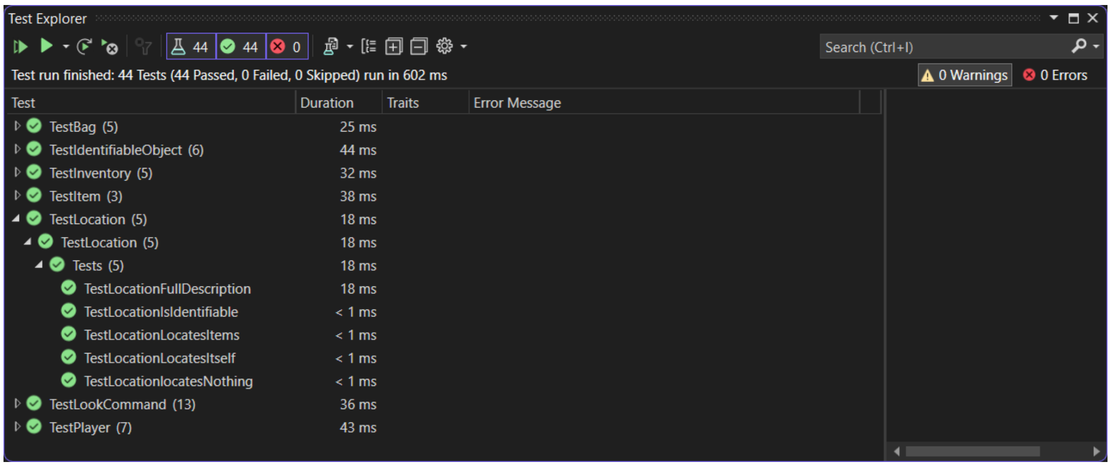
7.2C - Case Study - Iteration 6 – Locations

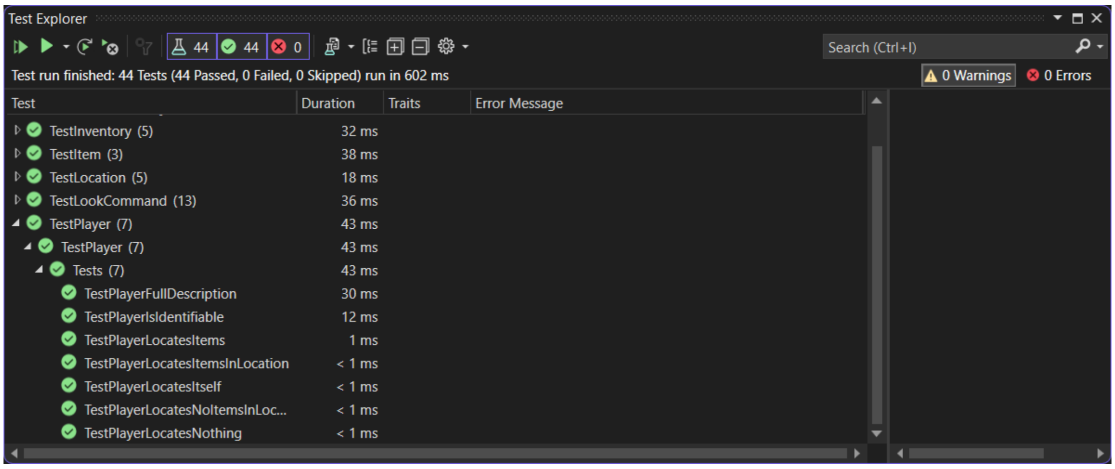
Jayden Kong, 104547242

Location tests passing:



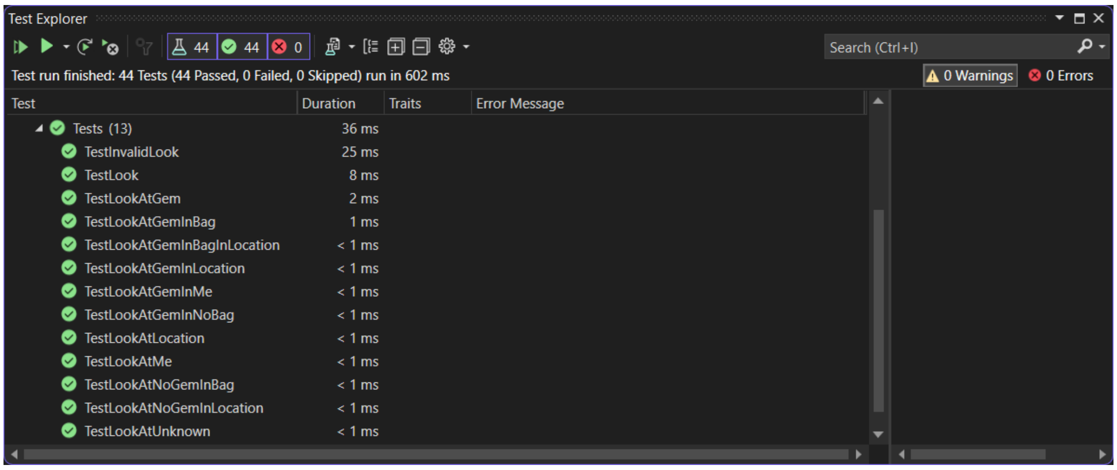
```
Test Explorer
44  44  0                                                      Search (Ctrl+I)
Test run finished: 44 Tests (44 Passed, 0 Failed, 0 Skipped) run in 602 ms     0 Warnings   0 Errors
Test                                Duration   Traits   Error Message
▷ TestBag (5)                          25 ms
▷ TestIdentifiableObject (6)           44 ms
▷ TestInventory (5)                    32 ms
▷ TestItem (3)                         38 ms
◢ TestLocation (5)                     18 ms
  ◢ TestLocation (5)                   18 ms
    ◢ Tests (5)                        18 ms
        TestLocationFullDescription    18 ms
        TestLocationIsIdentifiable    < 1 ms
        TestLocationLocatesItems      < 1 ms
        TestLocationLocatesItself     < 1 ms
        TestLocationlocatesNothing    < 1 ms
▷ TestLookCommand (13)                 36 ms
▷ TestPlayer (7)                       43 ms
```

Player tests passing:



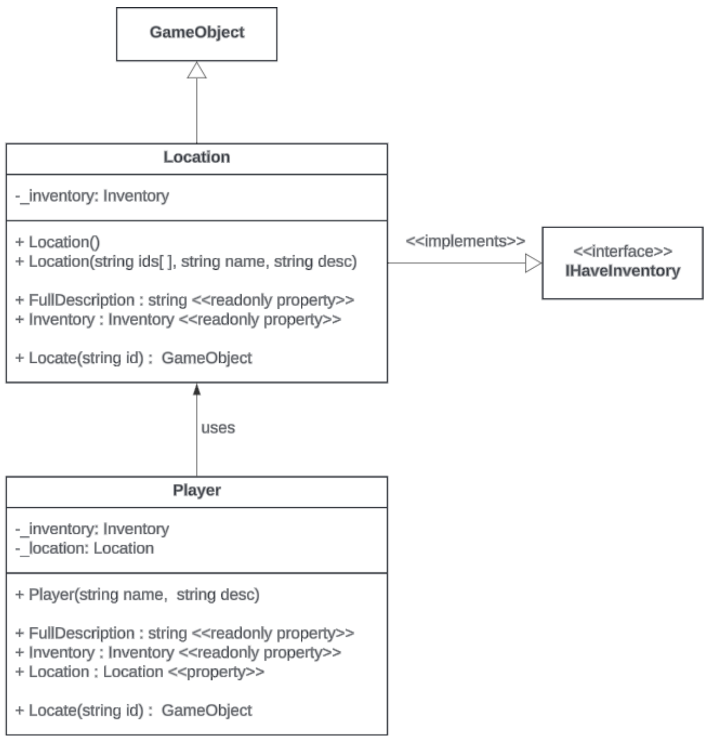
```
Test Explorer
44  44  0                                                      Search (Ctrl+I)
Test run finished: 44 Tests (44 Passed, 0 Failed, 0 Skipped) run in 602 ms     0 Warnings   0 Errors
Test                                     Duration   Traits   Error Message
▷ TestInventory (5)                         32 ms
▷ TestItem (3)                              38 ms
▷ TestLocation (5)                          18 ms
▷ TestLookCommand (13)                      36 ms
◢ TestPlayer (7)                            43 ms
  ◢ TestPlayer (7)                          43 ms
    ◢ Tests (7)                             43 ms
        TestPlayerFullDescription           30 ms
        TestPlayerIsIdentifiable            12 ms
        TestPlayerLocatesItems               1 ms
        TestPlayerLocatesItemsInLocation   < 1 ms
        TestPlayerLocatesItself            < 1 ms
        TestPlayerLocatesNoItemsInLoc...   < 1 ms
        TestPlayerLocatesNothing           < 1 ms
```

Look command tests passing:

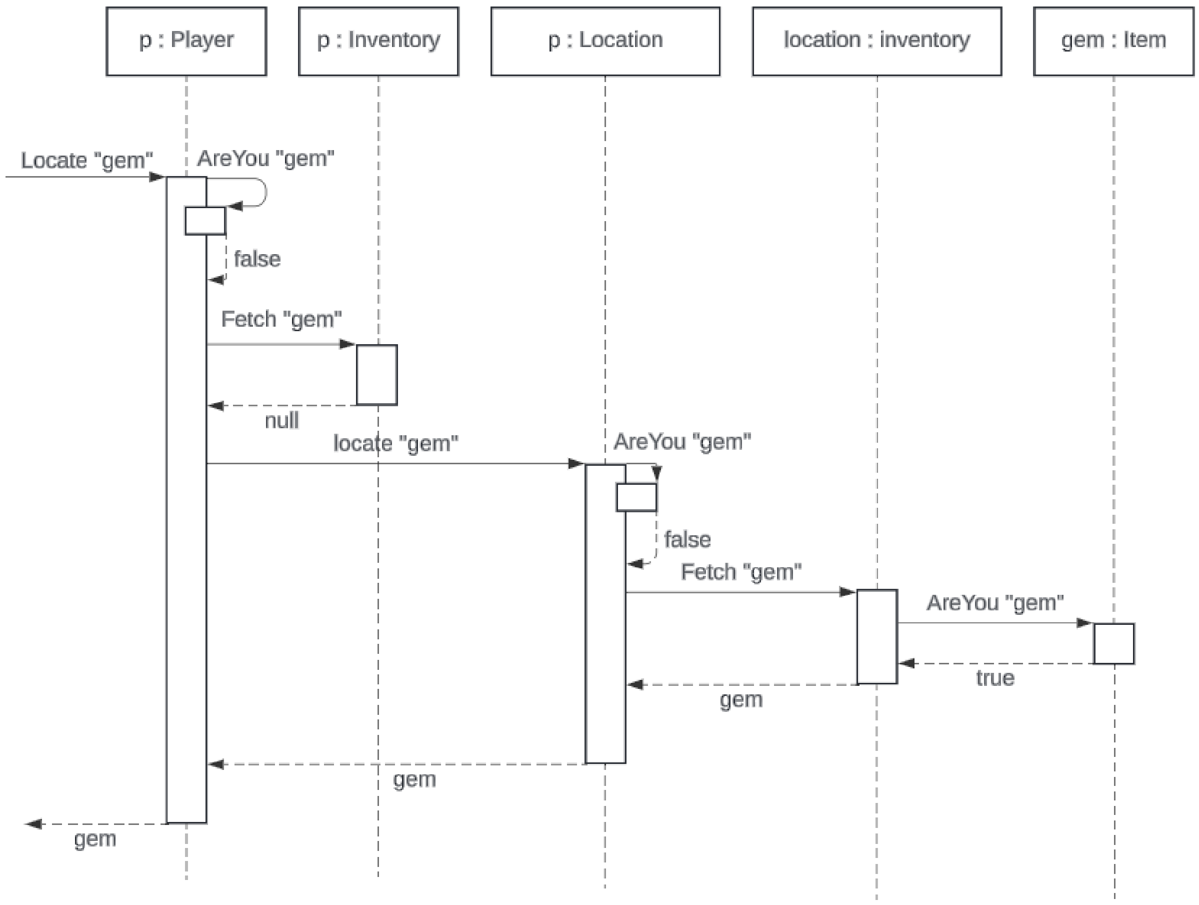


```
Test Explorer
44  44  0                                                      Search (Ctrl+I)
Test run finished: 44 Tests (44 Passed, 0 Failed, 0 Skipped) run in 602 ms     0 Warnings   0 Errors
Test                                     Duration   Traits   Error Message
    ◢ Tests (13)                            36 ms
        TestInvalidLook                     25 ms
        TestLook                             8 ms
        TestLookAtGem                        2 ms
        TestLookAtGemInBag                   1 ms
        TestLookAtGemInBagInLocation       < 1 ms
        TestLookAtGemInLocation            < 1 ms
        TestLookAtGemInMe                  < 1 ms
        TestLookAtGemInNoBag               < 1 ms
        TestLookAtLocation                 < 1 ms
        TestLookAtMe                       < 1 ms
        TestLookAtNoGemInBag               < 1 ms
        TestLookAtNoGemInLocation          < 1 ms
        TestLookAtUnknown                  < 1 ms
```

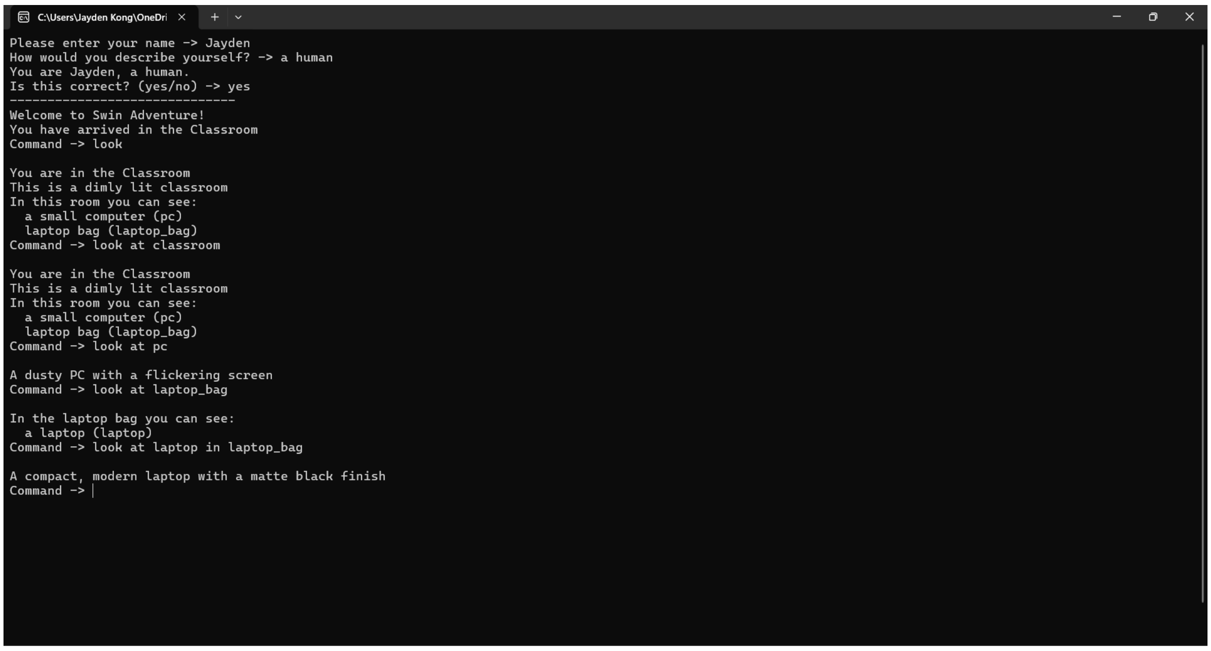
UML Diagram:


GameObject
Location
-_inventory: Inventory
+ Location()
+ Location(string ids[ ], string name, string desc)
+ FullDescription : string <<readonly property>>
+ Inventory : Inventory <<readonly property>>
+ Locate(string id) : GameObject
<<implements>>
<<interface>>
IHaveInventory
uses
Player
-_inventory: Inventory
-_location: Location
+ Player(string name, string desc)
+ FullDescription : string <<readonly property>>
+ Inventory : Inventory <<readonly property>>
+ Location : Location <<property>>
+ Locate(string id) : GameObject

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
In this room you can see:
  a small computer (pc)
  laptop bag (laptop_bag)
Command -> look at classroom

You are in the Classroom
This is a dimly lit classroom
In this room you can see:
  a small computer (pc)
  laptop bag (laptop_bag)
Command -> look at pc

A dusty PC with a flickering screen
Command -> look at laptop_bag

In the laptop bag you can see:
  a laptop (laptop)
Command -> look at laptop in laptop_bag

A compact, modern laptop with a matte black finish
Command ->
```

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

            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");
            Bag bag = new Bag(new string[] { "bag" }, "leather bag", "Crafted from supple brown leather, this small bag is perfect for carrying items");
            Item computer = new Item(new string[] { "pc" }, "a small computer", "A dusty PC with a flickering screen");
            Bag laptopBag = new Bag(new string[] { "laptop_bag", "bag" }, "laptop bag", "A sleek, black laptop bag. Its fabric is slightly worn from use");
            Item laptop = new Item(new string[] { "laptop" }, "a laptop", "A compact, modern laptop with a matte black finish");
            Location classroom = new Location(new string[] { "classroom", "location" }, "the Classroom", "This is a dimly lit classroom");        // Player will initially be in the classroom

            player.Location = classroom;
            laptopBag.Inventory.Put(laptop);
            player.Location.Inventory.Put(computer);
            player.Location.Inventory.Put(laptopBag);
            bag.Inventory.Put(ruby);
            player.Inventory.Put(shovel);
            player.Inventory.Put(bronzeSword);
            player.Inventory.Put(bag);

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

            bool gameLoop = true;
            while (gameLoop)
            {
                Console.Write("Command -> ");
                string? playerInput = Console.ReadLine();
                string[] inputToPass = playerInput.Split(new char[] {  }, StringSplitOptions.RemoveEmptyEntries);
                Console.WriteLine("");
                Console.WriteLine(lookCommand.Execute(player, inputToPass));
            }
        }
    }
}

```

```
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;

        public override string FullDescription
        {
            get
            {
                return string.Format("You are in {0}\n{1}\nIn this room you can see:{2}", Name, base.FullDescription, Inventory.ItemList);
            }
        }
        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

        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)
        {
            _inventory = new Inventory();
        }

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

```csharp
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"}, 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.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;
        }
    }
}
```

```
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);
        }
    }
}

```

```
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 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.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 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());
        }
    }
}

```

```
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;
        }

    }
}

```

```
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) { }
    }
}

```

```
using SwinAdventure;

namespace TestPlayer
{
    public class Tests
    {
        Location location;
        Item ruby;
        Player p;
        Item shovel;
        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");
            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 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;

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 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;
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"));
        }
    }
}
```

```csharp
using SwinAdventure;

namespace TestLocation
{
    public class Tests
    {
        Location location;
        Item ruby;

        [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");
            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()
        {
            string testLocationFullDescription = location.FullDescription;
            Assert.That(testLocationFullDescription, Is.EqualTo("You are in the Location\nThis is a test location\nIn this room you can see:\n  a red gem (gem)"));
        }
    }
}
```

```csharp
using SwinAdventure;


namespace TestLookCommand
{
    public class Tests
    {
        private LookCommand look;
        private Player testPlayer;
        private Item gem;
        private Bag bag;
        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"));
        }
    }
}
```