COS20007 - Object Oriented Programming

9.2C - Case Study - Advanced Iteration 7: Paths

Student name: Nguyen Duc Manh
ID: 105547489

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

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

        public Location(string[] idents, string name, string desc) : base
          (idents, name, desc)
        {
            _inventory = new Inventory();
            _paths = null;
        }

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

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

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

        public override string FullDescription
        {
            get
            {
                return $"You are in: {Name}, {base.FullDescription}. Here you can see: {_inventory.ItemList()}";
            }
        }

        public Inventory Inventory
        {
            get { return _inventory; }
        }

        public void AddPath(Path path)
        {
            if (_paths == null)
            {
                _paths = new List<Path>();
            }
            _paths.Add(path);
        }

        public Path Path
        {
            get { return _paths?[0]; } // But only if list is guaranteed to have at least 1
        }
    }
}
```

```
namespace SwinAdventure
{
    public class MoveCommand : Command
    {
        Player p;

        public MoveCommand() : base(["move", "go", "head", "leave"])
        { }

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

            string id = text[1].ToLower();
            GameObject obj = p.Location.Locate(id);

            if (obj == null)
            {
                return "There is no path in that direction.";
            }

            Path path = obj as Path;
            if (path != null)
            {
                p.Location = path.Destination;
                return $"You move {id} to {path.Destination.Name}";
            }
            else
                return "That doesn't seem like a valid path.";
        }
    }
}

```

```
namespace SwinAdventure;

public class MoveCommandTest
{
    Location start;
    Location end;
    Path north;
    Player player;
    MoveCommand moveCommand;

    [SetUp]
    public void Setup()
    {
        start = new Location(["duytan"], "80 Duy Tan", "Innovation Space");
        end = new Location(["vovinam"], "3.1 VOV", "Martial Art");
        north = new Path(["north"], "north move", "go through the north
          forrest", end);
        player = new Player("TestPlayer", "A test player");
        player.Location = start;
        moveCommand = new MoveCommand();
    }

    [Test]
    public void TestValidMove()
    {
        start.AddPath(north);
        string result = moveCommand.Execute(player, ["move", "north"]);
        string expect = $"You move north to {end.Name}";
        Assert.That(result, Is.EqualTo(expect));
    }

    [Test]
    public void TestInvalidMove()
    {
        string result = moveCommand.Execute(player, ["move", "south"]);
        string expect = "There is no path in that direction.";
        Assert.That(result, Is.EqualTo(expect));
    }

    [Test]
    public void TestNonPath()
    {
        Item item = new Item(["north"], "a test item", "this is a test
          item");
        start.Inventory.Put(item);
        string result = moveCommand.Execute(player, ["move", "north"]);
        string expect = "That doesn't seem like a valid path.";
        Assert.That(result, Is.EqualTo(expect));
    }

    [Test]
    public void TestInvalidCommand()
    {
        string result = moveCommand.Execute(player, ["move"]);
        string expect = "I don't know how to move like that";
        Assert.That(result, Is.EqualTo(expect));
    }
}

```

```
namespace SwinAdventure
{
    public class Path : GameObject
    {
        Location _des;

        public Path(string[] idents, string name, string desc, Location
          des) : base(idents, name, desc)
        {
            _des = des;
        }

        public Location Destination
        {
            get
            { return _des; }
        }
    }
}
```

```
namespace SwinAdventure
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Enter your player's name: ");
            string name = Console.ReadLine();

            Console.WriteLine("Enter player's description: ");
            string des = Console.ReadLine();

            // Create a player and some items
            Player player = new(name, des);
            Item itm1 = new(["hdmi"], "HDMI cord", "can connect to large
              screen");
            Item itm2 = new(["usb"], "an USB", "can store up to 1TB of
              data");
            Bags bag = new(["bag"], "a bag", "this bag is made by
              leather");
            Item itm3 = new(["mouse"], "a mouse", "gaming mouse with 0
              latency");
            Command lookCommand = new LookCommand();
            Location duytan = new(["duytan"], "duytan", "Innovation Center
              of Swinburne");
            Location duongkhue = new(["duongkhue"], "duongkhue", "Global
              Citizen Education");
            Path north = new(["north"], "north move", "Duy Tan street",
              duytan);
            Path south = new(["south"], "south move", "Cau Giay street",
              duongkhue);
            Command moveCommand = new MoveCommand();


            player.Inventory.Put(itm1);
            player.Inventory.Put(itm2);
            player.Inventory.Put(bag);
            bag.Inventory.Put(itm3);
            duytan.Inventory.Put(itm1);
            player.Location = duytan;
            duytan.AddPath(south);
            duongkhue.AddPath(north);

            string[] MoveCommandWord = { "move", "head", "go" };

            //loop command
            while (true)
            {
                Console.WriteLine("What do you want to find or move?");
                string userInput = Console.ReadLine();
                string[] userCommand = userInput.Split(' ');
                if (userInput.ToLower() != "exit")
                {
                    if (userInput == "look")
                    {
                        //string[] userCommand = [userInput];
                        string result = lookCommand.Execute(player, userCommand);
                        Console.WriteLine(" ");
                        Console.WriteLine(result);
                    }
                    if (MoveCommandWord.Contains(userCommand[0]))
                    {
                        //string[] userCommand = userInput.Split(' ');
                        string result = moveCommand.Execute(player, userCommand);
                        Console.WriteLine(" ");
                        Console.WriteLine(result);
                    }
                    else
                    {
                        //string[] userCommand = userInput.Split(' ');
                        string result = lookCommand.Execute(player, userCommand);
                        Console.WriteLine(" ");
                        Console.WriteLine($"{result}\n");
                    }
                }
                else break;
            }
            Console.WriteLine("Iteration 5 finished !");
        }
    }
}
```

Earlier code

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

namespace SwinAdventure
{
    public class Bags : Item, IHaveInventory
    {
        Inventory _inventory;

        public Bags(string[] idents, string name, string desc) : base
          (idents, name, desc)
        {
            _inventory = new Inventory();
        }

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

        public override string FullDescription
        {
            get { return $"In the {Name} you can see:\n{_inventory.ItemList
              ()}" ; }
        }

        public Inventory Inventory
        {
            get { return _inventory; }
        }
    }
}
```

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

namespace SwinAdventure
{
    public abstract class Command : IdenObj
    {
        public Command(string[] ids) : base(ids)
        {
            //
        }

        public abstract string Execute(Player p, string[] text);
    }
}

```

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

namespace SwinAdventure
{
    public abstract class GameObject : IdenObj
    {
        string _description;
        string _name;

        public GameObject(string[] idents, string name, string desc) : base
          (idents)
        {
            _name = name;
            _description = desc;
        }

        public string Name { get { return _name; } }

        public string ShortDescription { get { return $"{_name}
          ({FirstId})"; } }

        public virtual string FullDescription { get { return
          _description; } }
    }
}
```

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

namespace SwinAdventure
{
    public class IdenObj
    {
        //fields
        private List<string> _identifiers;
        string _myStudentID = "7489";

        //constructor
        public IdenObj(string[] idents)
        {
            _identifiers = new List<string>();
            if (idents != null)
            {
                for (int i = 0; i < idents.Length; i++)
                {
                    _identifiers.Add(idents[i].ToLower());
                }
            }
        }

        //methods
        public bool AreYou(string id)
        {
            return _identifiers.Contains(id.ToLower());
        }

        public string FirstId
        {
            get
            {
                if( _identifiers.Count == 0)
                {
                    return "";
                } else { return _identifiers.First(); }
            }
        }

        public void AddIdentifier(string id)
        {
            _identifiers.Add(id.ToLower());
        }

        public void PrivilegeEscalation(string pin)
        {
            if(pin.Length == 4)
            {
                if(pin == _myStudentID) //105547489
                {
                    _identifiers[0] = _myStudentID;
                }
            }
            else
            {
                return;
            }

        }
    }
}

```

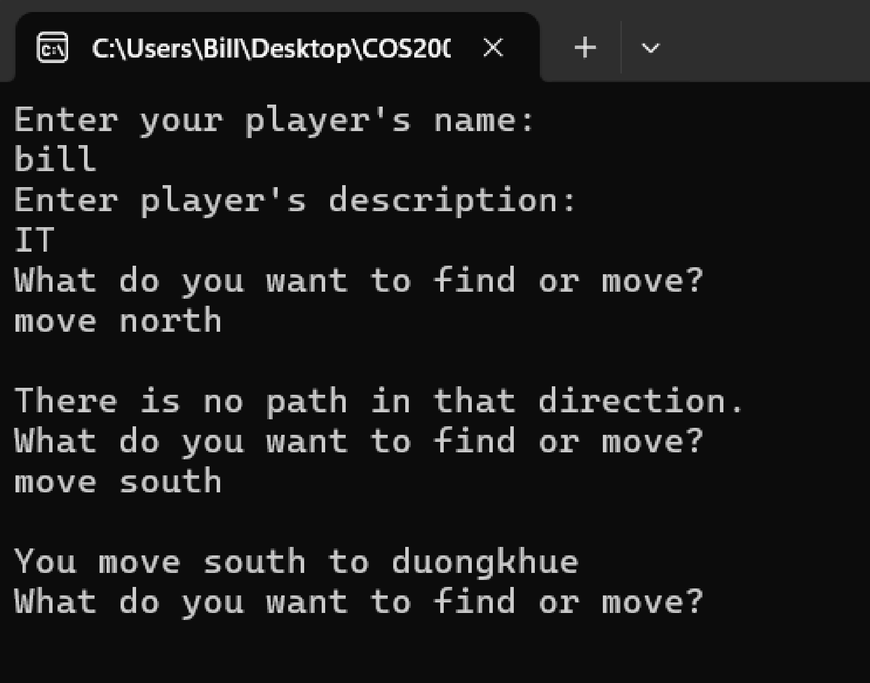
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public interface IHaveInventory
    {
        public GameObject Locate(string id);

        public string Name { get; }
    }
}

```

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

namespace SwinAdventure
{
    public class Item : GameObject
    {
        public Item(string[] idents, string name, string desc) : base
          (idents, name, desc)
        {
            //not yet
        }
    }
}

```

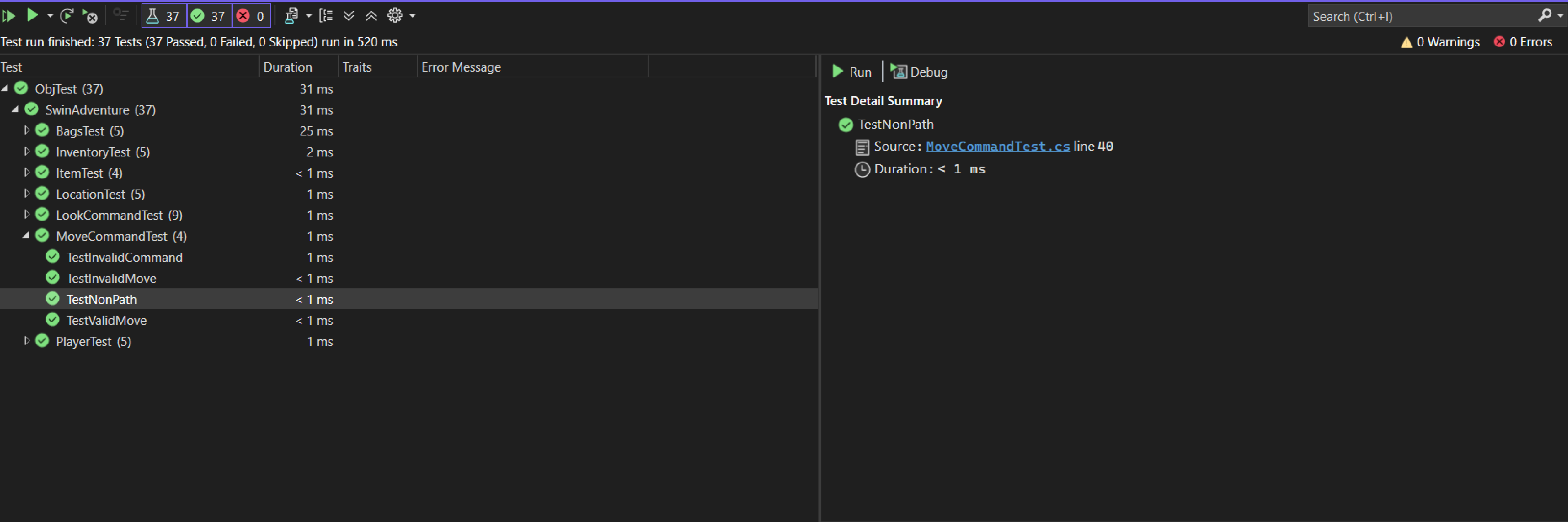
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure
{
    public class Inventory
    {
        List<Item> _items;

        public Inventory()
        {
            _items = new List<Item>();
        }

        public bool HasItem(string id)
        {
            foreach (Item item in _items)
            {
                if (item.AreYou(id))
                {
                    return true;
                }
            }
            return false;
        }

        public void Put(Item itm)
        {
            _items.Add(itm);
        }

        //public void RemoveItm(Item itm)
        //{
        //    if (_items.Contains(itm))
        //    {
        //        _items.Remove(itm);
        //    }
        //}

        public Item Take(string id)
        {
            foreach (Item item in _items)
            {
                if (item.AreYou(id))
                {
                    _items.Remove(item);
                    return item;
                }
            }
            return null;
        }

        public Item Fetch(string id)
        {
            foreach (Item item in _items)
            {
                if (item.AreYou(id))
                {
                    return item;
                }
            }
            return null;
        }

        public string ItemList()
        {
            string listitm = "";
            foreach (Item item in _items)
            {
                listitm = listitm + item.ShortDescription + "\n";
            }
            return listitm;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Runtime.InteropServices.JavaScript.JSType;
using System.Xml.Linq;

namespace SwinAdventure
{
    public class LookCommand : Command
    {
        IHaveInventory container;
        GameObject item;
        Player p;
        Location location;

        public LookCommand() : base(["look"]) { }

        public override string Execute(Player p, string[] text)
        {
            if (text.Length == 1 && text[0].ToLower() == "look")
            {
                return p.Location.FullDescription;
            }
            if (text.Length == 3 || text.Length == 5)
            {
                if (text[0] != "look")
                    return "Error in look input";
                if (text[1] != "at")
                    return "What do you want to look at?";
                if (text.Length == 5 && text[3] != "in")
                    return "What do you want to look in?";
                if (text.Length == 3)
                {
                    container = p;
                }
                else
                {
                    container = FetchContainer(p, text[4]);
                    if (container == null)
                        return $"I cannot find the {text[4]}";
                }

                return LookAtIn(text[2], container);
            }
            else
                return "I don't know how to look like that";
        }

        private IHaveInventory? FetchContainer(Player p, string containerId)
        {
            return p.Locate(containerId) as IHaveInventory;
        }

        private string LookAtIn(string thingId, IHaveInventory container)
        {
            if (container.Locate(thingId) != null)
            {
                return container.Locate(thingId).FullDescription;
            }
            else
                return $"I cannot find the {thingId}";
        }
    }
}

```

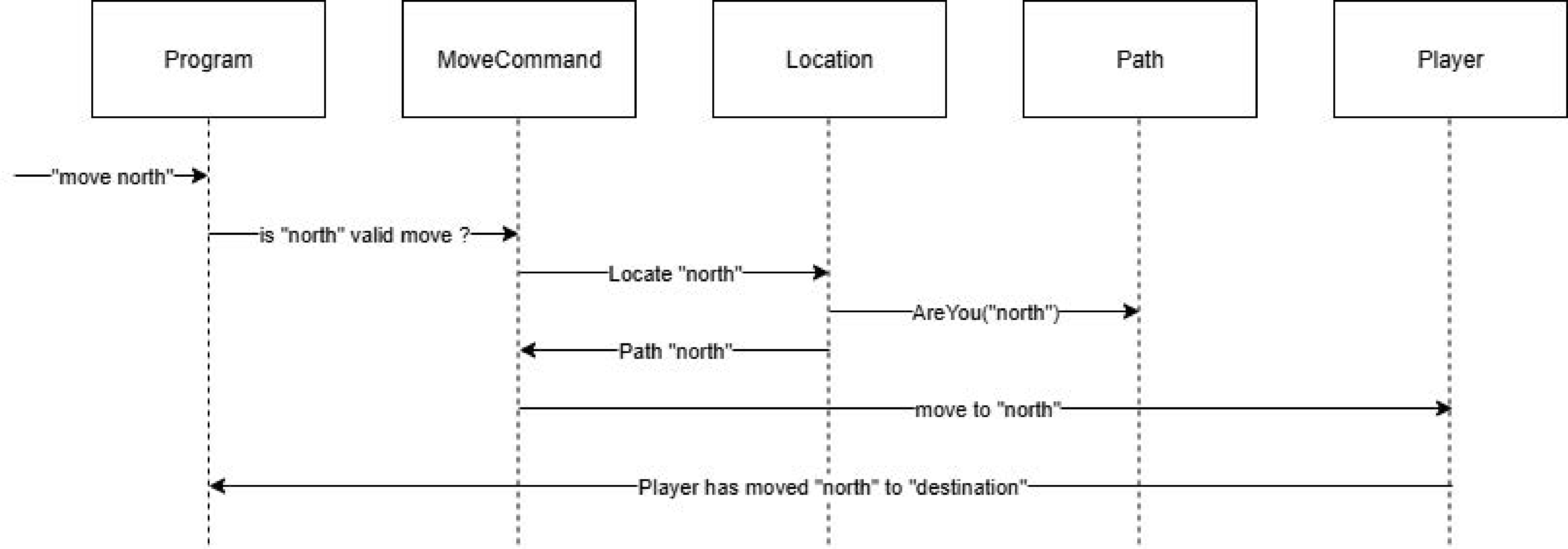
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

        public Player (string name, string desc) : base(new string[] {"
          me", "inventory"}, name, desc)
        {
            _inventory = new Inventory();
        }

        public GameObject Locate(string id)
        {
            if (AreYou(id))
            {
                return this;
            }
            var itm = _inventory.Fetch(id);
            if (itm != null)
            {
                return itm;
            }
            if (Location != null)
            {
                return Location.Locate(id);
            }
            return null;
        }

        public override string FullDescription
        {
            get
            {
                return $"{Name}, {base.ShortDescription}.You are carrying:
                  {_inventory.ItemList()}";
            }
        }

        public Inventory Inventory { get { return _inventory; } }

        public Location Location { get; set; }
    }
}

```

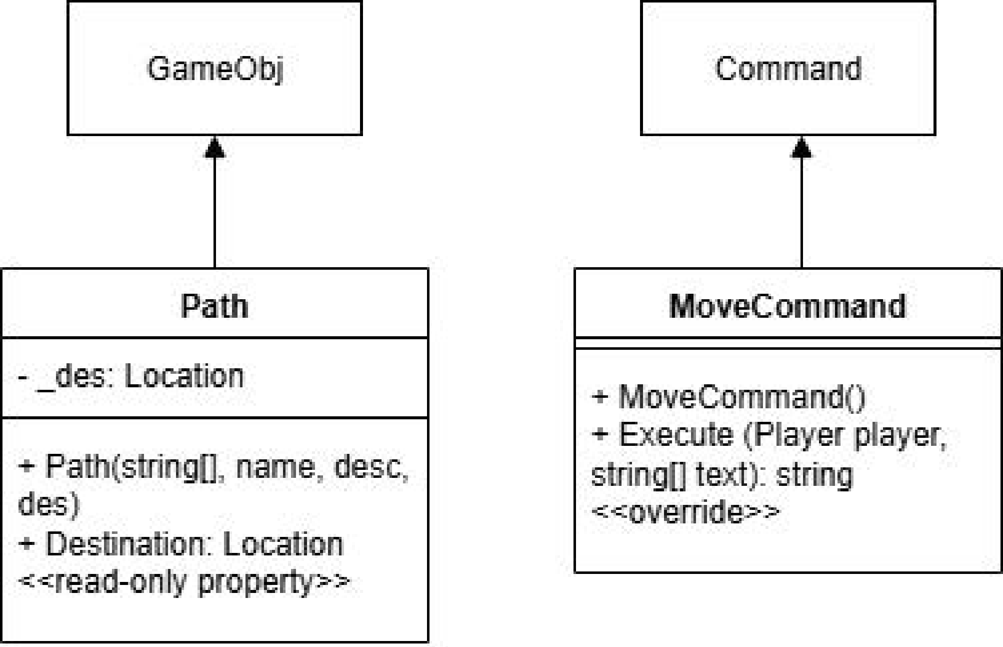


```
Enter your player's name:
bill
Enter player's description:
IT
What do you want to find or move?
move north

There is no path in that direction.
What do you want to find or move?
move south

You move south to duongkhue
What do you want to find or move?
```



```
Test run finished: 37 Tests (37 Passed, 0 Failed, 0 Skipped) run in 520 ms        0 Warnings   0 Errors

Test                              Duration   Traits   Error Message
ObjTest (37)                        31 ms
  SwinAdventure (37)                31 ms
    BagsTest (5)                    25 ms
    InventoryTest (5)                2 ms
    ItemTest (4)                   < 1 ms
    LocationTest (5)                 1 ms
    LookCommandTest (9)              1 ms
    MoveCommandTest (4)              1 ms
      TestInvalidCommand             1 ms
      TestInvalidMove              < 1 ms
      TestNonPath                  < 1 ms
      TestValidMove                < 1 ms
    PlayerTest (5)                   1 ms

Test Detail Summary
  TestNonPath
    Source: MoveCommandTest.cs line 40
    Duration: < 1 ms
```


Program
MoveCommand
Location
Path
Player
"move north"
is "north" valid move ?
Locate "north"
AreYou("north")
Path "north"
move to "north"
Player has moved "north" to "destination"


GameObj
Command
Path
- _des: Location
+ Path(string[], name, desc, des)
+ Destination: Location <<read-only property>>
MoveCommand
+ MoveCommand()
+ Execute (Player player, string[] text): string <<override>>