Items.cs

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

namespace Iteration2
{
    public class Item(string[] idents, string name, string desc) :
GameObject(idents, name, desc)
    {

    }
}
```

Players.cs

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

namespace Iteration2
{
    public class Player(string name, string desc) : GameObject(idents, name,
desc)
    {
        private readonly Inventory _inventory = new();
        private static readonly string[] idents = ["me", "inventory"];

        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;
            }
        }
    }
}
```

Inventory.cs

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

namespace Iteration2
{
    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 takenitem = Fetch(id);
            _items.Remove(takenitem);
            return takenitem;
        }

        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 Iteration2
{
    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());
        }
    }
}
```

GameObject.cs

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

namespace Iteration2
{
```

```
    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;
            }
        }
    }
}
```

ItemsTest.cs

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

namespace Iteration2
{
    [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 TestItemIsIdentifiable()
        {
            Assert.Multiple(() =>
            {
                Assert.That(shield.AreYou("shield"), Is.True, "True");
                Assert.That(shield.AreYou("sword"), Is.False, "True");
            });
        }
```

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

        [Test]
        public void TestFullDescription()
        {
            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 Iteration2
{
    [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 TestFindItem()
        {
            Assert.Multiple(() =>
            {
                Assert.That(inventory.HasItem("sword"), Is.True);
                Assert.That(inventory.HasItem("shield"), Is.True);
            });
        }
        [Test]
        public void TestNoItemFind()
```

```
        {
            Assert.That(inventory.HasItem("potion"), Is.False);
        }
        [Test]
        public void TestFetchItem()
        {
            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"));
        }
    }
}
```

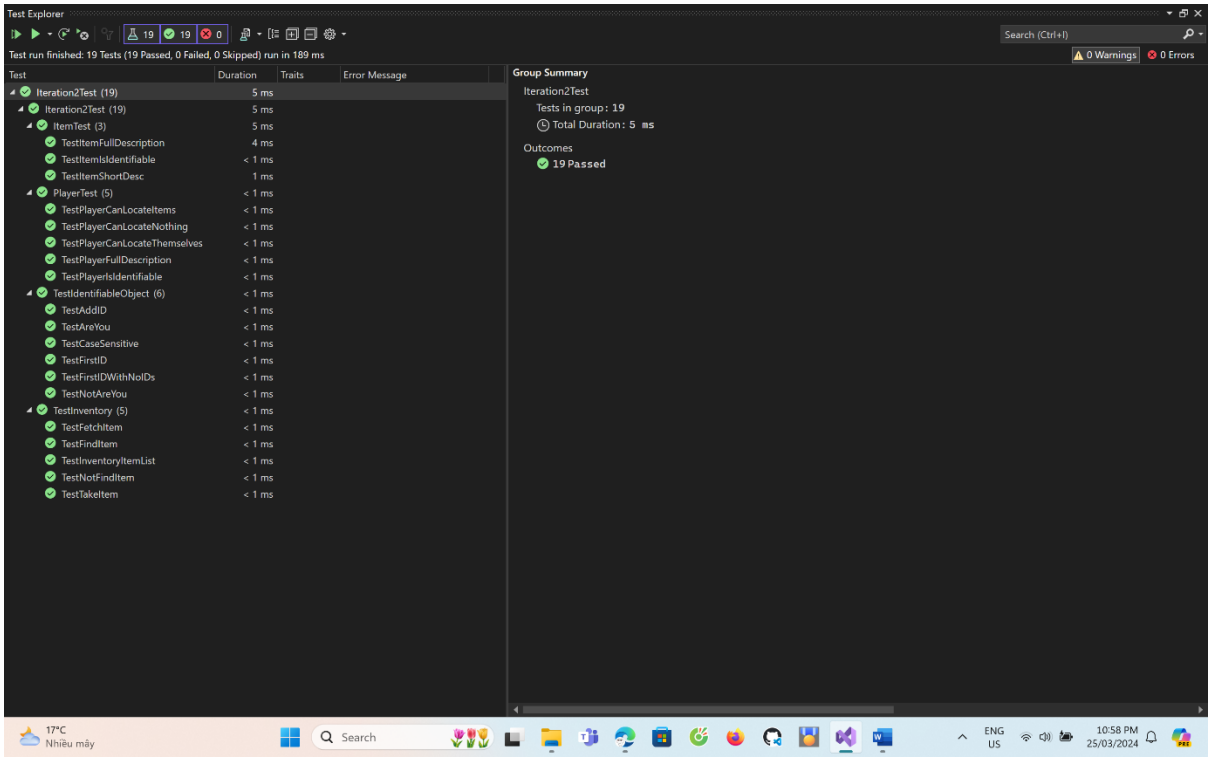
PlayersTest.cs

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

namespace Iteration2
{
    [TestFixture]
    public class TestPlayer
    {
        Inventory inventory;
        Player player;
        Item sword;
        Item shield;

        [SetUp]
        public void SetUp()
        {
            inventory = new Inventory();
            player = new("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 itemsLocated = player.Locate("sword");
            if (sword == itemsLocated)
            {
                result = true;
            }
            Assert.That(result, Is.True);

            _ = player.Locate("shield");
            if (shield == itemsLocated)
            {
                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"));
        }
    }
}
IdentifiableObject.cs



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

```
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;

namespace Iteration2
{
    [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 testIdentifiableObject = new(testArray);
            StringAssert.AreEqualIgnoringCase("",
testIdentifiableObject.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);
```

```
            });
        }
    }
}
```


Test Explorer
19
19
0
Search (Ctrl+I)
Test run finished: 19 Tests (19 Passed, 0 Failed, 0 Skipped) run in 189 ms
0 Warnings
0 Errors
Test
Duration
Traits
Error Message
Iteration2Test (19) 5 ms
Iteration2Test (19) 5 ms
ItemTest (3) 5 ms
TestItemFullDescription 4 ms
TestItemIsIdentifiable < 1 ms
TestItemShortDesc 1 ms
PlayerTest (5) < 1 ms
TestPlayerCanLocateItems < 1 ms
TestPlayerCanLocateNothing < 1 ms
TestPlayerCanLocateThemselves < 1 ms
TestPlayerFullDescription < 1 ms
TestPlayerIsIdentifiable < 1 ms
TestIdentifiableObject (6) < 1 ms
TestAddID < 1 ms
TestAreYou < 1 ms
TestCaseSensitive < 1 ms
TestFirstID < 1 ms
TestFirstIDWithNoIDs < 1 ms
TestNotAreYou < 1 ms
TestInventory (5) < 1 ms
TestFetchItem < 1 ms
TestFindItem < 1 ms
TestInventoryItemList < 1 ms
TestNotFindItem < 1 ms
TestTakeItem < 1 ms
Group Summary
Iteration2Test
Tests in group: 19
Total Duration: 5 ms
Outcomes
19 Passed
17°C
Nhiều mây
Search
ENG
US
10:58 PM
25/03/2024