COS20007: Object Oriented Programming

Pass Task 4.2: Case Study — Iteration 2: Players, Items, and Inventory

Show Wai Yan/105293041

GameObject.cs

```
namespace SwinAdventure
{
    public abstract class GameObject : IndentifiableObject
    {
        // Fields
        private string _description;
        private string _name;

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

        // Properties
        public string Name
        {
            get { return _name; }
        }

        public string ShortDescription
        {
            get { return $"a {Name.ToLower()} ({FirstId})"; }
        }

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

Item.cs

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

        }
    }
}
```

Inventor.cs

```
namespace SwinAdventure
{
    public class Inventory
    {
        // Fields
        private List<Item> _items = new List<Item>();

        // Constructor
        public Inventory()
        {

        }

        // Properties
        public string ItemList
        {
            get
            {
                string itemListText = "";
                foreach (Item item in _items)
                {
                    itemListText += $"\t{item.ShortDescription}\n";
                    // format the string to look like
                    // You are carrying
                    //    Item A
                    //    Item B
                    //    Item C
                }
                return itemListText;
            }
        }

        // Methods
        public bool HasItem(string id)
        {
            foreach (Item item in _items) // Finding through item
            {
                if (item.AreYou(id)) return true;
            }
            return false;
        }
        public void Put(Item itm)
        {
            _items.Add(itm);
        }
        public Item? Take(string id)
        {
            Item? itm = Fetch(id);

            if (itm == null) return null; // there is no such item
            _items.Remove(itm); // Remove from item list due to taken out
            return itm;
        }
        public Item? Fetch(string id)
        {
            foreach (Item item in _items) // Finding through list
            {
                if (item.AreYou(id)) return item; // check item is in inventory
            }
            return null; // null if not exist
        }
    }
}
```

Player.cs

```
namespace SwinAdventure
{
    public class Player : GameObject
    {
        // Field
        private Inventory _inventory = new Inventory();

        // Constructor
        public Player(string name, string desc) : base(new string[] { "me", "inventory" }, name, desc)
        {

        }

    // Properties
        public override string FullDescription
        {
            get { return $"You are {Name} {base.FullDescription}\nYou are carrying\n {Inventory.ItemList}"; }
        }

        public Inventory Inventory
        {
            get { return _inventory; }
        }

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

TestItem.cs

```
using System;
using SwinAdventure;
using NUnit.Framework;
using NUnit.Framework.Legacy;

namespace UnitTests
{
    [TestFixture]
    public class TestItem
    {
        private Item testItem;

        [SetUp]
        public void Setup()
        {
            testItem = new Item(new string[]{ "sword", "bronze sword" }, "Bronze Sword", "A shiny bronze sword");
        }

        [Test]
        public void TestItemIsIdentifiable()
        {
            ClassicAssert.True(testItem.AreYou("sword"));
            ClassicAssert.True(testItem.AreYou("bronze sword"));
            ClassicAssert.False(testItem.AreYou("golden sword"));
        }

        [Test]
        public void TestShortDescription()
        {
            ClassicAssert.AreEqual("a bronze sword (sword)", testItem.ShortDescription);
        }

        [Test]
        public void TestFullDescription()
        {
            string description = "A shiny bronze sword";

            ClassicAssert.AreEqual(description, testItem.FullDescription);
        }

        [Test]
        public void TestPrivilegeEscalarion()
        {
            string myStudentID = "105293041";
            testItem.PrivilegeEscalation("3041");

            // Test that aftert escalation the first id should be my student id;
            ClassicAssert.AreEqual(myStudentID, testItem.FirstId);
        }
    }
}
```

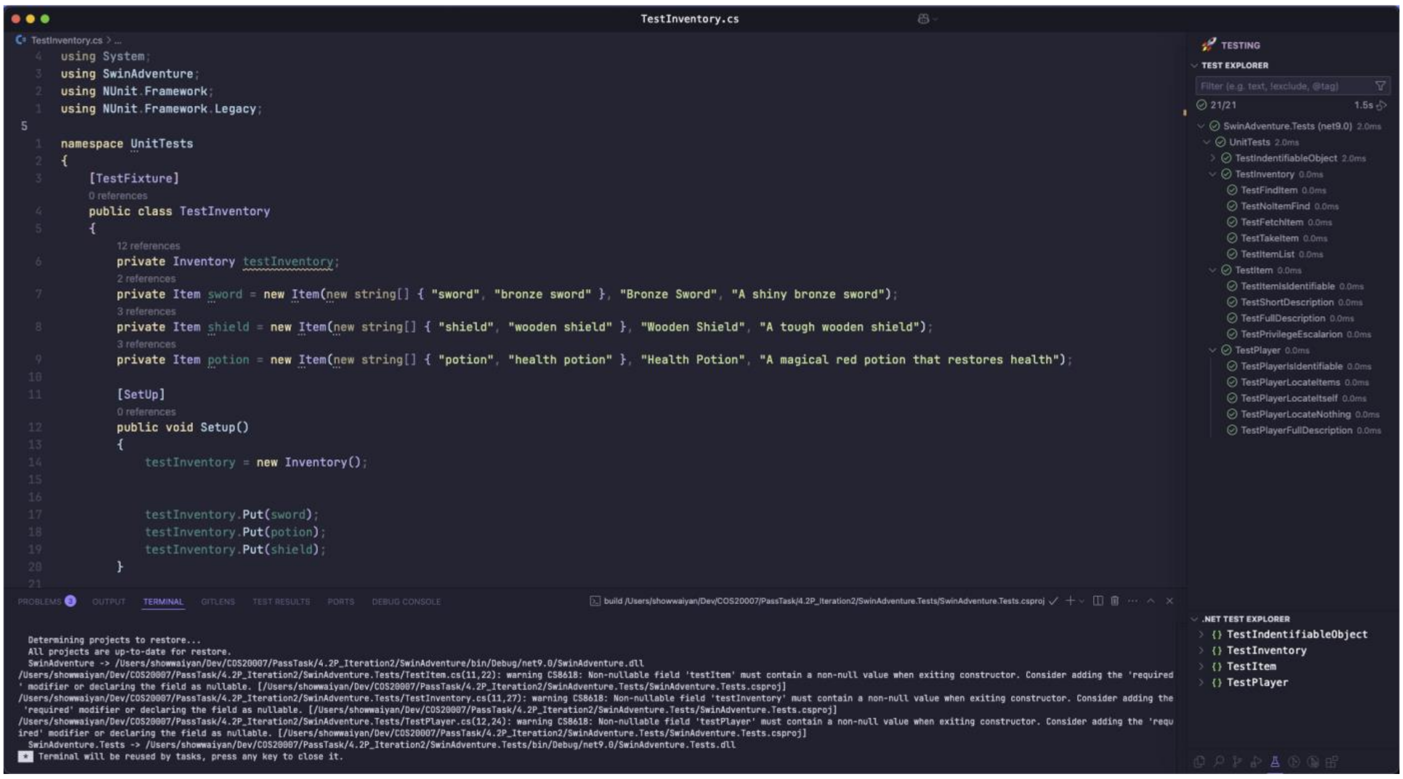
TestInventory.cs

```
using System;
using SwinAdventure;
using NUnit.Framework;
using NUnit.Framework.Legacy;

namespace UnitTests
{
    [TestFixture]
    public class TestInventory
    {
        private Inventory testInventory;
        private Item sword = new Item(new string[] { "sword", "bronze sword" }, "Bronze Sword", "A shiny bronze sword");
        private Item shield = new Item(new string[] { "shield", "wooden shield" }, "Wooden Shield", "A tough wooden shield");
        private Item potion = new Item(new string[] { "potion", "health potion" }, "Health Potion", "A magical red potion that
restores health");

        [SetUp]
        public void Setup()
        {
            testInventory = new Inventory();


            testInventory.Put(sword);
            testInventory.Put(potion);
            testInventory.Put(shield);
        }

        [Test]
        public void TestFindItem()
        {
            ClassicAssert.True(testInventory.HasItem("sword"));
            ClassicAssert.True(testInventory.HasItem("potion"));
        }

        [Test]
        public void TestNoItemFind()
        {
            ClassicAssert.False(testInventory.HasItem("arrow"));
        }

        [Test]
        public void TestFetchItem()
        {
            ClassicAssert.That(shield, Is.EqualTo(testInventory.Fetch("wooden shield")));
            ClassicAssert.True(testInventory.HasItem("wooden shield"));
        }

        [Test]
        public void TestTakeItem()
        {
            ClassicAssert.That(potion, Is.EqualTo(testInventory.Take("health potion")));
            ClassicAssert.False(testInventory.HasItem("health potion"));
        }

        [Test]
        public void TestItemList()
        {
            string testList = $"\t{sword.ShortDescription}\n\t{potion.ShortDescription}\n\t{shield.ShortDescription}\n";
            ClassicAssert.That(testList, Is.EqualTo(testInventory.ItemList));
        }
    }
}
```

TestPlayer.cs

```
using SwinAdventure;
using NUnit.Framework;
using NUnit.Framework.Legacy;

namespace UnitTests
{

    [TestFixture]
    public class TestPlayer
    {
        private Player testPlayer;
        private Item sword = new Item(new string[] { "sword", "bronze sword" }, "Bronze Sword", "A shiny bronze sword");
        private Item shield = new Item(new string[] { "shield", "wooden shield" }, "Wooden Shield", "A tough wooden shield");
        private Item potion = new Item(new string[] { "potion", "health potion" }, "Health Potion", "A magical red potion that
restores health");

        [SetUp]
        public void Setup()
        {
            testPlayer = new Player("Show", "The Programmer");

            testPlayer.Inventory.Put(sword);
            testPlayer.Inventory.Put(shield);
            testPlayer.Inventory.Put(potion);
        }

        [Test]
        public void TestPlayerIsIdentifiable()
        {
            ClassicAssert.True(testPlayer.AreYou("me"));
            ClassicAssert.True(testPlayer.AreYou("inventory"));
        }

        [Test]
        public void TestPlayerLocateItems()
        {
            ClassicAssert.That(sword, Is.EqualTo(testPlayer.Locate("sword")));
            ClassicAssert.True(testPlayer.Inventory.HasItem("sword"));
            ClassicAssert.That(shield, Is.EqualTo(testPlayer.Locate("wooden shield")));
            ClassicAssert.True(testPlayer.Inventory.HasItem("wooden shield"));
        }

        [Test]
        public void TestPlayerLocateItself()
        {
            ClassicAssert.That(testPlayer, Is.EqualTo(testPlayer.Locate("me")));
            ClassicAssert.That(testPlayer, Is.EqualTo(testPlayer.Locate("inventory")));
        }

        [Test]
        public void TestPlayerLocateNothing()
        {
            ClassicAssert.That(testPlayer.Locate("gun"), Is.EqualTo(null));
        }

        [Test]
        public void TestPlayerFullDescription()
        {
            string testDescription = $"You are Show The Programmer\nYou are carrying\n
\t{sword.ShortDescription}\n\t{shield.ShortDescription}\n\t{potion.ShortDescription}\n";
            ClassicAssert.That(testPlayer.FullDescription, Is.EqualTo(testDescription));
        }
    }
}
```

Screenshot of the Test Explorer showing all your unit test running


TestInventory.cs
TestInventory.cs > ...
PROBLEMS 3
OUTPUT
TERMINAL
GITLENS
TEST RESULTS
PORTS
DEBUG CONSOLE
build /Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/SwinAdventure.Tests.csproj
TESTING
TEST EXPLORER
Filter (e.g. text, !exclude, @tag)
.NET TEST EXPLORER

```
using System;
using SwinAdventure;
using NUnit.Framework;
using NUnit.Framework.Legacy;

namespace UnitTests
{
    [TestFixture]
    0 references
    public class TestInventory
    {
        12 references
        private Inventory testInventory;
        2 references
        private Item sword = new Item(new string[] { "sword", "bronze sword" }, "Bronze Sword", "A shiny bronze sword");
        3 references
        private Item shield = new Item(new string[] { "shield", "wooden shield" }, "Wooden Shield", "A tough wooden shield");
        3 references
        private Item potion = new Item(new string[] { "potion", "health potion" }, "Health Potion", "A magical red potion that restores health");

        [SetUp]
        0 references
        public void Setup()
        {
            testInventory = new Inventory();


            testInventory.Put(sword);
            testInventory.Put(potion);
            testInventory.Put(shield);
        }
```

```
  Determining projects to restore...
  All projects are up-to-date for restore.
  SwinAdventure -> /Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure/bin/Debug/net9.0/SwinAdventure.dll
/Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/TestItem.cs(11,22): warning CS8618: Non-nullable field 'testItem' must contain a non-null value when exiting constructor. Consider adding the 'required
' modifier or declaring the field as nullable. [/Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/SwinAdventure.Tests.csproj]
/Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/TestInventory.cs(11,27): warning CS8618: Non-nullable field 'testInventory' must contain a non-null value when exiting constructor. Consider adding the
 'required' modifier or declaring the field as nullable. [/Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/SwinAdventure.Tests.csproj]
/Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/TestPlayer.cs(12,24): warning CS8618: Non-nullable field 'testPlayer' must contain a non-null value when exiting constructor. Consider adding the 'requ
ired' modifier or declaring the field as nullable. [/Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/SwinAdventure.Tests.csproj]
  SwinAdventure.Tests -> /Users/showwaiyan/Dev/COS20007/PassTask/4.2P_Iteration2/SwinAdventure.Tests/bin/Debug/net9.0/SwinAdventure.Tests.dll
 *  Terminal will be reused by tasks, press any key to close it.
```

```
✓ 21/21                                        1.5s
  ✓ SwinAdventure.Tests (net9.0) 2.0ms
    ✓ UnitTests 2.0ms
      ✓ TestIndentifiableObject 2.0ms
      ✓ TestInventory 0.0ms
        ✓ TestFindItem 0.0ms
        ✓ TestNoItemFind 0.0ms
        ✓ TestFetchItem 0.0ms
        ✓ TestTakeItem 0.0ms
        ✓ TestItemList 0.0ms
      ✓ TestItem 0.0ms
        ✓ TestItemIsIdentifiable 0.0ms
        ✓ TestShortDescription 0.0ms
        ✓ TestFullDescription 0.0ms
        ✓ TestPrivilegeEscalarion 0.0ms
      ✓ TestPlayer 0.0ms
        ✓ TestPlayerIsIdentifiable 0.0ms
        ✓ TestPlayerLocateItems 0.0ms
        ✓ TestPlayerLocateItself 0.0ms
        ✓ TestPlayerLocateNothing 0.0ms
        ✓ TestPlayerFullDescription 0.0ms
```

```
> {} TestIndentifiableObject
> {} TestInventory
> {} TestItem
> {} TestPlayer
```