    {
        string exceptedString =
            "Traveller, forest is currently blocked, please try in another time!";
        p1.Lookable = false;
        ClassicAssert.That(
            move.Execute(player, new string[] { "move", "north" }),
```

```
                Is.EqualTo(exceptedString)
            );
        }

        [Test]
        public void TestPlayerTravel()
        {
            string exceptedString =
                $"{player.Exit(p1.FirstId)}\n{player.Travel(p1)}\nYou have arrived in {l2.ShortDescription}";
            ClassicAssert.That(
                move.Execute(player, new string[] { "move", "north" }),
                Is.EqualTo(exceptedString)
            );
        }

        [Test]
        public void TestPlayerMove()
        {
          Location exceptedLocation = l2;
          move.Execute(player,new string[] {"move", "north"});
          ClassicAssert.True(player.CurrentLocation == exceptedLocation);
        }

        [Test]
        public void TestPlayerNotMove()
        {
          Location exceptedLocation = player.CurrentLocation;
          move.Execute(player,new string[] {"moveing", "north"});
          ClassicAssert.True(player.CurrentLocation == exceptedLocation);
        }
    }
}
```

Program.cs

```
namespace SwinAdventure
{
    public class Program
    {
        public static void Main(string[] args)
        {
            // Configurations
            string helpCommand =
                $"Here is the List of command\n\t- look at me: Display what you are carrying in your
inventory\n\t- look at <item> [?in <container>]: Get description of that item,which inside in the
container\n\t- look: Display location's information\n\t- move <direction>: Player travel to that
location\n\t- quit/exit: Halt the program\n";

            // Getting Player's Name and Description
            string PlayerName = "";
            string PlayerDescription = "";
            Console.WriteLine("Write Your Name, Traveller!");
            Console.Write("NAME -> ");
            PlayerName = Console.ReadLine();
            Console.WriteLine("How about Your description, Traveller!");
            Console.Write("Description -> ");
            PlayerDescription = Console.ReadLine();

            // LOCATIONS
            Location shire = new Location(
                new[] { "shire" },
                "The Shire",
                "A peaceful land of Hobbits, green and quiet."
            );
            Location bree = new Location(
                new[] { "bree" },
                "Bree",
                "A small town with The Prancing Pony inn."
            );
            Location rivendell = new Location(
                new[] { "rivendell" },
                "Rivendell",
                "An Elven sanctuary full of ancient magic."
            );
            Location moria = new Location(
                new[] { "moria" },
                "Moria",
                "A dark underground Dwarven city, full of echo and danger."
            );
            Location mountDoom = new Location(
                new[] { "mount doom", "doom" },
                "Mount Doom",
```

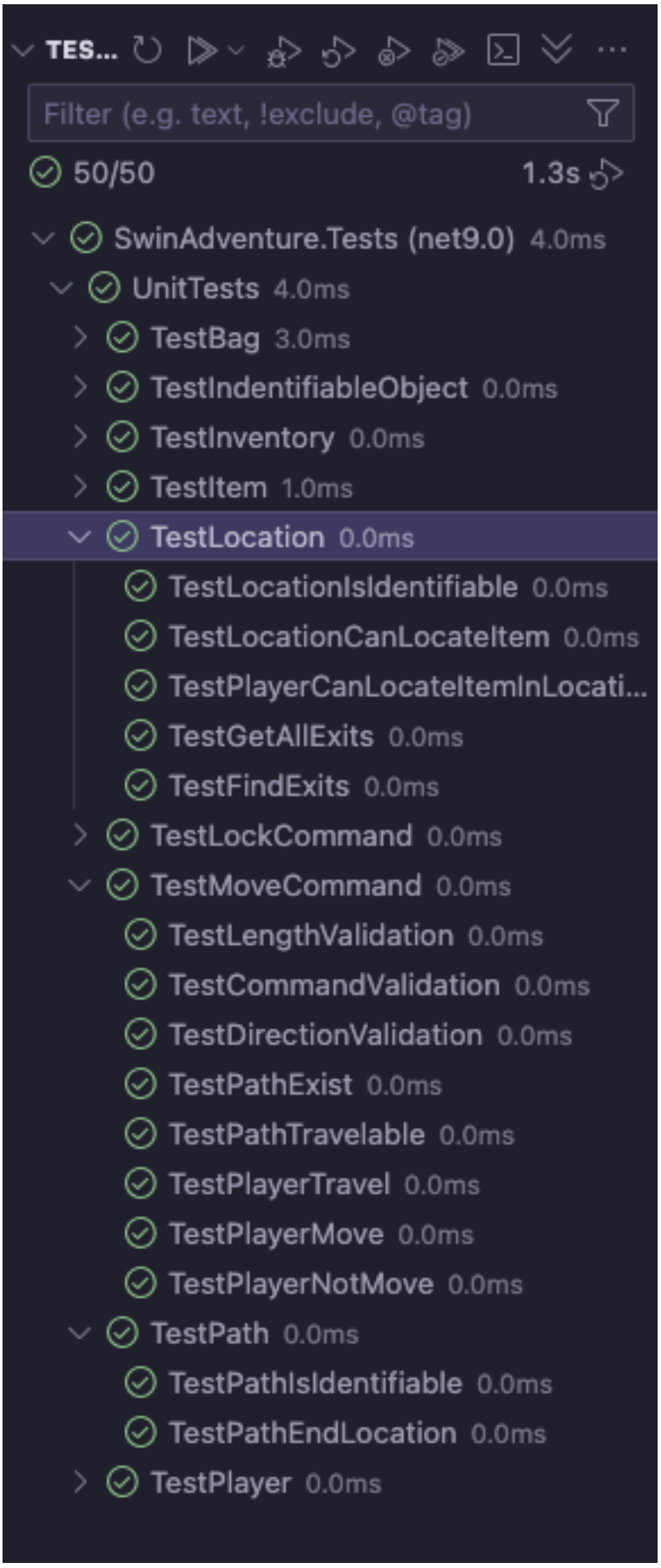
```
    "A fiery mountain in the heart of Mordor."
);
Location escapeTunnel = new Location(
    new[] { "tunnel", "escape tunnel" },
    "Secret Escape Tunnel",
    "A hidden tunnel beneath Mount Doom, dimly lit by glowing stones."
);

// Location items
// SHIRE
shire.Inventory.Put(
    new Item(
        new[] { "pipeweed", "pouch" },
        "Pipeweed Pouch",
        "A small pouch of fine pipeweed."
    )
);
shire.Inventory.Put(
    new Item(
        new[] { "hat", "farmer's hat" },
        "Farmer's Hat",
        "A straw hat once worn by a hobbit farmer."
    )
);
shire.Inventory.Put(
    new Item(
        new[] { "loaf", "bread" },
        "Hobbit Loaf",
        "Freshly baked bread from the Shire."
    )
);

// BREE
bree.Inventory.Put(
    new Item(
        new[] { "mug", "ale" },
        "Mug of Ale",
        "A frothy mug from The Prancing Pony."
    )
);
bree.Inventory.Put(
    new Item(
        new[] { "dagger", "rusty dagger" },
        "Rusty Dagger",
        "Old and blunt, but still dangerous."
    )
);
bree.Inventory.Put(
    new Item(
        new[] { "cloak", "travel cloak" },
        "Travel Cloak",
        "A heavy cloak for cold nights."
    )
);

// RIVENDELL
rivendell.Inventory.Put(
    new Item(
        new[] { "bread", "elven bread", "lembas" },
        "Elven Bread",
        "One bite is enough for a full day's journey."
    )
);
rivendell.Inventory.Put(
    new Item(
        new[] { "pendant", "silver pendant" },
        "Silver Pendant",
        "An Elven trinket that shimmers faintly."
    )
);
rivendell.Inventory.Put(
    new Item(
        new[] { "book", "ancient book" },
        "Ancient Book",
        "Filled with forgotten lore and legends."
    )
);

// MORIA
moria.Inventory.Put(
    new Item(
        new[] { "pickaxe", "broken pickaxe" },
        "Broken Pickaxe",
        "Snapped at the handle."
```

```
    )
);
moria.Inventory.Put(new Item(new[] { "torch" }, "Torch", "Still usable if relit."));
moria.Inventory.Put(
    new Item(
        new[] { "gauntlets", "dwarven gauntlets" },
        "Dwarven Gauntlets",
        "Heavy gloves forged in the mountains."
    )
);

// MOUNT DOOM
mountDoom.Inventory.Put(
    new Item(
        new[] { "ring shard", "shard" },
        "Black Ring Shard",
        "A broken piece of something ancient and cursed."
    )
);
mountDoom.Inventory.Put(
    new Item(new[] { "lava", "rock" }, "Lava Rock", "Still warm to the touch.")
);
mountDoom.Inventory.Put(
    new Item(
        new[] { "journal", "burned journal" },
        "Burned Journal",
        "Most pages are unreadable, but a few notes remain."
    )
);

// ESCAPE TUNNEL
escapeTunnel.Inventory.Put(
    new Item(
        new[] { "silk", "spider silk" },
        "Spider Silk",
        "Sticky and unnaturally strong."
    )
);
escapeTunnel.Inventory.Put(
    new Item(
        new[] { "crystal", "shard" },
        "Crystal Shard",
        "Glows faintly with magical energy."
    )
);
escapeTunnel.Inventory.Put(
    new Item(
        new[] { "torch", "elven torch" },
        "Elven Torch",
        "Lights automatically in the darkness."
    )
);

// PATHS (Bidirectional and One-Way)

// Shire ↔ Bree
Path shireToBree = new Path(
    new[] { "east", "e" },
    "east",
    "A path to Bree, lined with fields.",
    bree
);
Path breeToShire = new Path(
    new[] { "west", "w" },
    "west",
    "A path back to the Shire.",
    shire
);

// Bree ↔ Rivendell
Path breeToRivendell = new Path(
    new[] { "north", "n" },
    "north",
    "The path to Rivendell through forested slopes.",
    rivendell
);
Path rivendellToBree = new Path(
    new[] { "south", "s" },
    "south",
    "A path back down to Bree.",
    bree
);

// Shire ↔ Rivendell (shortcut)
```

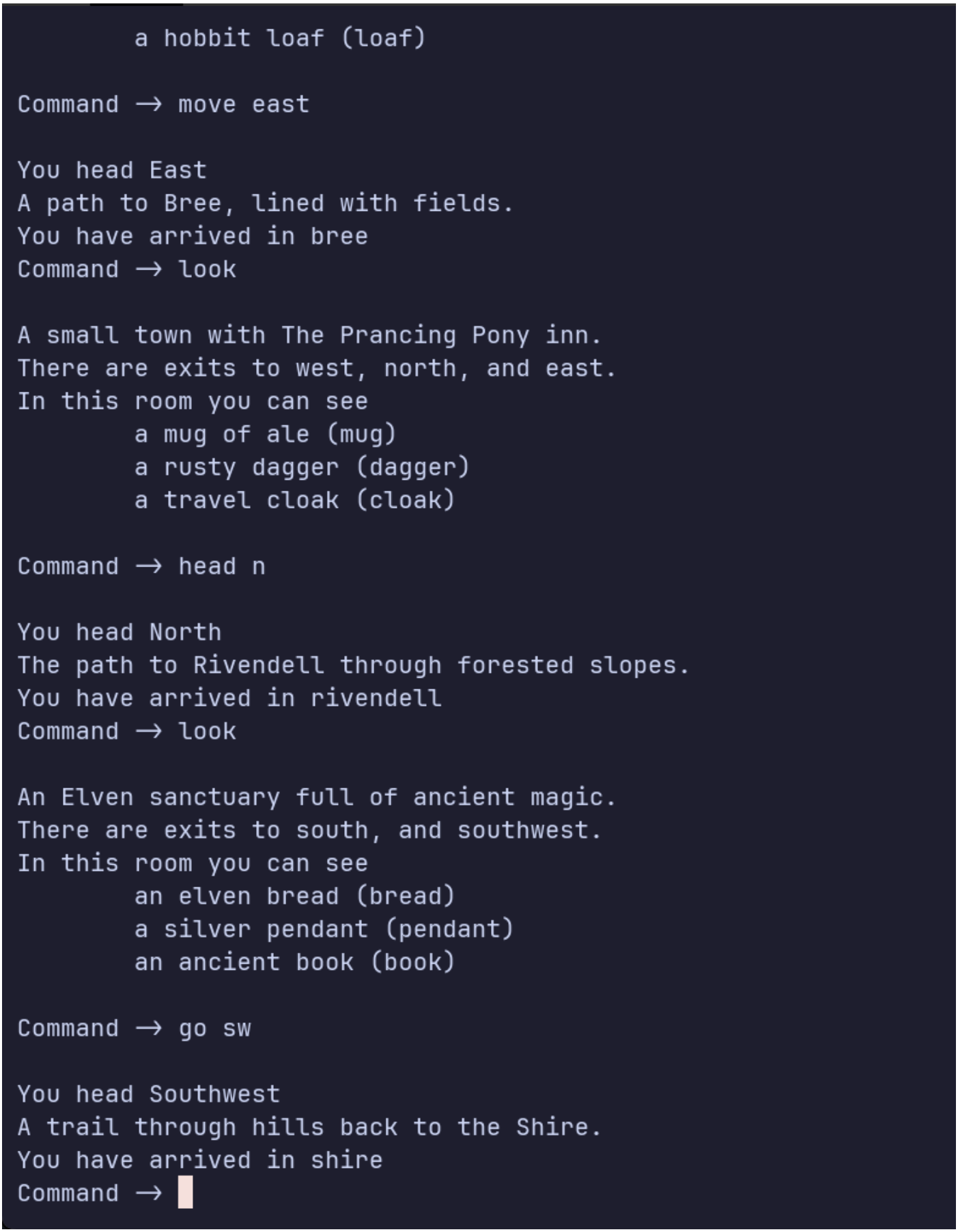
```
Path shireToRivendell = new Path(
    new[] { "northeast", "ne" },
    "northeast",
    "An old Elven path to Rivendell.",
    rivendell
);
Path rivendellToShire = new Path(
    new[] { "southwest", "sw" },
    "southwest",
    "A trail through hills back to the Shire.",
    shire
);

// Bree ↔ Moria
Path breeToMoria = new Path(
    new[] { "east", "e" },
    "east",
    "The eastern road to the mines of Moria.",
    moria
);
Path moriaToBree = new Path(
    new[] { "west", "w" },
    "west",
    "A narrow road back to Bree.",
    bree
);

// Moria → Mount Doom (one-way)
Path moriaToDoom = new Path(
    new[] { "south", "s" },
    "south",
    "A dark, narrow path leads to Mount Doom.",
    mountDoom
);

// Mount Doom → Escape Tunnel (one-way)
Path doomToTunnel = new Path(
    new[] { "down", "d" },
    "Escape Tunnel",
    "A rocky slope leads to a hidden escape tunnel.",
    escapeTunnel
);

// Escape Tunnel → Moria (return path)
Path tunnelToMoria = new Path(
    new[] { "up", "u" },
    "Moria",
    "You follow the tunnel upward back into Moria's depths.",
    moria
);

// ADD PATHS TO LOCATIONS

shire.AddPath(shireToBree);
shire.AddPath(shireToRivendell);

bree.AddPath(breeToShire);
bree.AddPath(breeToRivendell);
bree.AddPath(breeToMoria);

rivendell.AddPath(rivendellToBree);
rivendell.AddPath(rivendellToShire);

moria.AddPath(moriaToBree);
moria.AddPath(moriaToDoom); // No return from Doom to Moria

mountDoom.AddPath(doomToTunnel); // No path back to Moria

escapeTunnel.AddPath(tunnelToMoria); // Secret return

// Player
Player me = new Player(PlayerName, PlayerDescription, shire);

// Player Items
Item sword = new Item(
    new[] { "sword", "steel sword" },
    "Steel Sword",
    "A well-balanced sword of polished steel."
);
Item shield = new Item(
    new[] { "shield", "leather shield" },
    "Leather Shield",
    "A round shield made of hardened leather."
);
```

```
            Bag starterBag = new Bag(
                new[] { "bag", "satchel" },
                "Adventurer's Bag",
                "A worn leather bag with room for essentials."
            );

            // Items inside the bag
            Item healingPotion = new Item(
                new[] { "potion", "healing potion" },
                "Healing Potion",
                "Restores health when consumed."
            );
            Item mapFragment = new Item(
                new[] { "map", "fragment" },
                "Map Fragment",
                "A torn piece of an ancient map leading somewhere..."
            );

            // Add items to bag
            starterBag.Inventory.Put(healingPotion);
            starterBag.Inventory.Put(mapFragment);

            // Add everything to player
            me.Inventory.Put(sword);
            me.Inventory.Put(shield);
            me.Inventory.Put(starterBag);

            // Command Configuration
            LookCommand lookCommand = new LookCommand();
            MoveCommand moveCommand = new MoveCommand();

            // Game Loop
            Console.WriteLine("Write '-h' for helper");

            Console.WriteLine(me.Arrive());
            while (true)
            {
                string command = "";
                Console.Write("Command -> ");
                command = Console.ReadLine().ToLower();
                Console.WriteLine(); // to make clear after input line for presented looking

                if (command == "exit" || command == "quit")
                {
                    Console.WriteLine("Take the rest, Traveller!");
                    return;
                }
                else if (command == "-h")
                {
                    Console.WriteLine(helpCommand);
                }
                else if (lookCommand.AreYou(command.Split(' ')[0]))
                {
                    Console.WriteLine(lookCommand.Execute(me, command.Split(' ')));
                }
                else if (moveCommand.AreYou(command.Split(' ')[0]))
                {
                    Console.WriteLine(moveCommand.Execute(me, command.Split(' ')));
                }
                else
                    Console.WriteLine("I don't know that command, Traveller!");
            }
        }
    }
}
```

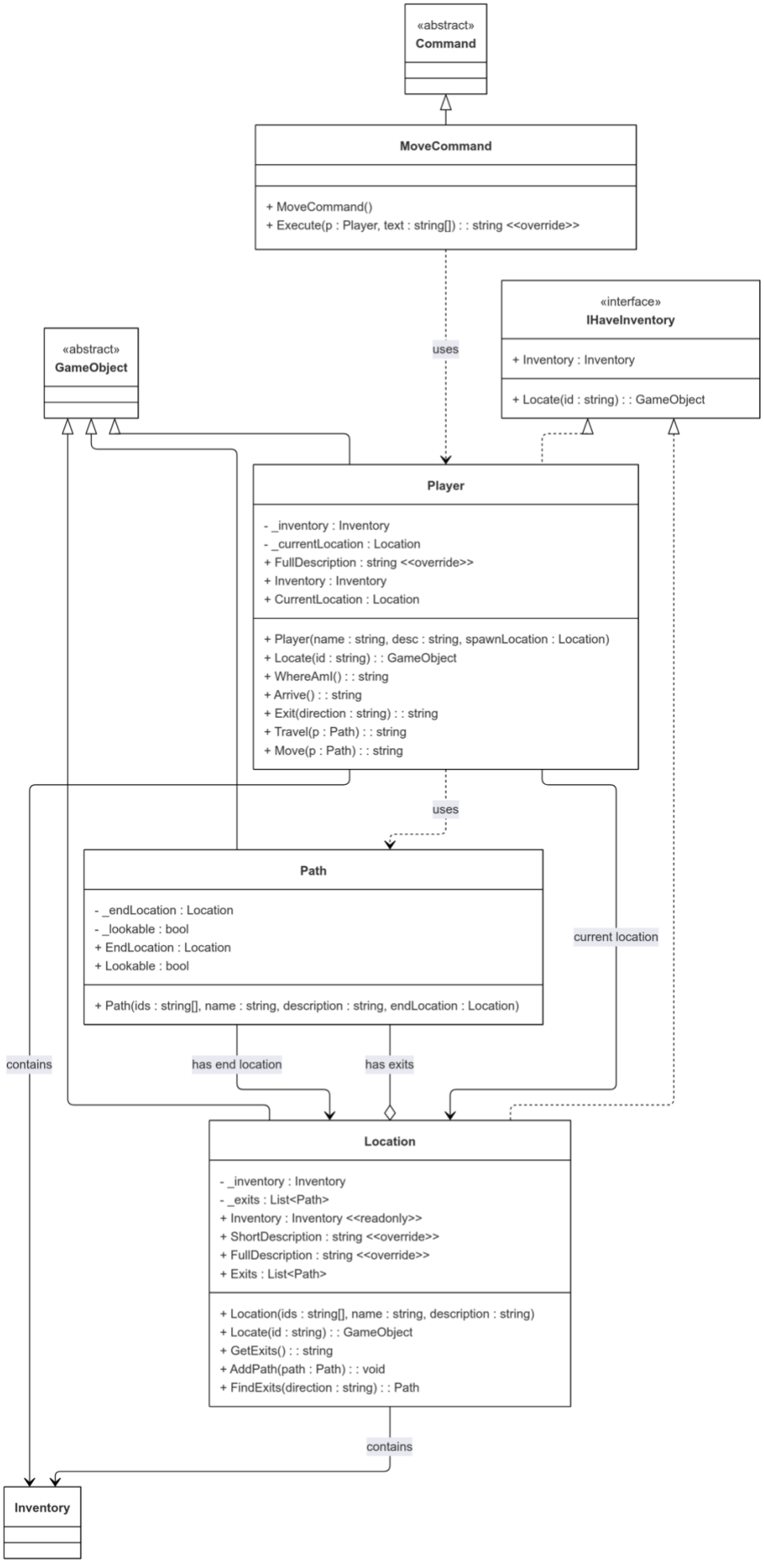
Screenshot of unit test passing



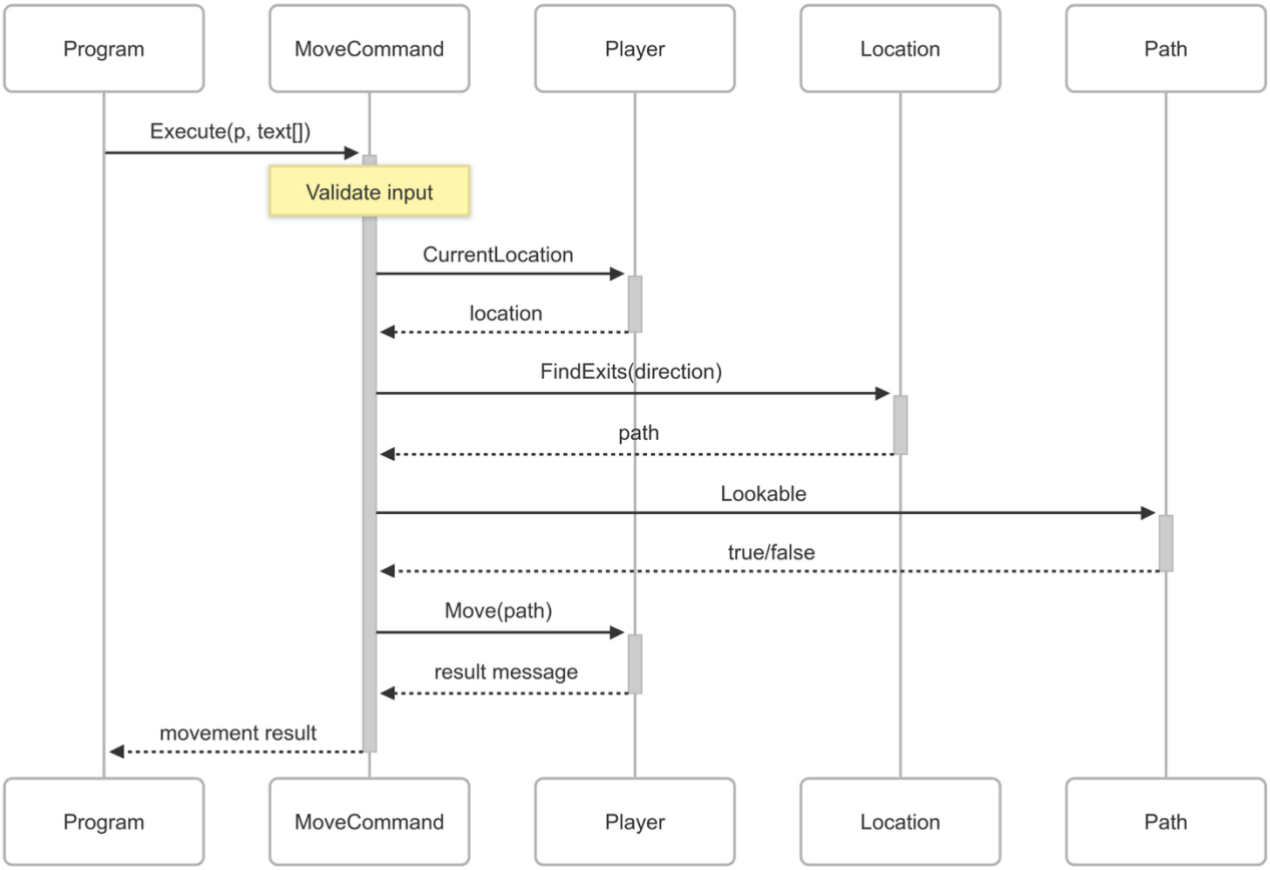
Screenshot of program running showing new commands related to locations



UML Class diagram showing what needs to be added



UML Sequence diagram to explain how Locate works in the Player



Map