Command.cs

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

namespace Iteration4
{
    public abstract class Command(string[] ids) : IdentifiableObject(ids)
    {
        public abstract string Execute(Player p, string[] text);
    }
}
```

Items.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Iteration4
{
    public class Item(string[] idents, string name, string desc) :
GameObject(idents, name, desc)
    {

    }
}
```

GameObject.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Iteration4
{
    public class GameObject(string[] idents, string name, string desc) :
IdentifiableObject(idents)
    {
        private readonly string _description = desc;
        private readonly string _name = name;

        public string Name
        {
            get
            {
                return _name;
            }
        }
        public string ShortDescription
        {
            get
            {
                return "a " + _name + " " + FirstID;
            }
        }
        public virtual string FullDescription
        {
            get
            {
```

```
                return _description;
            }
        }
    }
}
```

Players.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace Iteration4
{
    public class Player(string name, string desc) : GameObject(idents, name,
desc), IHaveInventory
    {
        private readonly Inventory _inventory = new();
        public GameObject Locate(string id)
        {
            if (AreYou(id))
                return this;
            return _inventory.Fetch(id);
        }

        public override string FullDescription
        {
            get
            {
                return "You are " + Name + ", " + base.FullDescription + ".\nYou
are carrying:\n" + _inventory.ItemList;
            }
        }

        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

        private static readonly string[] idents = ["me", "inventory"];
    }
}
```

Inventory.cs

```
using System;
using System.Collections.Generic;

namespace Iteration4
{
    public class Inventory
    {
        private readonly List<Item> _items;

        public Inventory()
        {
            _items = [];
```

```
        }

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

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

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

        public Item Take(string id)
        {
            Item takeitem = Fetch(id);
            _items.Remove(takeitem);
            return takeitem;
        }

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

IdentifiableObject.cs

```
using System;
using System.Collections.Generic;

namespace Iteration4
{
    public class IdentifiableObject
    {
        private readonly List<string> _idents = [];
```

```
        public IdentifiableObject(string[] idents)
        {
            foreach (string s in idents)
            {
                AddIdentifier(s);
            }
        }

        public bool AreYou(string id)
        {
            return _idents.Contains(id.ToLower());
        }

        public string FirstID
        {
            get
            {
                if (_idents.Count == 0)
                {
                    return "";
                }
                else
                {
                    return _idents[0];
                }
            }
        }

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

IhaveInventory.cs

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

namespace Iteration4
{
    interface IHaveInventory
    {
        GameObject Locate(string id);
        string Name
        {
            get;
        }
    }
}
```

Look.cs

```
using System;
```

```
namespace Iteration4
{
    public class Look : Command
    {
        public Look() : base(["look"]) { }

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

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

            if (!text[1].Equals("at", StringComparison.CurrentCultureIgnoreCase))
            {
                return "What do you want to look at?";
            }

            IHaveInventory container = p;
            if (text.Length == 5)
            {
                if (!text[3].Equals("in",
StringComparison.CurrentCultureIgnoreCase))
                    return "What do you want to look in?";

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

                return LookAtIn(text[2], container);

        }

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

        private static string LookAtIn(string thingId, IHaveInventory container)
        {
            GameObject foundItem = container.Locate(thingId);
            if (foundItem == null)
            {
                if (container == container.Locate("inventory"))
                {
                    return $"I can't find the {thingId}";
                }
                else
                {
                    return $"I can't find the {thingId} in the {container.Name}";
                }
            }
            return foundItem.FullDescription;
        }
    }
}
```

Bag.cs

```
using System;
using System.Xml.Linq;

namespace Iteration4
{
    public class Bag(string[] idents, string name, string desc) : Item(idents,
name, desc), IHaveInventory
    {
        private readonly Inventory _inventory = new();

        public Inventory Inventory
        {
            get
            {
                return _inventory;
            }
        }

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

        }

        public override string FullDescription
        {
            get
            {
                string InventoryDescription = "In the " + Name + " you can
see:\n";
                InventoryDescription += _inventory.ItemList;
                return InventoryDescription;
            }
        }
    }
}
```

LookTest.cs

```
using System;
using System.ComponentModel;
using System.Linq;

namespace Iteration4
{
    [TestFixture]
    public class TestLook
    {
        Look look;
        Player player;
        Bag bag;
        Item sword;
```

```
        Item shield;
        Item potion;

        [SetUp]
        public void SetUp()
        {
            look = new Look();
            player = new Player("ruchan", "a member of a chess club");

            bag = new Bag(["bag"], "leather bag", "a light bag, suitable for
short trips");
            sword = new Item(["sword"], "diamond", "a diamond sword which has not
broken once");
            shield = new Item(["shield"], "gold", "a gold shield that lasts a
lifetime");
            potion = new Item(["potion"], "healing", "a healing potion which is
needed for the adventurers");
        }

        [Test]
        public void TestLookAtMe()
        {
            Assert.That(look.Execute(player, ["look", "at", "me"]),
Is.EqualTo(player.FullDescription));
        }

        [Test]
        public void TestLookAtSword()
        {
            player.Inventory.Put(sword);
            Assert.That(look.Execute(player, ["look", "at", "sword"]),
Is.EqualTo(sword.FullDescription));
        }

        [Test]
        public void TestLookAtUnknownItems()
        {
            Assert.That(look.Execute(player, ["look", "at", "plate"]),
Is.EqualTo($"I can't find the plate"));
        }

        [Test]
        public void TestLookAtSwordInMe()
        {
            player.Inventory.Put(sword);
            Assert.That(look.Execute(player, ["look", "at", "sword", "in",
"me"]), Is.EqualTo(sword.FullDescription));
        }
        [Test]
        public void TestLookAtSwordInBag()
        {
            bag.Inventory.Put(sword);
            bag.Inventory.Put(shield);
            player.Inventory.Put(bag);
            Assert.That(look.Execute(player, ["look", "at", "sword", "in",
"bag"]), Is.EqualTo(sword.FullDescription));
        }

        [Test]
        public void TestLookAtPotionInNoBag()
        {
            bag.Inventory.Put(potion);
```

```
            Assert.That(look.Execute(player, ["look", "at", "potion", "in",
"bag"]), Is.EqualTo("I can't find the bag"));
        }

        [Test]
        public void TestLookAtNoShieldInBag()
        {
            bag.Inventory.Put(sword);
            player.Inventory.Put(bag);
            Assert.Multiple(() =>
            {
                Assert.That(look.Execute(player, ["look", "at", "shield", "in",
"bag"]), Is.EqualTo("I can't find the shield in the leather bag"));
                Assert.That(look.Execute(player, ["look", "at", "potion", "in",
"bag"]), Is.EqualTo("I can't find the potion in the leather bag"));
            });
        }
        [Test]
        public void TestInvalidLook()
        {
            Assert.Multiple(() =>
            {
                Assert.That(look.Execute(player, ["look", "down"]), Is.EqualTo("I
don't know how to look like that"));
                Assert.That(look.Execute(player, ["stare", "at", "plate"]),
Is.EqualTo("Error in look input"));
                Assert.That(look.Execute(player, ["look", "at", "potion", "on",
"bag"]), Is.EqualTo("What do you want to look in?"));
                Assert.That(look.Execute(player, ["look", "for", "shield"]),
Is.EqualTo("What do you want to look at?"));
            });
        }
    }
}
```

PlayersTest.cs

```
using System;
using System.Collections.Generic;
using NUnit.Framework;

namespace Iteration4
{
    [TestFixture]
    public class TestPlayer
    {
        Player player;
        Item sword;
        Item shield;

        [SetUp]
        public void Setup()
        {
            player = new Player("ruchan", "a member of a chess club");
            sword = new Item(["sword"], "diamond", "a diamond sword which has not
broken once");
            shield = new Item(["shield"], "gold", "a gold shield that lasts a
lifetime");
```

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

        [Test]
        public void TestPLayerIsIdentifiable()
        {
            Assert.Multiple(() =>
            {
                Assert.That(player.AreYou("me"), Is.True, "True");
                Assert.That(player.AreYou("inventory"), Is.True, "True");
            });
        }

        [Test]
        public void TestPlayerLocatesItems()
        {
            var result = false;
            var itemLocated = player.Locate("sword");
            if (sword == itemLocated)
            {
                result = true;
            }
            Assert.That(result, Is.True);

            _ = player.Locate("shield");
            if (shield == itemLocated)
            {
                result = true;
            }
            Assert.That(result, Is.True);
        }

        [Test]
        public void TestPlayerLocatesItself()
        {
            Assert.Multiple(() =>
            {
                Assert.That(player.Locate("me"), Is.EqualTo(player));
                Assert.That(player.Locate("inventory"), Is.EqualTo(player));
            });
        }

        [Test]
        public void TestPlayerLocatesNothing()
        {
            Assert.That(player.Locate("plate"), Is.EqualTo(null));
        }

        [Test]
        public void TestPlayerFullDescription()
        {
            Assert.That(player.FullDescription, Is.EqualTo("You are ruchan, a
member of a chess club.\nYou are carrying:\n\ta diamond sword\n\ta gold
shield\n"));
        }
    }
}
```

ItemsTest.cs

```
using System;
using System.Collections.Generic;
```

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

namespace Iteration4
{
    [TestFixture]
    public class TestItem
    {
        Item shield;

        [SetUp]
        public void SetUp()
        {
            shield = new Item(["shield"], "gold", "a gold shield that lasts a
lifetime");
        }
        [Test]
        public void TestItemIdentifiable()
        {
            Assert.That(shield.AreYou("shield"), Is.True, "True");
            Assert.That(shield.AreYou("sword"), Is.False, "True");
        }

        [Test]
        public void TestShortDesc()
        {
            Assert.That(shield.ShortDescription, Is.EqualTo("a gold shield"));
        }

        [Test]
        public void TestFullDesc()
        {
            Assert.That(shield.FullDescription, Is.EqualTo("a gold shield that
lasts a lifetime"));

        }

    }
}
```

InventoryTest.cs

```
using System;
using System.Collections.Generic;
using NUnit.Framework;

namespace Iteration4
{
    [TestFixture]
    public class TestInventory
    {
        Inventory inventory;
        Item sword;
        Item shield;
        Item potion;

        [SetUp]
        public void SetUp()
        {
            inventory = new Inventory();
```

```
            sword = new Item(["sword"], "diamond", "a diamond sword which has not
broken once");
            shield = new Item(["shield"], "gold", "a gold shield that lasts a
lifetime");
            potion = new Item(["potion"], "healing", "a healing potion which is
needed for the adventurers");

            inventory.Put(sword);
            inventory.Put(shield);
        }
        [Test]
        public void TestFoundItem()
        {
            Assert.Multiple(() =>
            {
                Assert.That(inventory.HasItem("sword"), Is.True);
                Assert.That(inventory.HasItem("shield"), Is.True);
            });
        }
        [Test]
        public void TestNoItemFound()
        {
            Assert.That(inventory.HasItem("potion"), Is.False);
        }
        [Test]
        public void TestFecthItem()
        {
            Assert.Multiple(() =>
            {
                Assert.That(inventory.Fetch("sword"), Is.EqualTo(sword));
                Assert.That(inventory.HasItem("sword"), Is.True);
            });
        }
        [Test]
        public void TestTakeItem()
        {
            Assert.Multiple(() =>
            {
                Assert.That(inventory.Take("sword"), Is.EqualTo(sword));
                Assert.That(inventory.HasItem("sword"), Is.False);
                Assert.That(inventory.HasItem("shield"), Is.True);
                Assert.That(inventory.HasItem("potion"), Is.False);
            });
        }
        [Test]
        public void TestItemList()
        {
            Assert.That(inventory.ItemList, Is.EqualTo("\ta diamond sword\n\ta
gold shield\n"));
        }
    }
}
```

IdentifiableObjectTest.cs

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

```
namespace Iteration4
{
    [TestFixture]
    public class TestIdentifiableObject
    {

        [Test]
        public void TestAreYou()
        {
            string[] testArray = ["Fred", "Bob"];
            IdentifiableObject testIdentifiableObject = new(testArray);
            Assert.That(testIdentifiableObject.AreYou("fred"), Is.True);
        }

        [Test]
        public void TestNotAreYou()
        {
            string[] testArray = ["Fred", "Bob"];
            IdentifiableObject testIdentifiableObject = new(testArray);
            Assert.That(testIdentifiableObject.AreYou("wilma"), Is.False);
        }

        [Test]
        public void TestCaseSensitive()
        {
            string[] testArray = ["Fred", "Bob"];
            IdentifiableObject testIdentifiableObject = new(testArray);
            Assert.That(testIdentifiableObject.AreYou("bOB"), Is.True);
        }

        [Test]
        public void TestFirstID()
        {
            string[] testArray = ["Fred", "Bob"];
            IdentifiableObject testIdentifiableObject = new(testArray);
            StringAssert.AreEqualIgnoringCase("fred",
testIdentifiableObject.FirstID);
        }

        [Test]
        public void TestFirstIDWithNoIDs()
        {
            string[] testArray = [];
            IdentifiableObject testIdentifableObject = new(testArray);
            StringAssert.AreEqualIgnoringCase("", testIdentifableObject.FirstID);
        }

        [Test]
        public void TestAddID()
        {
            string[] testArray = ["Fred", "Bob"];
            IdentifiableObject testIdentifiableObject = new(testArray);
            testIdentifiableObject.AddIdentifier("Wilma");
            Assert.Multiple(() =>
            {
                Assert.That(testIdentifiableObject.AreYou("fred"), Is.True);
                Assert.That(testIdentifiableObject.AreYou("bob"), Is.True);
                Assert.That(testIdentifiableObject.AreYou("wilma"), Is.True);
            });
        }
    }
}
```

BagTest.cs

```
using System;
using System.Collections.Generic;
using NUnit.Framework;

namespace Iteration4
{
    [TestFixture]
    public class TestBag
    {
        Item sword;
        Item shield;
        Bag bag;
        Bag backpack;

        [SetUp]
        public void SetUp()
        {
            sword = new Item(["sword"], "diamond", "a diamond sword which has not
broken once");
            shield = new Item(["shield"], "gold", "a gold shield that lasts a
lifetime");
            bag = new Bag(["bag"], "leather bag", "a light bag, suitable for
short trips");
            backpack = new Bag(["backpack"], "fabric backpack", "a medium-sized
backpack, suitable for abroad travelling");

            bag.Inventory.Put(sword);
            backpack.Inventory.Put(shield);
            backpack.Inventory.Put(bag);
        }

        [Test]
        public void TestBagLocateItems()
        {
            Assert.Multiple(() =>
            {
                Assert.That(bag.Locate("sword"), Is.EqualTo(sword));
                Assert.That(backpack.Locate("shield"), Is.EqualTo(shield));
            });
        }
        [Test]
        public void TestBagLocatesItself()
        {
            Assert.Multiple(() =>
            {
                Assert.That(bag.Locate("bag"), Is.EqualTo(bag));
                Assert.That(backpack.Locate("backpack"), Is.EqualTo(backpack));
            });
        }
        [Test]
        public void TestBagLocatesNothing()
        {
            Assert.That(bag.Locate("Nothing"), Is.EqualTo(null));
        }
        [Test]
        public void TestBagFullDesc()
        {
            Assert.That(bag.FullDescription, Is.EqualTo("In the leather bag you
can see:\n\ta diamond sword\n"));
        }
        [Test]
```

```
        public void TestBagInBag()
        {
            Assert.Multiple(() =>
            {
                Assert.That(backpack.Locate("bag"), Is.EqualTo(bag));
                Assert.That(bag.Locate("sword"), Is.EqualTo(sword));
                Assert.That(bag.Locate("shield"), Is.EqualTo(null));
            });
        }
    }
}
```