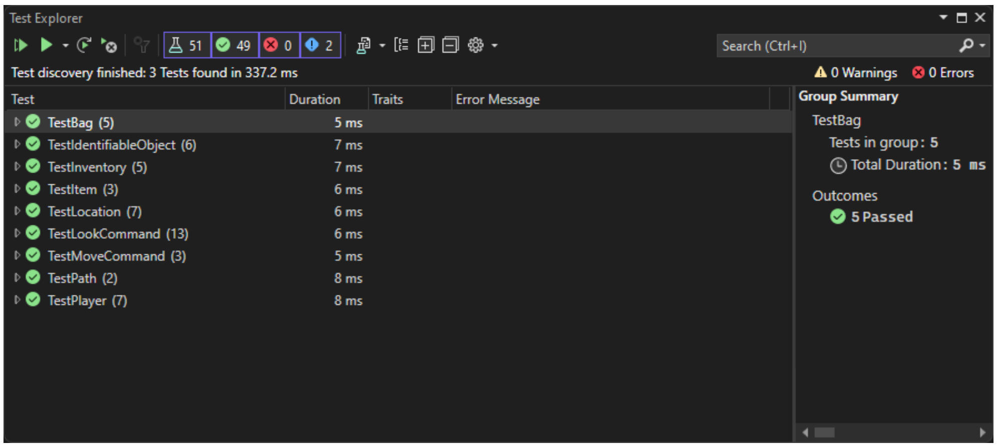
9.2C - Case Study - Iteration 7 – Paths

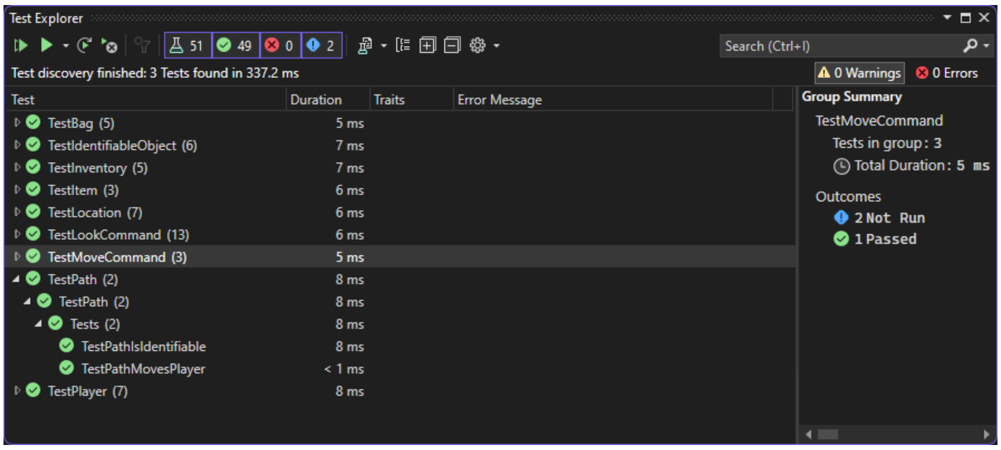
Jayden Kong, 104547242

<u>All tests passing:</u>



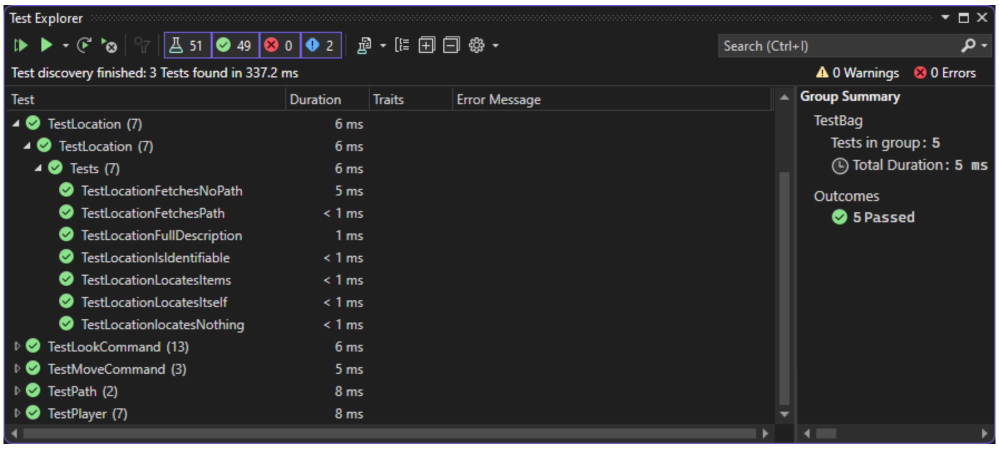
```
Test Explorer
51  49  0  2                                              Search (Ctrl+I)
Test discovery finished: 3 Tests found in 337.2 ms        0 Warnings  0 Errors
Test                          Duration  Traits  Error Message    Group Summary
TestBag (5)                       5 ms                             TestBag
TestIdentifiableObject (6)        7 ms                               Tests in group: 5
TestInventory (5)                 7 ms                               Total Duration: 5 ms
TestItem (3)                      6 ms                             Outcomes
TestLocation (7)                  6 ms                               5 Passed
TestLookCommand (13)              6 ms
TestMoveCommand (3)               5 ms
TestPath (2)                      8 ms
TestPlayer (7)                    8 ms
```

<u>Path tests:</u>



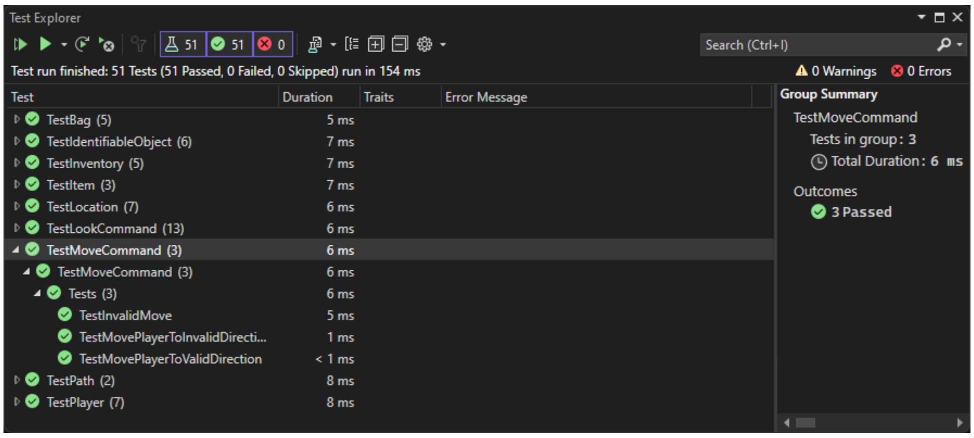
```
Test Explorer
51  49  0  2                                              Search (Ctrl+I)
Test discovery finished: 3 Tests found in 337.2 ms        0 Warnings  0 Errors
Test                          Duration  Traits  Error Message    Group Summary
TestBag (5)                       5 ms                             TestMoveCommand
TestIdentifiableObject (6)        7 ms                               Tests in group: 3
TestInventory (5)                 7 ms                               Total Duration: 5 ms
TestItem (3)                      6 ms                             Outcomes
TestLocation (7)                  6 ms                               2 Not Run
TestLookCommand (13)              6 ms                               1 Passed
TestMoveCommand (3)               5 ms
TestPath (2)                      8 ms
  TestPath (2)                    8 ms
    Tests (2)                     8 ms
      TestPathIsIdentifiable      8 ms
      TestPathMovesPlayer       < 1 ms
TestPlayer (7)                    8 ms
```

<u>Location tests:</u>



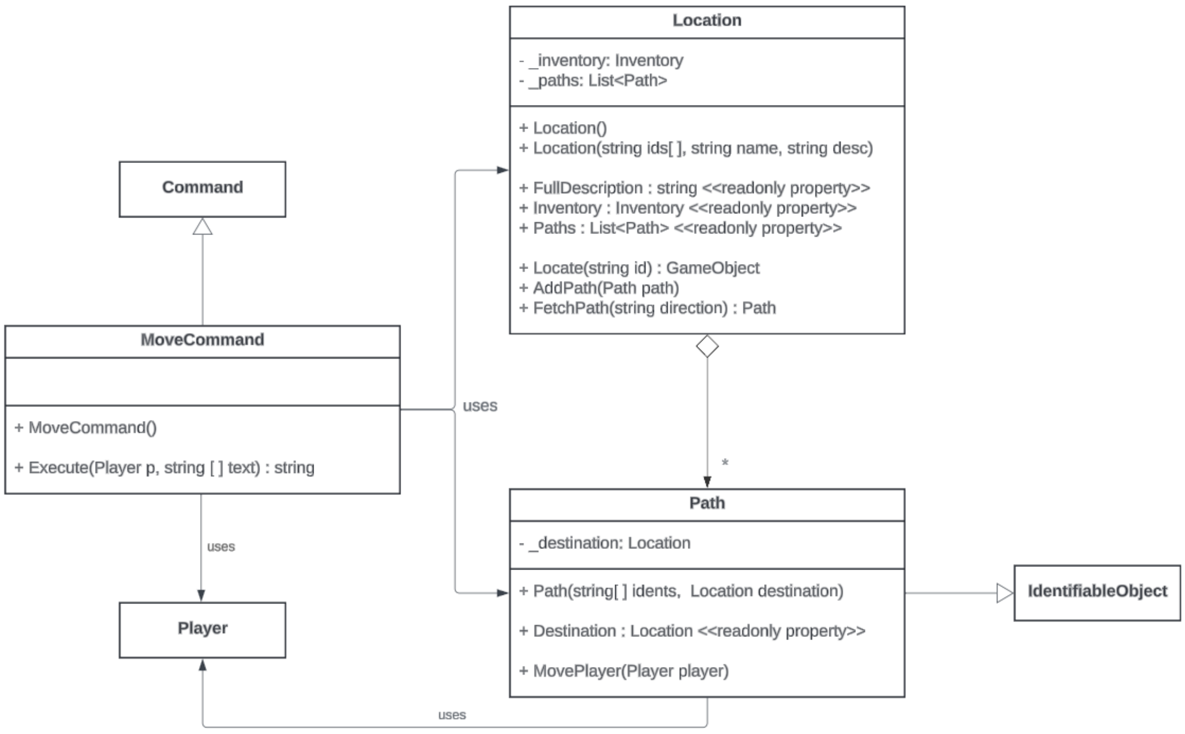
```
Test Explorer
51  49  0  2                                              Search (Ctrl+I)
Test discovery finished: 3 Tests found in 337.2 ms        0 Warnings  0 Errors
Test                          Duration  Traits  Error Message    Group Summary
TestLocation (7)                  6 ms                             TestBag
  TestLocation (7)                6 ms                               Tests in group: 5
    Tests (7)                     6 ms                               Total Duration: 5 ms
      TestLocationFetchesNoPath   5 ms                             Outcomes
      TestLocationFetchesPath   < 1 ms                               5 Passed
      TestLocationFullDescription 1 ms
      TestLocationIsIdentifiable < 1 ms
      TestLocationLocatesItems  < 1 ms
      TestLocationLocatesItself < 1 ms
      TestLocationlocatesNothing < 1 ms
TestLookCommand (13)              6 ms
TestMoveCommand (3)               5 ms
TestPath (2)                      8 ms
TestPlayer (7)                    8 ms
```

Move command tests:

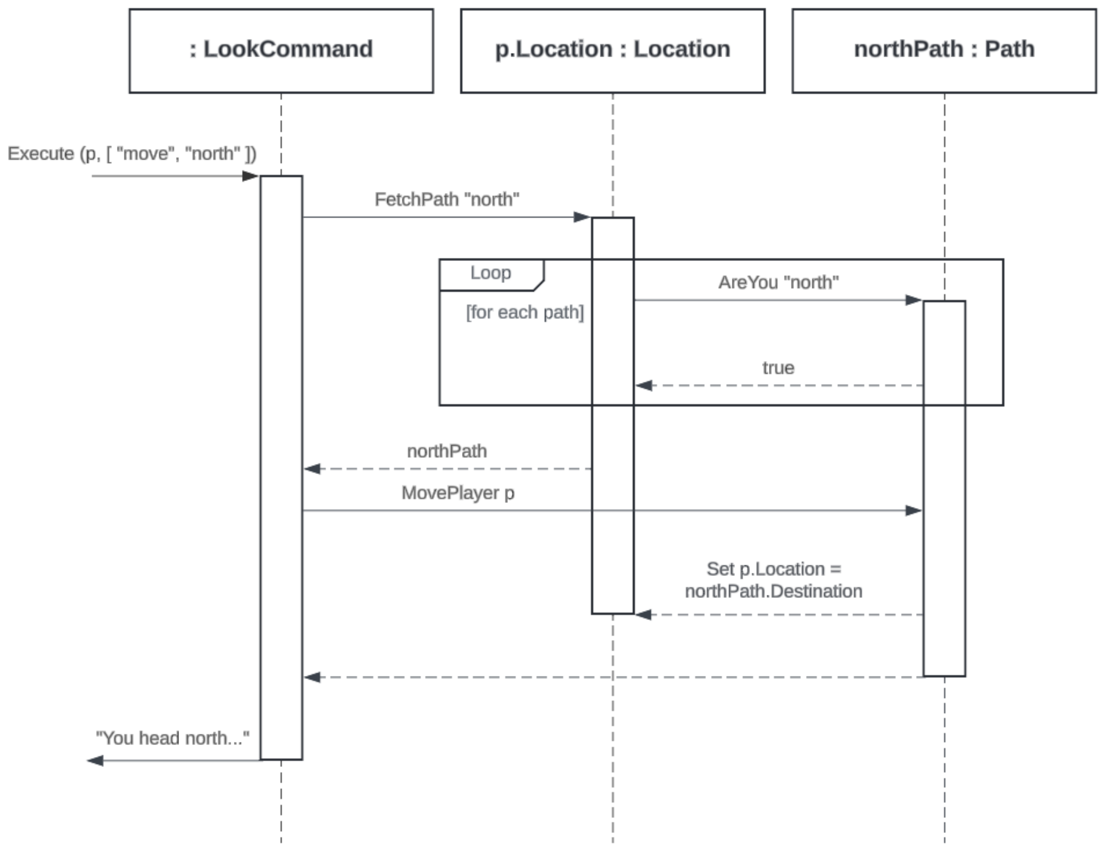


```
Test Explorer
51  51  0                                                    Search (Ctrl+I)
Test run finished: 51 Tests (51 Passed, 0 Failed, 0 Skipped) run in 154 ms      0 Warnings   0 Errors
Test                                   Duration   Traits   Error Message       Group Summary
  TestBag (5)                              5 ms                                  TestMoveCommand
  TestIdentifiableObject (6)               7 ms                                    Tests in group: 3
  TestInventory (5)                        7 ms                                    Total Duration: 6 ms
  TestItem (3)                             7 ms                                  Outcomes
  TestLocation (7)                         6 ms                                    3 Passed
  TestLookCommand (13)                     6 ms
  TestMoveCommand (3)                      6 ms
    TestMoveCommand (3)                    6 ms
      Tests (3)                            6 ms
        TestInvalidMove                    5 ms
        TestMovePlayerToInvalidDirecti...  1 ms
        TestMovePlayerToValidDirection   < 1 ms
  TestPath (2)                             8 ms
  TestPlayer (7)                           8 ms
```

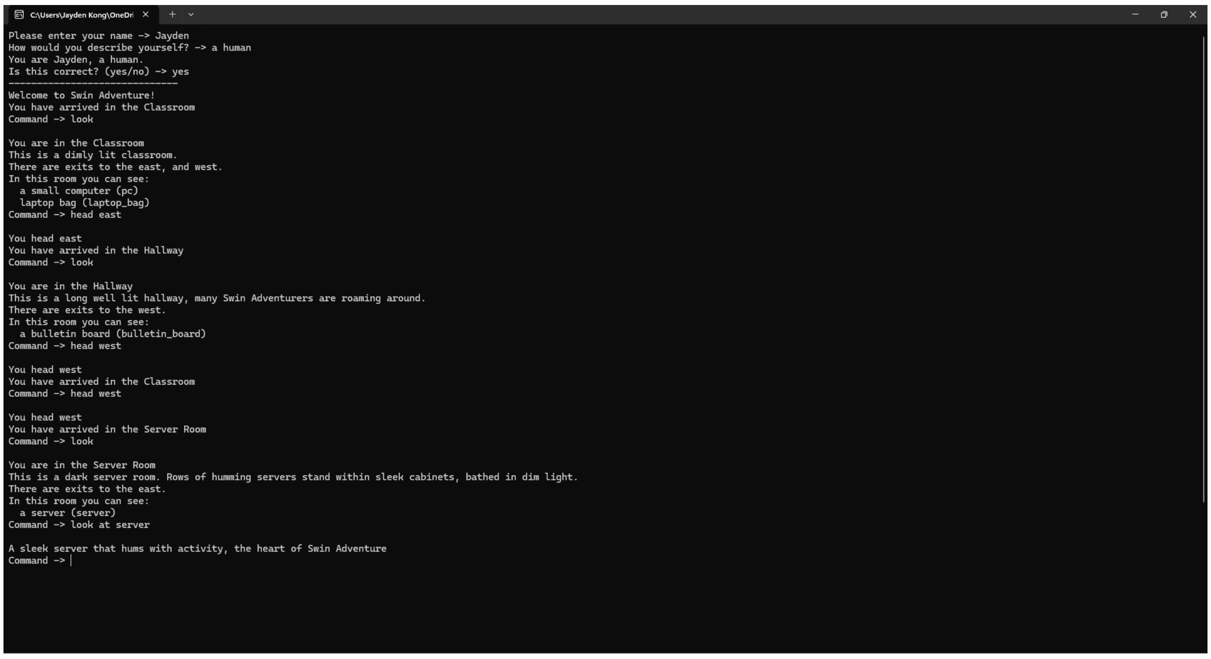
UML Class Diagram:


Location
- _inventory: Inventory
- _paths: List<Path>
+ Location()
+ Location(string ids[ ], string name, string desc)
+ FullDescription : string <<readonly property>>
+ Inventory : Inventory <<readonly property>>
+ Paths : List<Path> <<readonly property>>
+ Locate(string id) : GameObject
+ AddPath(Path path)
+ FetchPath(string direction) : Path
Command
MoveCommand
+ MoveCommand()
+ Execute(Player p, string [ ] text) : string
uses
uses
*
Path
- _destination: Location
+ Path(string[ ] idents, Location destination)
+ Destination : Location <<readonly property>>
+ MovePlayer(Player player)
IdentifiableObject
Player
uses

UML Sequence Diagram:



Program running:



```
Please enter your name -> Jayden
How would you describe yourself? -> a human
You are Jayden, a human.
Is this correct? (yes/no) -> yes
------------------------------
Welcome to Swin Adventure!
You have arrived in the Classroom
Command -> look

You are in the Classroom
This is a dimly lit classroom.
There are exits to the east, and west.
In this room you can see:
  a small computer (pc)
  laptop bag (laptop_bag)
Command -> head east

You head east
You have arrived in the Hallway
Command -> look

You are in the Hallway
This is a long well lit hallway, many Swin Adventurers are roaming around.
There are exits to the west.
In this room you can see:
  a bulletin board (bulletin_board)
Command -> head west

You head west
You have arrived in the Classroom
Command -> head west

You head west
You have arrived in the Server Room
Command -> look

You are in the Server Room
This is a dark server room. Rows of humming servers stand within sleek cabinets, bathed in dim light.
There are exits to the east.
In this room you can see:
  a server (server)
Command -> look at server

A sleek server that hums with activity, the heart of Swin Adventure
Command -> |
```

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

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

        public Location Destination
        {
            get
            {
                return _destination;
            }
        }

        public Path(string[] idents, Location destination) : base
          (idents)
        {
            _destination = destination;
        }

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

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

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)
            {
                return "I don't know how to move like that";
            }

            if (!AreYou(text[0]))
            {
                return "Error in move input";
            }

            Path? path;
            string direction;
            switch (text[1])
            {
                case string north when north == "north" || north == "n":
                    direction = "north";
                    break;
                case string northEast when northEast == "north_east" || northEast == "ne":
                    direction = "north_east";
                    break;
                case string northWest when northWest == "north_west" || northWest == "nw":
                    direction = "north_west";
                    break;
                case string south when south == "south" || south == "s":
                    direction = "south";
                    break;
                case string southEast when southEast == "south_east" || southEast == "se":
                    direction = "south_east";
                    break;
                case string southWest when southWest == "south_west" || southWest == "sw":
                    direction = "south_west";
                    break;
                case string east when east == "east" || east == "e":
                    direction = "east";
                    break;
                case string west when west == "west" || west == "w":
                    direction = "west";
                    break;
                case "up":
                    direction = text[1];
                    break;
                case "down":
                    direction = text[1];
                    break;
                default:
                    return "I don't know that direction";
            }
            path = p.Location.FetchPath(direction);

            if (path == null)
            {
                return string.Format("I cannot move {0}", text[1]);
            }

            path.MovePlayer(p);
            return string.Format("You head {0}\nYou have arrived in
              {1}", path.FirstID, path.Destination.Name);
        }
    }
}
```

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

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

        public override string FullDescription
        {
            get
            {
                string description = string.Format("You are in {0}\n{1}", Name, base.FullDescription);
                if (Paths.Count > 0)
                {
                    description += "\nThere are exits to the ";
                    int lastIndex = Paths.Count - 1;
                    int i = 0;
                    foreach (Path path in Paths)
                    {
                        if (Paths.Count > 1)
                        {
                            if (i < lastIndex)
                            {
                                description += string.Format("{0}, ", path.FirstID);
                            }
                            else
                            {
                                description += string.Format("and {0}.", path.FirstID);
                            }
                            i++;
                        }
                        else
                        {
                            description += string.Format("{0}.", path.FirstID);
                        }
                    }
                }
                description += string.Format("\nIn this room you can see:{0}", Inventory.ItemList);
                return description;
            }
        }
        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

        public List<Path> Paths
        {
            get
            {
                return _paths;
            }
        }

        public Location() : this(new string[] { "location", "unknown" }, "an unknown location", "This is a mysterious location") { }            // Default constructor, to make sure the player has a location if not allocated one
        public Location(string[] ids, string name, string desc) : base(ids, name, desc)
        {
            AddIdentifier("room");
            AddIdentifier("here");
            AddIdentifier("location");
            _inventory = new Inventory();
            _paths = new List<Path>();
        }

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

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

        public Path? FetchPath(string direction)
        {
            foreach (Path path in Paths)
            {
                if (path.AreYou(direction))
                {
                    return path;
                }
            }
            return null;
        }
    }
}
```

```csharp
using System.Diagnostics;
using System.Runtime.InteropServices;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace SwinAdventure
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Player? player = null;
            Command lookCommand = new LookCommand();
            Command moveCommand = new MoveCommand();

            // Player creation menu
            while (player == null)
            {
                Console.Write("Please enter your name -> ");
                string? playerName = Console.ReadLine();
                Console.Write("How would you describe yourself? -> ");
                string? playerDescription = Console.ReadLine();
                Console.Write("You are {0}, {1}.\nIs this correct? (yes/no) -> ", playerName, playerDescription);
                bool confirmationMenuLoop = true;
                while (confirmationMenuLoop)
                {
                    string? decision = Console.ReadLine().ToLower();
                    switch (decision)
                    {
                        case "yes":
                            player = new Player(playerName, playerDescription);
                            confirmationMenuLoop = false;
                            break;
                        case "no":
                            confirmationMenuLoop = false;
                            break;
                        default:
                            Console.Write("Invalid option: please enter yes or no. -> ");
                            break;
                    }
                }
            }

            // Create Items
            Item shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "A sturdy shovel, the perfect tool for digging");
            Item bronzeSword = new Item(new string[] { "sword" }, "a bronze sword", "A short sword forged from bronze");
            Item ruby = new Item(new string[] { "gem", "ruby" }, "a red gem", "A brilliant ruby, glows with a fiery red hue");
            Item computer = new Item(new string[] { "pc", "computer" }, "a small computer", "A dusty PC with a flickering screen");
            Item laptop = new Item(new string[] { "laptop" }, "a laptop", "A compact, modern laptop with a matte black finish");
            Item bulletinBoard = new Item(new string[] { "bulletin_board" }, "a bulletin board", "A small bulletin board filled with announcements, flyers, and posters");
            Item server = new Item(new string[] { "server" }, "a server", "A sleek server that hums with activity, the heart of Swin Adventure");

            // Create Bags
            Bag laptopBag = new Bag(new string[] { "laptop_bag", "bag" }, "laptop bag", "A sleek, black laptop bag. Its fabric is slightly worn from use");
            Bag bag = new Bag(new string[] { "bag" }, "leather bag", "A small bag crafted from supple brown leather, perfect for carrying items");

            // Create Locations
            Location classroom = new Location(new string[] { "classroom" }, "the Classroom", "This is a dimly lit classroom.");        // Player will initially be in the classroom
            Location hallway = new Location(new string[] { "hallway" }, "the Hallway", "This is a long well lit hallway, many Swin Adventurers are roaming around.");
            Location serverRoom = new Location(new string[] { "server_room" }, "the Server Room", "This is a dark server room. Rows of humming servers stand within sleek cabinets, bathed in dim light.");

            // Create Paths
            Path classroomToHallway = new Path(new string[] { "east", "hallway" }, hallway);
            Path hallwayToClassroom = new Path(new string[] { "west", "classroom" }, classroom);

            Path classroomToServerRoom = new Path(new string[] { "west", "server_room" }, serverRoom);
            Path serverRoomToClassroom = new Path(new string[] { "east", "classroom" }, classroom);

            // Add Paths to Locations
            classroom.AddPath(classroomToHallway);
            classroom.AddPath(classroomToServerRoom);
            hallway.AddPath(hallwayToClassroom);
            serverRoom.AddPath(serverRoomToClassroom);

            // Set Player Location
            player.Location = classroom;

            // Distribute Items to Player
            bag.Inventory.Put(ruby);
            player.Inventory.Put(shovel);
            player.Inventory.Put(bronzeSword);
            player.Inventory.Put(bag);

            // Distribute items to Classroom
            laptopBag.Inventory.Put(laptop);
            classroom.Inventory.Put(computer);
            classroom.Inventory.Put(laptopBag);

            // Distribute items to Hallway
            hallway.Inventory.Put(bulletinBoard);

            // Distribute items to Server Room
            serverRoom.Inventory.Put(server);


            // Introduction text
            Console.WriteLine("------------------------------");
            Console.WriteLine("Welcome to Swin Adventure!");
            Console.WriteLine("You have arrived in {0}",
              player.Location.Name);

            // Game loop
            bool gameLoop = true;
            while (gameLoop)
            {
                Console.Write("Command -> ");
                string? playerInput = Console.ReadLine();
                string[] inputToPass = playerInput.Split(new char[]
                  {  }, StringSplitOptions.RemoveEmptyEntries);
                Console.WriteLine("");
                foreach (string input in inputToPass)
                {
                    if (lookCommand.AreYou(input))
                    {
                        Console.WriteLine(lookCommand.Execute(player,
                  inputToPass));
                    }
                    else if (moveCommand.AreYou(input))
                    {
                        Console.WriteLine(moveCommand.Execute(player,
                  inputToPass));
                    }
                }
            }
        }
    }
}
```

```
using SwinAdventure;

namespace TestPath
{
    public class Tests
    {
        private Location hallway;
        private SwinAdventure.Path testPathToHallway;
        private Player p;

        [SetUp]
        public void Setup()
        {
            hallway = new Location(new string[] { "hallway" }, "the 
              Hallway", "This is a long well lit hallway, many Swin 
              Adventurers are roaming around.");
            testPathToHallway = new SwinAdventure.Path(new string[] 
              { "east", "hallway" }, hallway);
            p = new Player("Tester", "the mighty test player");
        }

        [Test]
        public void TestPathIsIdentifiable()
        {
            bool testPathIsIdentifiable = testPathToHallway.AreYou
              ("east");
            Assert.That(testPathIsIdentifiable, Is.EqualTo(true));
        }

        [Test]
        public void TestPathMovesPlayer()
        {
            testPathToHallway.MovePlayer(p);
            Assert.That(p.Location, Is.EqualTo(hallway));
        }
    }
}
```

```csharp
using SwinAdventure;

namespace TestMoveCommand
{
    public class Tests
    {
        private MoveCommand move;
        private Player testPlayer;
        private Location location;
        private Location hallway;
        private SwinAdventure.Path testPathToHallway;


        [SetUp]
        public void Setup()
        {
            move = new MoveCommand();
            testPlayer = new Player("testPlayer", "test player description");
            location = new Location(new string[] { "location" }, "the Location", "This is a test location");
            hallway = new Location(new string[] { "hallway" }, "the Hallway", "This is a long well lit hallway, many Swin Adventurers are roaming around.");
            testPathToHallway = new SwinAdventure.Path(new string[] { "east", "hallway" }, hallway);
            testPlayer.Location = location;
            location.AddPath(testPathToHallway);
        }

        [Test]
        public void TestMovePlayerToValidDirection()
        {
            string testMoveToDirection = move.Execute(testPlayer, new string[] { "move", "east" });
            Assert.That(testMoveToDirection, Is.EqualTo("You head east\nYou have arrived in the Hallway"));
            Assert.That(testPlayer.Location, Is.EqualTo(hallway));
        }

        [Test]
        public void TestMovePlayerToInvalidDirection()
        {
            string testMoveToDirection = move.Execute(testPlayer, new string[] { "move", "west" });
            Assert.That(testMoveToDirection, Is.EqualTo("I cannot move west"));
            Assert.That(testPlayer.Location, Is.EqualTo(location));
        }

        [Test]
        public void TestInvalidMove()
        {
            string testTextLengthNot2 = move.Execute(testPlayer, new string[] { "move", "test", "length", "not", "two" });
            string testMoveIsNotFirstWord = move.Execute(testPlayer, new string[] { "test", "move" });
            string testMoveToUnknownDirection = move.Execute(testPlayer, new string[] { "move", "unknown" });

            Assert.That(testTextLengthNot2, Is.EqualTo("I don't know how to move like that"));
            Assert.That(testMoveIsNotFirstWord, Is.EqualTo("Error in move input"));
            Assert.That(testMoveToUnknownDirection, Is.EqualTo("I don't know that direction"));
            Assert.That(testPlayer.Location, Is.EqualTo(location));                     // Player stays in current location
        }
    }
}
```

```csharp
using SwinAdventure;

namespace TestLocation
{
    public class Tests
    {
        private Location location;
        private Item ruby;
        private SwinAdventure.Path testPathToHallway;
        private Location hallway;


        [SetUp]
        public void Setup()
        {
            location = new Location(new string[] { "location" }, "the Location", "This is a test location");
            hallway = new Location(new string[] { "hallway" }, "the Hallway", "This is a long well lit hallway, many Swin Adventurers are roaming around.");
            testPathToHallway = new SwinAdventure.Path(new string[] { "east", "hallway" }, hallway);
            ruby = new Item(new string[] { "gem", "ruby" }, "a red gem", "A brilliant ruby, glows with a fiery red hue");
            location.Inventory.Put(ruby);
        }

        [Test]
        public void TestLocationIsIdentifiable()
        {
            GameObject? testLocationId = location.Locate("location");
            Assert.That(testLocationId, Is.EqualTo(location));
        }

        [Test]
        public void TestLocationLocatesItems()
        {
            GameObject? testLocationLocatesRuby = location.Locate("ruby");
            Assert.That(testLocationLocatesRuby, Is.EqualTo(ruby));
        }

        [Test]
        public void TestLocationLocatesItself()
        {
            GameObject? testLocationLocatesItself = location.Locate("location");
            Assert.That(testLocationLocatesItself, Is.EqualTo(location));
        }

        [Test]
        public void TestLocationlocatesNothing()
        {
            GameObject? testLocationLocatesItself = location.Locate("nothing");
            Assert.That(testLocationLocatesItself, Is.EqualTo(null));
        }

        [Test]
        public void TestLocationFullDescription()
        {
            location.AddPath(testPathToHallway);
            string testLocationFullDescription = location.FullDescription;
            Assert.That(testLocationFullDescription, Is.EqualTo("You are in the Location\nThis is a test location\nThere are exits to the east.\nIn this room you can see:\n  a red gem (gem)"));
        }

        [Test]
        public void TestLocationFetchesPath()
        {
            location.AddPath(testPathToHallway);
            SwinAdventure.Path? testFetchPath = location.FetchPath("east");
            Assert.That(testFetchPath, Is.EqualTo(testPathToHallway));
        }

        [Test]
        public void TestLocationFetchesNoPath()
        {
            SwinAdventure.Path? testFetchPath = location.FetchPath("east");
            Assert.That(testFetchPath, Is.EqualTo(null));
        }
    }
}
```

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

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

        public override string FullDescription
        {
            get
            {
                return string.Format("You are {0}, {1}.\nYou are 
                  carrying: {2}", Name, base.FullDescription, 
                  Inventory.ItemList);
            }
        }

        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

        public Location Location
        {
            get
            {
                return _location;
            }
            set
            {
                _location = value;
            }
        }

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

        public GameObject? Locate(string id)
        {
            if (AreYou(id))
            {
                return this;
            }
            GameObject? item = Inventory.Fetch(id);
            if (item != null)
            {
                return item;
            }
            item = Location.Locate(id);
            return item;
        }
    }
}

```

```
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) { }
    }
}

```

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

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

        public override string Execute(Player p, string[] text)
        {
            if (!(text.Length == 1 || 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.Length == 1)
            {
                string locationDescription = p.Location.FullDescription;
                return locationDescription;
            }

            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)
            {
                string? itemDescription3 = LookAtIn(text[2], p);
                if (itemDescription3 == null)
                {
                    return string.Format("I cannot find the {0}", text 
                  [2]);
                }
                return itemDescription3;
            }

            // By this point the 1 and 3 element look commands are done
            IHaveInventory? container = FetchContainer(p, text[4]);
            if (container == null)
            {
                return string.Format("I cannot find the {0}", text[4]);
            }

            string? itemDescription5 = LookAtIn(text[2], container);
            if (itemDescription5 == null)
            {
                return string.Format("I cannot find the {0} in the {1}", 
                  text[2], text[4]);
            }
            return itemDescription5;
        }

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

        private string? LookAtIn(string thingId, IHaveInventory 
          container)
        {
            GameObject? item = container.Locate(thingId);
            if (item == null)
            {
                return null;
            }
            return item.FullDescription;
        }
    }
}

```

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

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

        public string Name
        {
            get
            {
                return _name;
            }
        }

        public string ShortDescription
        {
            get
            {
                return string.Format("{0} ({1})", Name, base.FirstID);
            }
        }

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

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

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

namespace SwinAdventure
{
    public interface IHaveInventory
    {
        public string Name { get; }

        public GameObject? Locate(string id);
    }
}

```

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

namespace SwinAdventure
{
    public class IdentifiableObject
    {
        private List<string> _identifiers;

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

        public IdentifiableObject(string[] idents)
        {
            _identifiers = new List<string>();
            for (int i = 0; i < idents.Length; i++)
            {
                _identifiers.Add(idents[i].ToLower());
            }
        }

        public bool AreYou(string id)
        {
            bool result = false;

            foreach (string ident in _identifiers)
            {
                if (ident == id.ToLower())
                {
                    result = true;
                    break;
                }
            }

            return result;
        }

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

```

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

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

        public string ItemList
        {
            get
            {
                string itemList = "";
                foreach (Item item in _items)
                {
                    itemList += (string.Format("\n  {0}", 
                  item.ShortDescription));
                }

                return itemList;
            }
        }

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

        public bool HasItem(string id)
        {
            return Fetch(id) != null;
        }

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

    }
}

```

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

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

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

```

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

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

        public override string FullDescription
        {
            get
            {
                return string.Format("In the {0} you can see: {1}",
                  Name, Inventory.ItemList);
            }
        }

        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

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

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

```csharp
using SwinAdventure;

namespace TestBag
{
    public class Tests
    {
        [Test]
        public void TestBagLocatesItems()
        {
            Bag testBag = new Bag(new string[] { "bag", "testingBag"}, "test bag", "this is the test bag's description");
            Item shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "shovel description");
            testBag.Inventory.Put(shovel);

            GameObject? testLocateShovel = testBag.Locate("shovel");
            GameObject? testShovelRemainsInBag = testBag.Locate("shovel");
            Assert.That(testLocateShovel, Is.EqualTo(shovel));
            Assert.That(testShovelRemainsInBag, Is.EqualTo(shovel));
        }

        [Test]
        public void TestBagLocatesItself()
        {
            Bag testBag = new Bag(new string[] { "bag", "testingBag" }, "test bag", "this is the test bag's description");

            GameObject? testLocateBagID1 = testBag.Locate("bag");
            GameObject? testLocateBagID2 = testBag.Locate("testingBag");
            Assert.That(testLocateBagID1, Is.EqualTo(testBag));
            Assert.That(testLocateBagID2, Is.EqualTo(testBag));
        }

        [Test]
        public void TestBagLocatesNothing()
        {
            Bag testBag = new Bag(new string[] { "bag", "testingBag" }, "test bag", "this is the test bag's description");

            GameObject? testLocateShovel = testBag.Locate("shovel");
            Assert.That(testLocateShovel, Is.EqualTo(null));
        }

        [Test]
        public void TestBagFullDescription()
        {
            Bag testBag = new Bag(new string[] { "bag", "testingBag" }, "test bag", "this is the test bag's description");
            Item shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "shovel description");
            Item bronzeSword = new Item(new string[] { "sword", "bronze sword" }, "a bronze sword", "bronze sword description");
            testBag.Inventory.Put(shovel);
            testBag.Inventory.Put(bronzeSword);

            string testBagFullDescription = testBag.FullDescription;
            Assert.That(testBagFullDescription, Is.EqualTo("In the test bag you can see: \n  a shovel (shovel)\n  a bronze sword (sword)"));
        }

        [Test]
        public void TestBagInBag()
        {
            Bag b1 = new Bag(new string[] { "bag", "testingBag1" }, "test bag 1", "this is test bag 1's description");
            Bag b2 = new Bag(new string[] { "bag", "testingBag2" }, "test bag 2", "this is test bag 2's description");
            Item shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "shovel description");
            Item bronzeSword = new Item(new string[] { "sword", "bronze sword" }, "a bronze sword", "bronze sword description");
            b1.Inventory.Put(shovel);
            b2.Inventory.Put(bronzeSword);
            b1.Inventory.Put(b2);

            GameObject? testB1LocatesB2 = b1.Locate("testingBag2");
            GameObject? testB1LocatesShovel = b1.Locate("shovel");
            GameObject? testB1LocatesBronzeSword = b1.Locate("sword");
            Assert.That(testB1LocatesB2, Is.EqualTo(b2));
            Assert.That(testB1LocatesShovel, Is.EqualTo(shovel));
            Assert.That(testB1LocatesBronzeSword, Is.EqualTo(null));
        }
    }
}
```

```
using SwinAdventure;

namespace TestInventory
{
    public class Tests
    {

        [Test]
        public void TestFindItem()
        {
            Item shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "shovel description");
            Item bronzeSword = new Item(new string[] { "sword", "bronze sword" }, "a bronze sword", "bronze sword description");
            Inventory testInventory = new Inventory();
            testInventory.Put(shovel);
            testInventory.Put(bronzeSword);

            bool testShovel = testInventory.HasItem("shovel");
            bool testBronzeSword = testInventory.HasItem("sword");
            Assert.That(testShovel, Is.EqualTo(true));
            Assert.That(testBronzeSword, Is.EqualTo(true));
        }

        [Test]
        public void TestNoItemFind()
        {
            Item shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "shovel description");
            Item bronzeSword = new Item(new string[] { "sword", "bronze sword" }, "a bronze sword", "bronze sword description");
            Inventory testInventory = new Inventory();
            testInventory.Put(shovel);
            testInventory.Put(bronzeSword);

            bool testSmallComputer = testInventory.HasItem("pc");
            Assert.That(testSmallComputer, Is.EqualTo(false));
        }

        [Test]
        public void TestFetchItem()
        {
            Item shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "shovel description");
            Item bronzeSword = new Item(new string[] { "sword", "bronze sword" }, "a bronze sword", "bronze sword description");
            Inventory testInventory = new Inventory();
            testInventory.Put(shovel);
            testInventory.Put(bronzeSword);

            Item? testShovel = testInventory.Fetch("shovel");
            Item? testBronzeSword = testInventory.Fetch("sword");
            Assert.That(testShovel, Is.EqualTo(shovel));
            Assert.That(testBronzeSword, Is.EqualTo(bronzeSword));
        }

        [Test]
        public void TestTakeItem()
        {
            Item shovel = new Item(new string[] { "shovel", "spade" },
              "a shovel", "shovel description");
            Item bronzeSword = new Item(new string[] { "sword", "bronze
              sword" }, "a bronze sword", "bronze sword description");
            Inventory testInventory = new Inventory();
            testInventory.Put(shovel);
            testInventory.Put(bronzeSword);

            Item? testFetchShovel = testInventory.Take("shovel");
            bool testShovelInInventory = testInventory.HasItem
              ("shovel");
            Assert.That(testFetchShovel, Is.EqualTo(shovel));
            Assert.That(testShovelInInventory, Is.EqualTo(false));
        }

        [Test]
        public void TestItemList()
        {
            Item shovel = new Item(new string[] { "shovel", "spade" },
              "a shovel", "shovel description");
            Item bronzeSword = new Item(new string[] { "sword", "bronze
              sword" }, "a bronze sword", "bronze sword description");
            Inventory testInventory = new Inventory();
            testInventory.Put(shovel);
            testInventory.Put(bronzeSword);

            string testInventoryList = testInventory.ItemList;
            Assert.That(testInventoryList, Is.EqualTo("\n  a shovel
              (shovel)\n  a bronze sword (sword)"));
        }
    }
}
```

```
using SwinAdventure;

namespace TestIdentifiableObject
{
    public class Tests
    {

        [Test]
        public void TestAreYou()
        {
            IdentifiableObject myIdents = new IdentifiableObject(new 
              string[] { "fred", "bob" });

            bool fred = myIdents.AreYou("fred");
            Assert.That(fred, Is.EqualTo(true));
            bool bob = myIdents.AreYou("bob");
            Assert.That(bob, Is.EqualTo(true));
        }

        [Test]

        public void TestNotAreYou()
        {
            IdentifiableObject myIdents = new IdentifiableObject(new 
              string[] { "fred", "bob" });

            bool wilma = myIdents.AreYou("wilma");
            Assert.That(wilma, Is.EqualTo(false));
            bool boby = myIdents.AreYou("boby");
            Assert.That(boby, Is.EqualTo(false));
        }

        [Test]

        public void TestCaseSensitive()
        {
            IdentifiableObject myIdents = new IdentifiableObject(new 
              string[] { "fred", "bob" });

            bool fred = myIdents.AreYou("FRED");
            Assert.That(fred, Is.EqualTo(true));
            bool bob = myIdents.AreYou("bOB");
            Assert.That(bob, Is.EqualTo(true));
        }

        [Test]

        public void TestFirstID()
        {
            IdentifiableObject myIdents = new IdentifiableObject(new 
              string[] { "fred", "bob" });

            string firstID = myIdents.FirstID;
            Assert.That(firstID, Is.EqualTo("fred"));
        }

        [Test]

        public void TestFirstIDNoIDs()
        {
            IdentifiableObject myIdents = new IdentifiableObject(new
              string[] {});

            string firstID = myIdents.FirstID;
            Assert.That(firstID, Is.EqualTo(""));
        }

        [Test]

        public void TestAddIDs()
        {
            IdentifiableObject myIdents = new IdentifiableObject(new
              string[] { "fred", "bob" });
            myIdents.AddIdentifier("wilma");

            bool fred = myIdents.AreYou("fred");
            Assert.That(fred, Is.EqualTo(true));
            bool bob = myIdents.AreYou("bob");
            Assert.That(bob, Is.EqualTo(true));
            bool wilma = myIdents.AreYou("wilma");
            Assert.That(wilma, Is.EqualTo(true));
        }
    }
}
```

```csharp
using SwinAdventure;
using System.Reflection.Metadata;

namespace TestItem
{
    public class Tests
    {
        [Test]
        public void TestItemIsIdentifiable()
        {
            // testing identifiers of Item object
            Item bronzeSword = new Item(new string[] { "sword", "bronze
              sword" }, "a bronze sword", "bronze sword description");
            bool testBronzeSwordID1 = bronzeSword.AreYou("sword");
            bool testBronzeSwordID2 = bronzeSword.AreYou("bronze
              sword");
            Assert.That(testBronzeSwordID1, Is.EqualTo(true));
            Assert.That(testBronzeSwordID2, Is.EqualTo(true));
        }

        [Test]

        public void TestItemShortDescription()
        {
            Item bronzeSword = new Item(new string[] { "sword", "bronze
              sword" }, "a bronze sword", "bronze sword description");
            string testBronzeSword = bronzeSword.ShortDescription;
            Assert.That(testBronzeSword, Is.EqualTo("a bronze sword
              (sword)"));
        }

        [Test]

        public void TestItemFullDescription()
        {
            Item bronzeSword = new Item(new string[] { "sword", "bronze
              sword" }, "a bronze sword", "bronze sword description");
            string testBronzeSword = bronzeSword.FullDescription;
            Assert.That(testBronzeSword, Is.EqualTo("bronze sword
              description"));
        }
    }
}
```

```
using SwinAdventure;

namespace TestPlayer
{
    public class Tests
    {
        private Location location;
        private Item ruby;
        private Player p;
        private Item shovel;
        private Item bronzeSword;

        [SetUp]
        public void Setup()
        {
            location = new Location(new string[] { "location" }, "the Location", "This is a test location");
            ruby = new Item(new string[] { "gem", "ruby" }, "a red gem", "A brilliant ruby, glows with a fiery red hue");
            p = new Player("Tester", "the mighty test player");
            shovel = new Item(new string[] { "shovel", "spade" }, "a shovel", "shovel description");
            bronzeSword = new Item(new string[] { "sword", "bronze sword" }, "a bronze sword", "bronze sword description");
        }


        [Test]
        public void TestPlayerIsIdentifiable()
        {
            bool testPMe = p.AreYou("me");
            bool testPInventory = p.AreYou("inventory");
            Assert.That(testPMe, Is.EqualTo(true));
            Assert.That(testPInventory, Is.EqualTo(true));
        }

        [Test]
        public void TestPlayerLocatesItems()
        {
            p.Inventory.Put(shovel);
            p.Inventory.Put(bronzeSword);

            GameObject? testLocateShovel = p.Locate("shovel");
            GameObject? testLocateBronzeSword = p.Locate("sword");
            Assert.That(testLocateShovel, Is.EqualTo(shovel));
            Assert.That(testLocateBronzeSword, Is.EqualTo(bronzeSword));
        }

        [Test]
        public void TestPlayerLocatesItself()
        {
            GameObject? testLocatePMe = p.Locate("me");
            GameObject? testLocatePInventory = p.Locate("inventory");
```

```csharp
            Assert.That(testLocatePMe, Is.EqualTo(p));
            Assert.That(testLocatePInventory, Is.EqualTo(p));
        }

        [Test]
        public void TestPlayerLocatesNothing()
        {
            p.Inventory.Put(shovel);
            p.Inventory.Put(bronzeSword);

            GameObject? testLocateNothing = p.Locate("nothing");
            Assert.That(testLocateNothing, Is.EqualTo(null));
        }

        [Test]
        public void TestPlayerFullDescription()
        {
            p.Inventory.Put(shovel);
            p.Inventory.Put(bronzeSword);

            string testFullDescription = p.FullDescription;
            Assert.That(testFullDescription, Is.EqualTo("You are Tester, the mighty test player.\nYou are carrying: \n  a shovel (shovel)\n  a bronze sword (sword)"));
        }

        [Test]
        public void TestPlayerLocatesItemsInLocation()
        {
            location.Inventory.Put(ruby);
            p.Location = location;

            GameObject? testPlayerLocatesRubyInLocation = p.Locate("ruby");
            Assert.That(testPlayerLocatesRubyInLocation, Is.EqualTo(ruby));
        }

        [Test]
        public void TestPlayerLocatesNoItemsInLocation()
        {
            p.Location = location;

            GameObject? testPlayerLocatesNothingInLocation = p.Locate("nothing");
            Assert.That(testPlayerLocatesNothingInLocation, Is.EqualTo(null));
        }
    }
}
```

```
using SwinAdventure;


namespace TestLookCommand
{
    public class Tests
    {
        private LookCommand look;
        private Player testPlayer;
        private Item gem;
        private Bag bag;
        private Location location;

        [SetUp]
        public void Setup()
        {
            look = new LookCommand();
            testPlayer = new Player("testPlayer", "test player description");
            gem = new Item(new string[] { "gem" }, "a gem", "gem's description");
            bag = new Bag(new string[] { "bag" }, "a bag", "bag's description");
            location = new Location(new string[] { "location" }, "the Location", "This is a test location");
            testPlayer.Location = location;
        }

        [Test]
        public void TestLookAtMe()
        {
            string testLookAtInventory = look.Execute(testPlayer, new string[] { "look", "at", "inventory" });
            Assert.That(testLookAtInventory, Is.EqualTo("You are testPlayer, test player description.\nYou are carrying: "));
        }

        [Test]
        public void TestLookAtGem()
        {
            testPlayer.Inventory.Put(gem);
            string testLookAtGem = look.Execute(testPlayer, new string[] { "look", "at", "gem" });
            Assert.That(testLookAtGem, Is.EqualTo("gem's description"));
        }

        [Test]
        public void TestLookAtUnknown()
        {
            string testLookAtUnknown = look.Execute(testPlayer, new string[] { "look", "at", "gem" });
            Assert.That(testLookAtUnknown, Is.EqualTo("I cannot find the gem"));
        }

        [Test]
        public void TestLookAtGemInMe()
        {
            testPlayer.Inventory.Put(gem);
            string testLookAtGem = look.Execute(testPlayer, new string[] { "look", "at", "gem", "in", "inventory" });
            Assert.That(testLookAtGem, Is.EqualTo("gem's description"));
        }

        [Test]
        public void TestLookAtGemInBag()
        {
            bag.Inventory.Put(gem);
            testPlayer.Inventory.Put(bag);
            string testLookAtGemInBag = look.Execute(testPlayer, new string[] { "look", "at", "gem", "in", "bag" });
            Assert.That(testLookAtGemInBag, Is.EqualTo("gem's description"));
        }

        [Test]
        public void TestLookAtGemInNoBag()
        {
            string testLookAtGemInNoBag = look.Execute(testPlayer, new string[] { "look", "at", "gem", "in", "bag" });
            Assert.That(testLookAtGemInNoBag, Is.EqualTo("I cannot find the bag"));
        }

        [Test]
        public void TestLookAtNoGemInBag()
        {
            testPlayer.Inventory.Put(bag);
            string testLookAtNoGemInBag = look.Execute(testPlayer, new string[] { "look", "at", "gem", "in", "bag" });
            Assert.That(testLookAtNoGemInBag, Is.EqualTo("I cannot find the gem in the bag"));
        }

        [Test]
        public void TestInvalidLook()
        {
            string testIncorrectTextLength = look.Execute(testPlayer, new string[] { "testing", "incorrect", "text", "length" });
            string testLookNotFirstWord = look.Execute(testPlayer, new string[] { "testing", "look", "is", "not", "first" });
            string testAtNotSecondWord = look.Execute(testPlayer, new
                string[] { "look", "test", "at", "not", "second" });
            string testInNotFourthWord = look.Execute(testPlayer, new string[] { "look", "at", "in", "not", "fourth" });
            Assert.That(testIncorrectTextLength, Is.EqualTo("I don't know how to look like that"));
            Assert.That(testLookNotFirstWord, Is.EqualTo("Error in look input"));
            Assert.That(testAtNotSecondWord, Is.EqualTo("What do you want to look at?"));
            Assert.That(testInNotFourthWord, Is.EqualTo("What do you want to look in?"));
        }

        [Test]
        public void TestLook()
        {
            location.Inventory.Put(gem);
            string testLook = look.Execute(testPlayer, new string[] { "look" });
            Assert.That(testLook, Is.EqualTo("You are in the Location\nThis is a test location\nIn this room you can see:\n  a gem (gem)"));
        }

        [Test]
        public void TestLookAtLocation()
        {
            location.Inventory.Put(gem);
            string testLookAtLocation = look.Execute(testPlayer, new string[] { "look", "at", "location" });
            Assert.That(testLookAtLocation, Is.EqualTo("You are in the Location\nThis is a test location\nIn this room you can see:\n  a gem (gem)"));
        }

        [Test]
        public void TestLookAtGemInLocation()
        {
            location.Inventory.Put(gem);
            string testLookAtGem = look.Execute(testPlayer, new string[] { "look", "at", "gem" });
            string testLookAtGemInLocation = look.Execute(testPlayer, new string[] { "look", "at", "gem", "in", "location" });
            Assert.That(testLookAtGem, Is.EqualTo("gem's description"));
            Assert.That(testLookAtGemInLocation, Is.EqualTo("gem's description"));
        }

        [Test]
        public void TestLookAtNoGemInLocation()
        {
            string testLookAtGemInLocation = look.Execute(testPlayer,
                new string[] { "look", "at", "gem", "in", "location" });
            Assert.That(testLookAtGemInLocation, Is.EqualTo("I cannot find the gem in the location"));
        }

        [Test]
        public void TestLookAtGemInBagInLocation()
        {
            bag.Inventory.Put(gem);
            location.Inventory.Put(bag);
            string testLookAtGemInBag = look.Execute(testPlayer, new string[] { "look", "at", "gem", "in", "bag" });
            Assert.That(testLookAtGemInBag, Is.EqualTo("gem's description"));
        }
    }
}
```