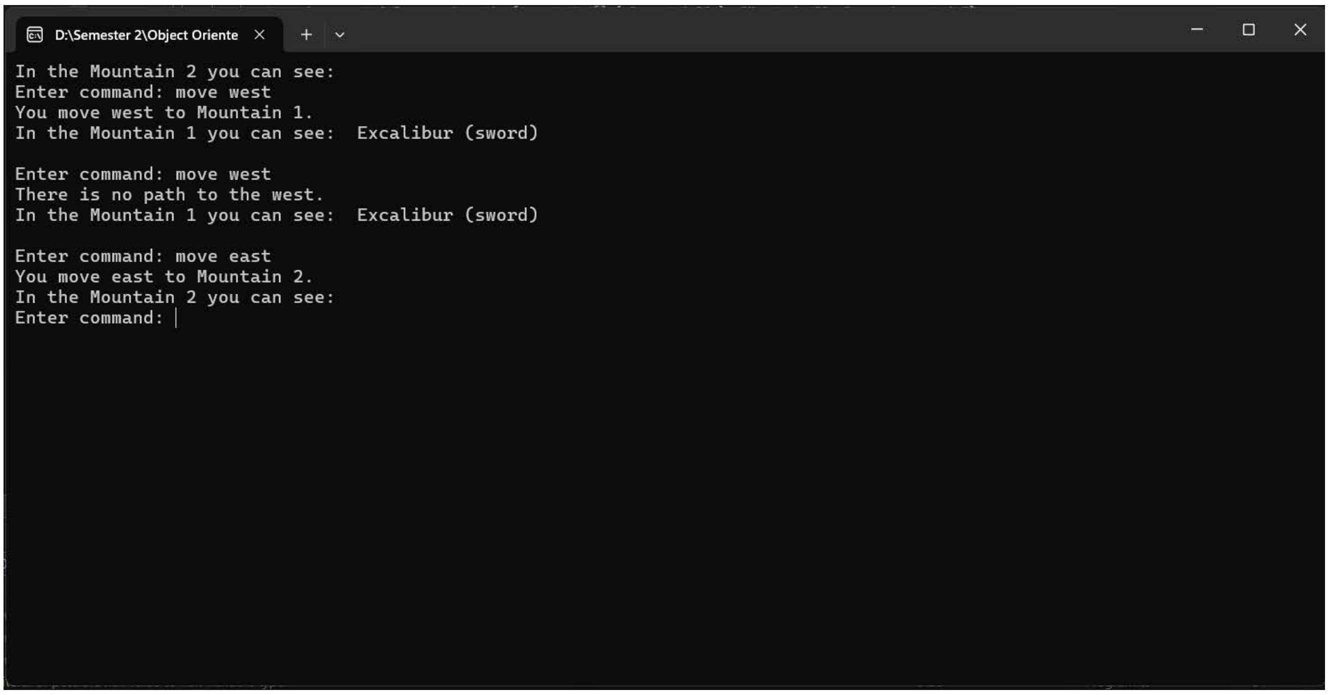
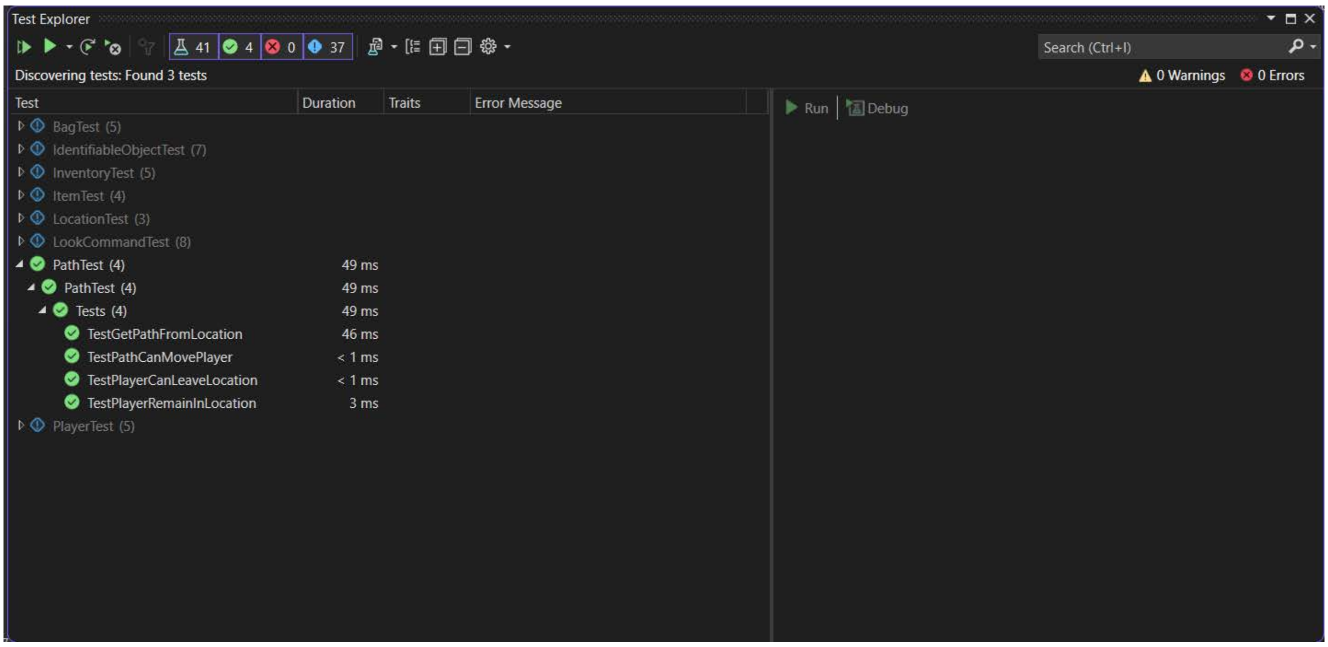
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace _9._2C
{
    public class Bag : Item, IHaveInventory
    {
        private Inventory _inventory;

        public Bag(string[] ids, string name, string description) : base(ids, name, description)
        {
            _inventory = new Inventory();
        }

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

        public override string FullDescription
        {
            get
            {
                return $"In the {Name} you can see: {string.Join(", ", _inventory.ItemList)}"; //add "," between every elements
            }
        }

        public Inventory Inventory
        { get { return _inventory; } }



    }
}

```

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

namespace _9._2C
{
    public abstract class Command : IdentifiableObject //base class for other classes, cannot create an object
    {
        private string[] _ids;
        public Command(string[] ids) : base(ids)
        {
            _ids = ids;
        }

        public abstract string Execute(Player p, string[] text); // define without implementation



    }
}

```



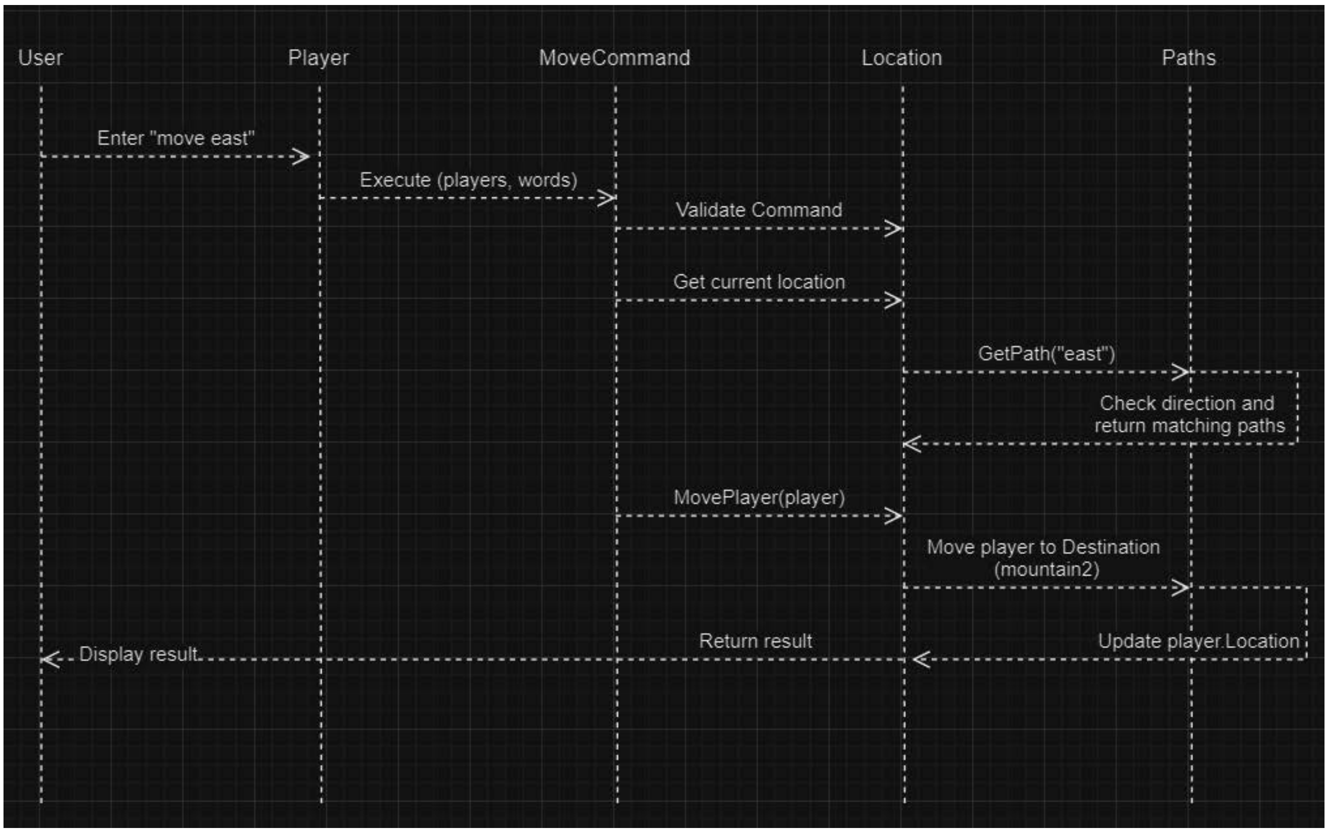
```
In the Mountain 2 you can see:
Enter command: move west
You move west to Mountain 1.
In the Mountain 1 you can see:  Excalibur (sword)

Enter command: move west
There is no path to the west.
In the Mountain 1 you can see:  Excalibur (sword)

Enter command: move east
You move east to Mountain 2.
In the Mountain 2 you can see:
Enter command: 
```

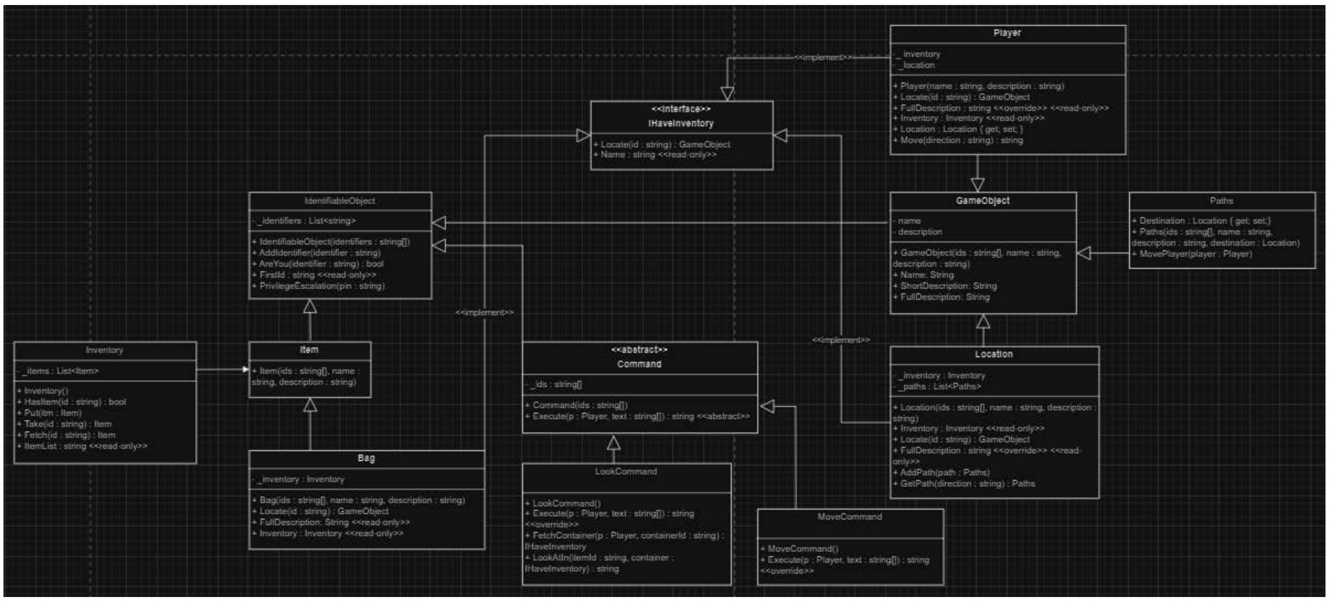
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _9._2C
{
    public class GameObject : IdentifiableObject
    {
        private string _name;
        private string _description;

        public GameObject(string[] ids, string name, string description) : base(ids)    //call constructor of the base class
        {
            _name = name;
            _description = description;
        }

        public string Name
        {
            get { return _name; }
        }

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

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

```



```
Test Explorer
41  4  0  37                                   Search (Ctrl+I)
Discovering tests: Found 3 tests                          0 Warnings  0 Errors
Test                                    Duration  Traits  Error Message        Run | Debug
 BagTest (5)
 IdentifiableObjectTest (7)
 InventoryTest (5)
 ItemTest (4)
 LocationTest (3)
 LookCommandTest (8)
 PathTest (4)                           49 ms
   PathTest (4)                         49 ms
     Tests (4)                          49 ms
       TestGetPathFromLocation          46 ms
       TestPathCanMovePlayer            < 1 ms
       TestPlayerCanLeaveLocation       < 1 ms
       TestPlayerRemainInLocation       3 ms
 PlayerTest (5)
```

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

namespace _9._2C
{
    public class IdentifiableObject
    {
        private List<string> _identifiers = new List<string>();
        public IdentifiableObject(string[] identifiers)
        {


            foreach (string id in identifiers)
            {
                AddIdentifier(id);
            }
        }

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

        public bool AreYou(string identifier)
        { return _identifiers.Contains(identifier.ToLower()); }

        public string FirstId
        {
            get
            {
                if (_identifiers.Count > 0)
                {
                    return _identifiers[0];
                }
                else
                {
                    return "";
                }
            }
        }



        public void PrivilegeEscalation(string pin)
        {
            if (pin == "2183" && _identifiers.Count > 0)
            {
                _identifiers[0] = "7";
            }
        }
        }
    }

```


User
Player
MoveCommand
Location
Paths
Enter "move east"
Execute (players, words)
Validate Command
Get current location
GetPath("east")
Check direction and return matching paths
MovePlayer(player)
Move player to Destination (mountain2)
Display result
Return result
Update player.Location


Player
- _inventory
- _location
+ Player(name : string, description : string)
+ Locate(id : string) : GameObject
+ FullDescription : string <<override>> <<read-only>>
+ Inventory : Inventory <<read-only>>
+ Location : Location { get; set; }
+ Move(direction : string) : string
<<implement>>
<<interface>>
IHaveInventory
+ Locate(id : string) : GameObject
+ Name : string <<read-only>>
IdentifiableObject
- _identifiers : List<string>
+ IdentifiableObject(identifiers : string[])
+ AddIdentifier(identifier : string)
+ AreYou(identifier : string) : bool
+ FirstId : string <<read-only>>
+ PrivilegeEscalation(pin : string)
GameObject
- name
- description
+ GameObject(ids : string[], name : string, description : string)
+ Name: String
+ ShortDescription: String
+ FullDescription: String
Paths
+ Destination : Location { get; set;}
+ Paths(ids : string[], name : string, description : string, destination : Location)
+ MovePlayer(player : Player)
<<implement>>
<<implement>>
Inventory
- _items : List<Item>
+ Inventory()
+ HasItem(id : string) : bool
+ Put(itm : Item)
+ Take(id : string) : Item
+ Fetch(id : string) : Item
+ ItemList : string <<read-only>>
Item
+ Item(ids : string[], name : string, description : string)
<<abstract>>
Command
- _ids : string[]
+ Command(ids : string[])
+ Execute(p : Player, text : string[]) : string <<abstract>>
Location
- _inventory : Inventory
- _paths : List<Paths>
+ Location(ids : string[], name : string, description : string)
+ Inventory : Inventory <<read-only>>
+ Locate(id : string) : GameObject
+ FullDescription : string <<override>> <<read-only>>
+ AddPath(path : Paths)
+ GetPath(direction : string) : Paths
Bag
- _inventory : Inventory
+ Bag(ids : string[], name : string, description : string)
+ Locate(id : string) : GameObject
+ FullDescription: String <<read-only>>
+ Inventory : Inventory <<read-only>>
LookCommand
+ LookCommand()
+ Execute(p : Player, text : string[]) : string <<override>>
+ FetchContainer(p : Player, containerId : string) : IHaveInventory
+ LookAtIn(itemId : string, container : IHaveInventory) : string
MoveCommand
+ MoveCommand()
+ Execute(p : Player, text : string[]) : string <<override>>

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

namespace _9._2C
{
    public interface IHaveInventory
    {
        GameObject Locate(string id); //locate item
        string Name { get; } //a name property
    }
}

```

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

namespace _9._2C
{
    public class Inventory
    {
        private List<Item> _items = new List<Item>();
        public Inventory() { }
        public bool HasItem(string id)
        {
            return Fetch(id) != null;
        }
        public void Put(Item itm)
        {
            _items.Add(itm);
        }
        public Item Take(string id)
        {
            Item item = Fetch(id);
            if (item != null)
            {
                _items.Remove(item);
            }
            return item;
        }

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

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

        }
    }

```

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

namespace _9._2C
{
    public class Item : GameObject
    {
        public Item(string[] ids, string name, string description) :
          base(ids, name, description)
        {

        }
    }
}

```

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

namespace _9._2C
{
    public class Location : GameObject, IHaveInventory
    {
        private Inventory _inventory;
        private List<Paths> _paths;
        public Location(string[] ids, string name, string description) : base(ids, name, description)
        {
            _inventory = new Inventory();
            _paths = new List<Paths>();
        }

        public Inventory Inventory
        { get { return _inventory; } }

        public GameObject Locate(string id) //the purpose is to return the gameobject itself
        {
            if (AreYou(id))
                {  return this; }
            return _inventory.Fetch(id);

        }

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

        public void AddPath(Paths path)
        {

                _paths.Add(path);

        }

        public Paths GetPath(string direction)
        {
            foreach (Paths path in _paths)
            {
                if (path.AreYou(direction))
```

```
                    {
                        return path;
                    }
                }
                return null;
            }


        }
    }

```

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

namespace _9._2C
{
    public class LookCommand : Command
    {
        public LookCommand () : base(new string[] {"look"})
        {
        }

        public override string Execute(Player p, string[] text)
        {
            if (text.Length != 3 && text.Length != 5)
            {
                return "I don't know how to look like that";
            }

            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?";
            }

            IHaveInventory container;
            if (text.Length == 3)
            {

                container = FetchContainer(p, "inventory");
            }

            else
            {

                container = FetchContainer(p, text[4]);
                if (container == null)
                {
                    return $"I cannot find the {text[4]}";
                }
            }

            // Step 7: The item id is the 3rd word
            string itemId = text[2];
            return LookAtIn(itemId, container);

        }
        public IHaveInventory FetchContainer(Player p, string 
          containerId)
        {
            if (containerId.ToLower() == "inventory")
            {
                return p;
            }

            GameObject obj = p.Locate(containerId);
            if (obj is IHaveInventory)
            {
                return (IHaveInventory)obj; //container is bag(?)
            }
            return null;
        }

        public string LookAtIn(string itemId, IHaveInventory container)
        {
            // Try to locate the item within the specified container
            GameObject item = container.Locate(itemId);
            if (item == null)
            {
                return $"I cannot find the {itemId} in 
                  {container.Name}";
            }
            ////

            // Return the item's full description if found
            return item.FullDescription;
        }
    }
}
```

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

namespace _9._2C
{
    public class MoveCommand : Command
    {
        public MoveCommand() : base(new string[] {"move", "head", "go", "leave"})
        { }

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

            if (!(new string[] { "move", "go", "head", "leave" }).Contains(text.ElementAt(0))) //If not those words, ask again
            {
                return "Where would you like to move?";
            }

            string direction = text[1];

            Paths path = p.Location.GetPath(direction);
            if (path == null)
            {
                return $"There is no path to the {direction}.";
            }
            path.MovePlayer(p);
            return $"You move {direction} to {p.Location.Name}.";

        }
    }
}

```

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

namespace _9._2C
{
    public class Paths : GameObject
    {
        public Location Destination { get; set; }

        public Paths(string[] ids, string name, string description, 
          Location destination) : base(ids, name, description)
        {
            Destination = destination;
        }

        public void MovePlayer(Player player)
        {
            player.Location = Destination;
        }
    }
}
```

```csharp
using _9._2C;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using NUnit.Framework;
using System.Numerics;

namespace PathTest
{

    [TestFixture]
    public class Tests
    {
        private Location _mountain1;
        private Location _mountain2;
        private Player _player;
        private Item _sword;
        private MoveCommand _moveCommand;
        private Paths _pathToMountain1;
        private Paths _pathToMountain2;
        [SetUp]
        public void Setup()
        {

            _mountain1 = new Location(new string[] { "mountain1" }, "Mountain 1", "first mountain");
            _mountain2 = new Location(new string[] { "mountain2" }, "Mountain 2", "second mountain");
            _sword = new Item(new string[] { "sword" }, "Excalibur", "a strong sword");
            _mountain1.Inventory.Put(_sword);


            _pathToMountain1 = new Paths(new string[] { "west" }, "Journey to the West", "path leading West", _mountain1);
            _pathToMountain2 = new Paths(new string[] { "east" }, "Journey to the East", "path leading East", _mountain2);

            _mountain1.AddPath(_pathToMountain2);
            _mountain2.AddPath(_pathToMountain1);
            _player = new Player("Wukong", "The monkey");
            _player.Location = _mountain1;


            _moveCommand = new MoveCommand();
        }

        [Test]
        public void TestPathCanMovePlayer()
        {
            string result = _moveCommand.Execute(_player, new string[] { "move", "east" });
            Assert.AreEqual(_mountain2, _player.Location);

        }

        [Test]
        public void TestGetPathFromLocation()
        {
            Paths path = _mountain1.GetPath("east");
            Assert.IsNotNull(path);
            Assert.AreEqual(_mountain2, path.Destination);
        }

        [Test]
        public void TestPlayerCanLeaveLocation()
        {
            _moveCommand.Execute(_player, new string[] { "head", "east" });
            Assert.AreEqual(_mountain2, _player.Location);


        }

        [Test]
        public void TestPlayerRemainInLocation()
        {
            _moveCommand.Execute(_player, new string[] { "head", "north" });
            Assert.AreNotEqual(_mountain2, _player.Location);
        }
    }
}
```

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

namespace _9._2C
{
    public class Player : GameObject, IHaveInventory
    {
        private Inventory _inventory = new Inventory();
        private Location _location;

        public Player(string name, string description) : base(new string [] { "me", "inventory" }, name, description) { } //name and des gotten from GameObject
        //help the class identify itself and its item, 3 batteries, 2 from GO and 1 from IO
        public GameObject Locate(string id)
        {
            if (AreYou(id))
            {
                return this; //return then player object itself
            }
            GameObject item = _inventory.Fetch(id); // Fetch the item from the inventory if it exists.
            if (item != null)
            {
                return item; // Return the item if found in the inventory.
            }
            //Check for location if not found in inventory
            if (_location != null)
            {
                return _location.Locate(id); //instead of returning null like the first time, this time it will look for the location
            }
            return null;
        }

        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 Location
```

```
            {
                get { return _location; }
                set { _location = value; }
            }

        public string Move(string direction)
        {
            MoveCommand moveCommand = new MoveCommand();
            return moveCommand.Execute(this, new string[] { "move",
              direction });
                                        //instance of player
        }


    }
}
```

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

namespace _9._2C
{
    public class Program
    {
        public static void Main(string[] args)
        {
            /**
            Console.WriteLine("Enter your name: ");
            string playerName = Console.ReadLine();
            Console.WriteLine("Enter your description: ");
            string playerDescription = Console.ReadLine();
            Player player = new Player(playerName, playerDescription);

            Item sword = new Item(new string[] {"sword"}, "Excalibur", 
              "a strong sword");
            Item shield = new Item(new string[] {"shield"}, "Aegis", "a 
              strong shield");

            player.Inventory.Put(sword);
            player.Inventory.Put(shield);

            Bag backpack = new Bag(new string[] { "backpack" }, 
              "Adidas", "a big backpack");
            player.Inventory.Put(backpack);
            Item gem = new Item(new string[] { "gem" }, "Ruby", "a rare 
              gem");
            backpack.Inventory.Put(gem);

            /////
            LookCommand lookCommand = new LookCommand();
            while (true)
            {
                Console.Write("What do you want to look at?: ");
                string input = Console.ReadLine();
                string[] commandWords = input.Split(' ');
                string result = lookCommand.Execute(player, 
                  commandWords);
                Console.WriteLine(result);
            }
            **/

            Location mountain1 = new Location(new string[]
              { "mountain1" }, "Mountain 1", "first mountain");
            Location mountain2 = new Location(new string[]
              { "mountain2" }, "Mountain 2", "second mountain");
            Item sword = new Item(new string[] { "sword" }, "Excalibur", 
               "a strong sword");
```

```csharp
            mountain1.Inventory.Put(sword);


            Paths pathToMountain1 = new Paths(new string[] { "west" }, "Journey to the West", "path leading West", mountain1);
            Paths pathToMountain2 = new Paths(new string[] { "east" }, "Journey to the East", "path leading East", mountain2);

            mountain1.AddPath(pathToMountain2);
            mountain2.AddPath(pathToMountain1);

            Player player = new Player("Wukong", "The monkey");
            player.Location = mountain1;
            player.Location = mountain2;

            MoveCommand moveCommand = new MoveCommand();
            while (true)
            {
                Console.WriteLine(player.Location.FullDescription);
                Console.Write("Enter command: ");
                string input = Console.ReadLine();
                string[] commandWords = input.Split(' '); //split into [0] and [1]
                string result = moveCommand.Execute(player, commandWords);
                Console.WriteLine(result);
            }

        }
    }
}

```