School of Science, Computing and Engineering Technologies

Object Oriented Programming

Week 8: SwinAdventure — Iteration 5

Overview

In this week, there are two tasks 8.1 and 8.2. Each task contributes 2% to your final grade. Noting that you need to complete this task before coming to your allocated lab. In the lab, there will be a verification task and short interview to verify your understanding.

These tasks extend from your current SwinAdventure application that you created in the task 5.2 of Week 5.

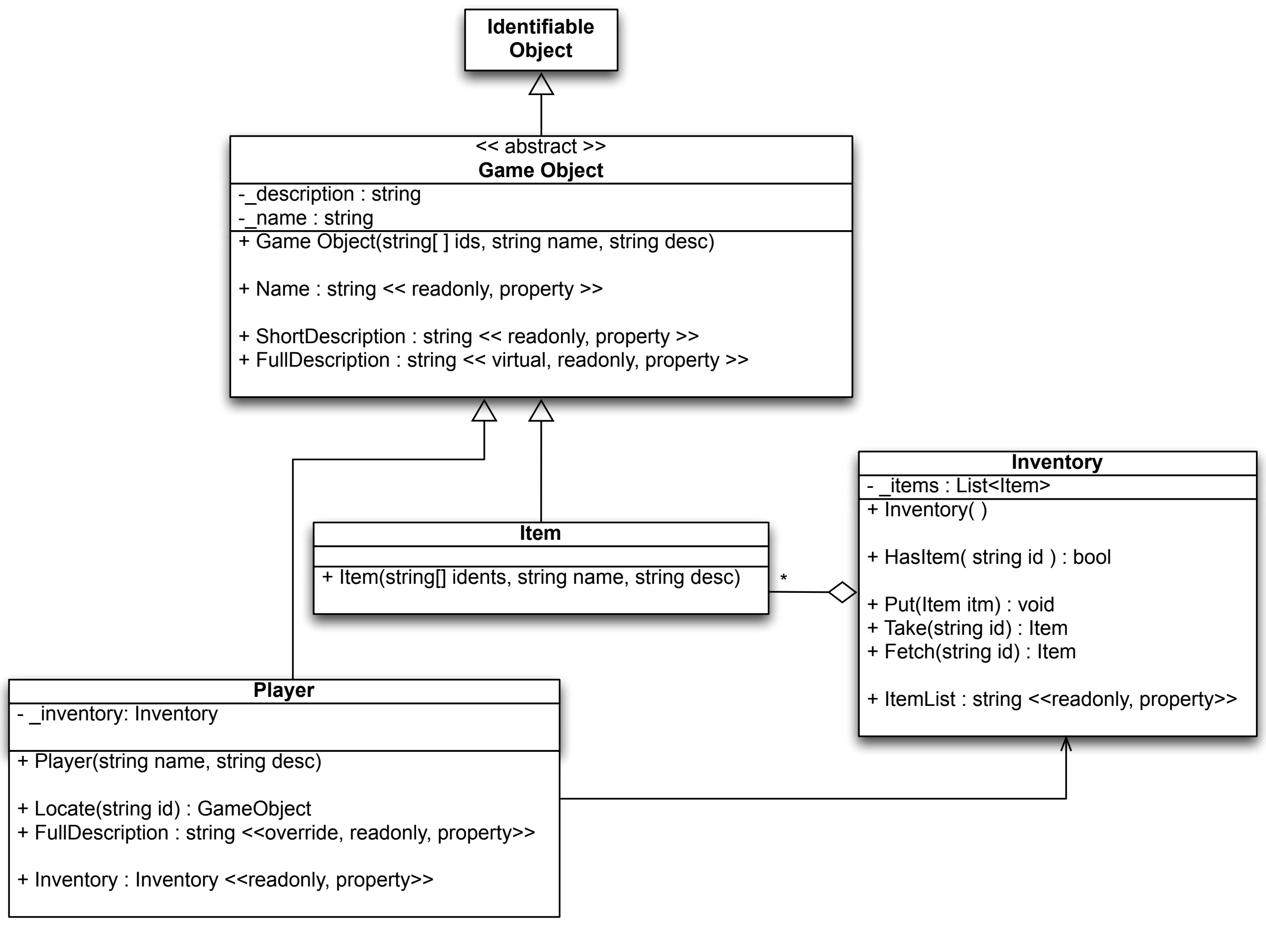
Purposes

Task 8.1	This task aims to develop a Player class for the SwinAdventure game. You will then create unit test cases for this class.
Task 8.2	This task aims to save the current player's information to a file. You can later open the file to verify whether the player's information has been stored correctly.

Instructions

1. Review the ***Case Study Requirements*** document. It outlines what you need to create.

2. In the previous Iterations 2 and 3 and 4 of the SwinAdventure game, you have optimized the code by using inheritance and creating a **GameObject** class. In addition, you also implemented the **Item** and **Inventory** classes.
3. With the above current implementation, the goal of this task is to develop a **Player** class.
4. We provide you with the newly updated UML class diagram as follows.


Identifiable Object
<< abstract >>
Game Object
-_description : string
-_name : string
+ Game Object(string[ ] ids, string name, string desc)
+ Name : string << readonly, property >>
+ ShortDescription : string << readonly, property >>
+ FullDescription : string << virtual, readonly, property >>
Item
+ Item(string[] idents, string name, string desc)
*
Inventory
- _items : List<Item>
+ Inventory( )
+ HasItem(string id) : bool
+ Put(Item itm) : void
+ Take(string id) : Item
+ Fetch(string id) : Item
+ ItemList : string <<readonly, property>>
Player
- _inventory: Inventory
+ Player(string name, string desc)
+ Locate(string id) : GameObject
+ FullDescription : string <<override, readonly, property>>
+ Inventory : Inventory <<readonly, property>>

Player is also a kind of **Game Object**. This will be a object through which the player will interact with the game world.

- The Player constructor will call the GameObject constructor and pass up identifiers for "me" and "inventory".

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

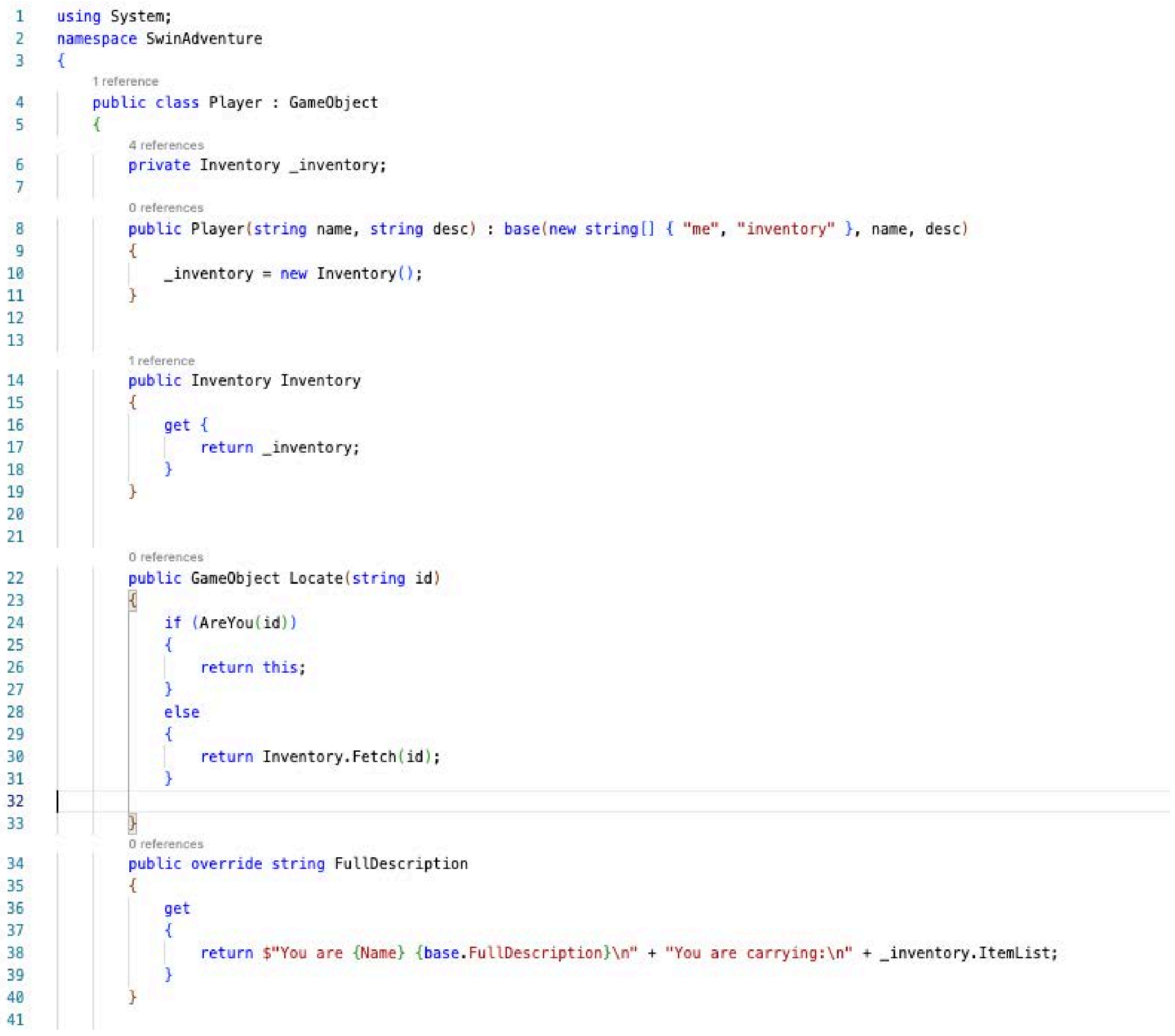
Player - has following fields and methods

■ An Inventory object is used to manage the Player's items

■ Full Description is overridden to include the player’s name, description, and details of the items in the player's inventory.

■ Locate "finds" a GameObject somewhere around the player. At this stage that includes the player themselves, or an item the player has in their inventory

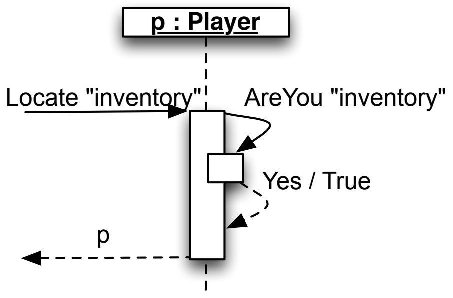
5. The below sample code demonstrates how to implement this class - Player



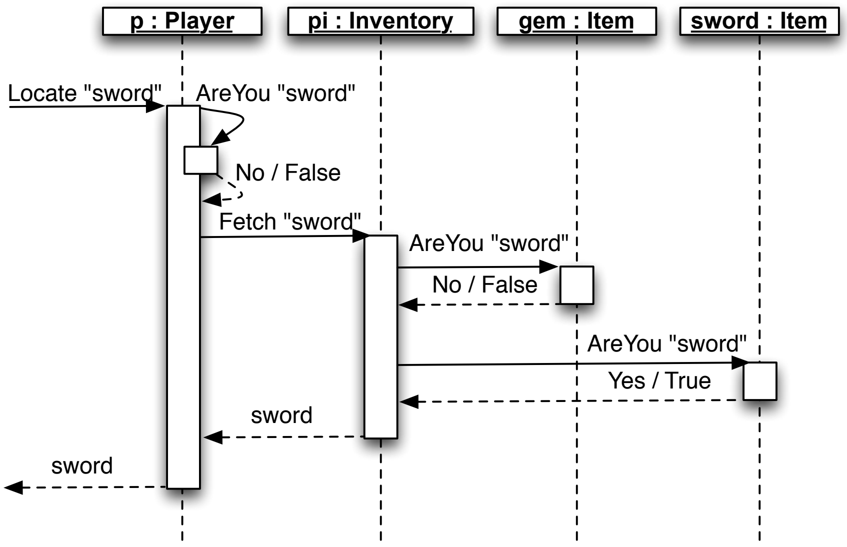
```
using System;
namespace SwinAdventure
{
    1 reference
    public class Player : GameObject
    {
        4 references
        private Inventory _inventory;

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


        1 reference
        public Inventory Inventory
        {
            get {
                return _inventory;
            }
        }


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

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

```

6. The following UML sequence diagrams shows the sequence of messages involved in locating the player and their items.

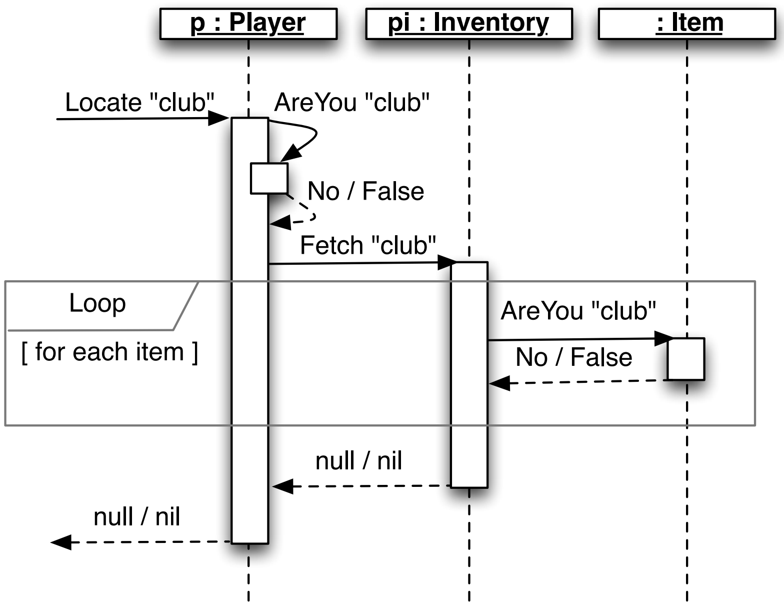
The player can "locate" themselves.



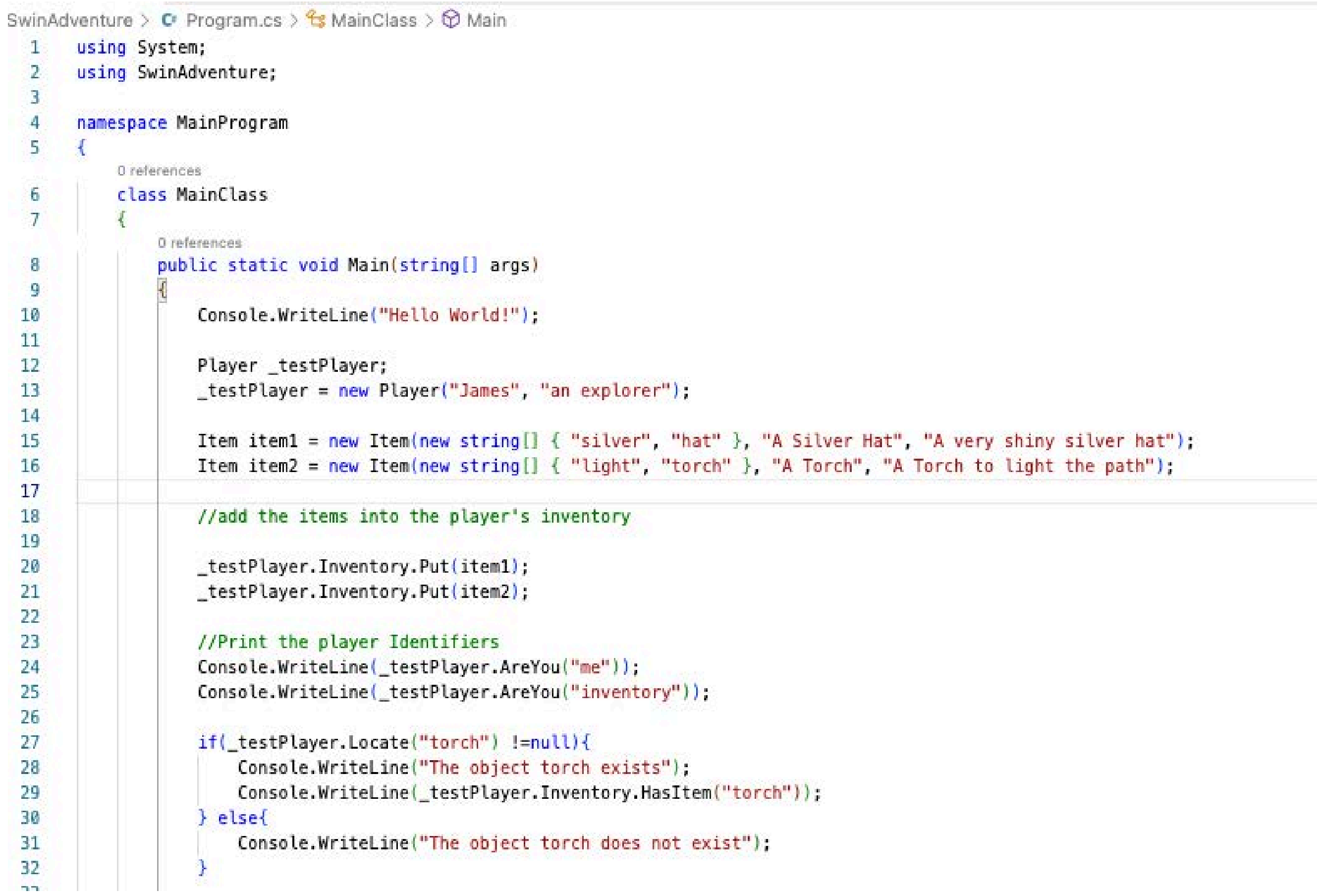
The player can locate items in their inventory. Note: *pi* is the player's inventory object.



When there are no items that match then null/nil is returned.



7. We develop the main Program.cs as follow. The sample source code of these screenshots can be downloaded from the Assignment module. Run the project to obtain the result

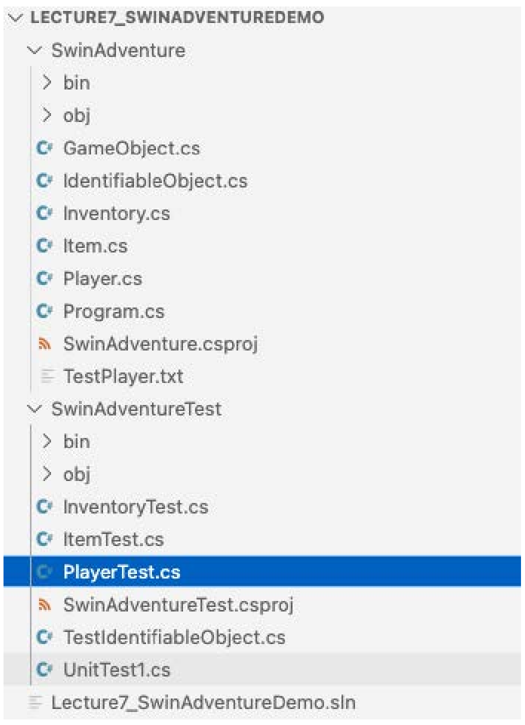


SwinAdventure > Program.cs > MainClass > Main

```
using System;
using SwinAdventure;

namespace MainProgram
{
    0 references
    class MainClass
    {
        0 references
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");

            Player _testPlayer;
            _testPlayer = new Player("James", "an explorer");

            Item item1 = new Item(new string[] { "silver", "hat" }, "A Silver Hat", "A very shiny silver hat");
            Item item2 = new Item(new string[] { "light", "torch" }, "A Torch", "A Torch to light the path");

            //add the items into the player's inventory

            _testPlayer.Inventory.Put(item1);
            _testPlayer.Inventory.Put(item2);

            //Print the player Identifiers
            Console.WriteLine(_testPlayer.AreYou("me"));
            Console.WriteLine(_testPlayer.AreYou("inventory"));

            if(_testPlayer.Locate("torch") !=null){
                Console.WriteLine("The object torch exists");
                Console.WriteLine(_testPlayer.Inventory.HasItem("torch"));
            } else{
                Console.WriteLine("The object torch does not exist");
            }
```

The above code puts two items into the player's inventory. It then try to test whether the object relating to the identifier "torch" exists in the inventory.

8. In your SwinAdventureTest project, we create a new **PlayerTest.cs** that contains test cases for the Player class. The file structure of this test project is as follows.

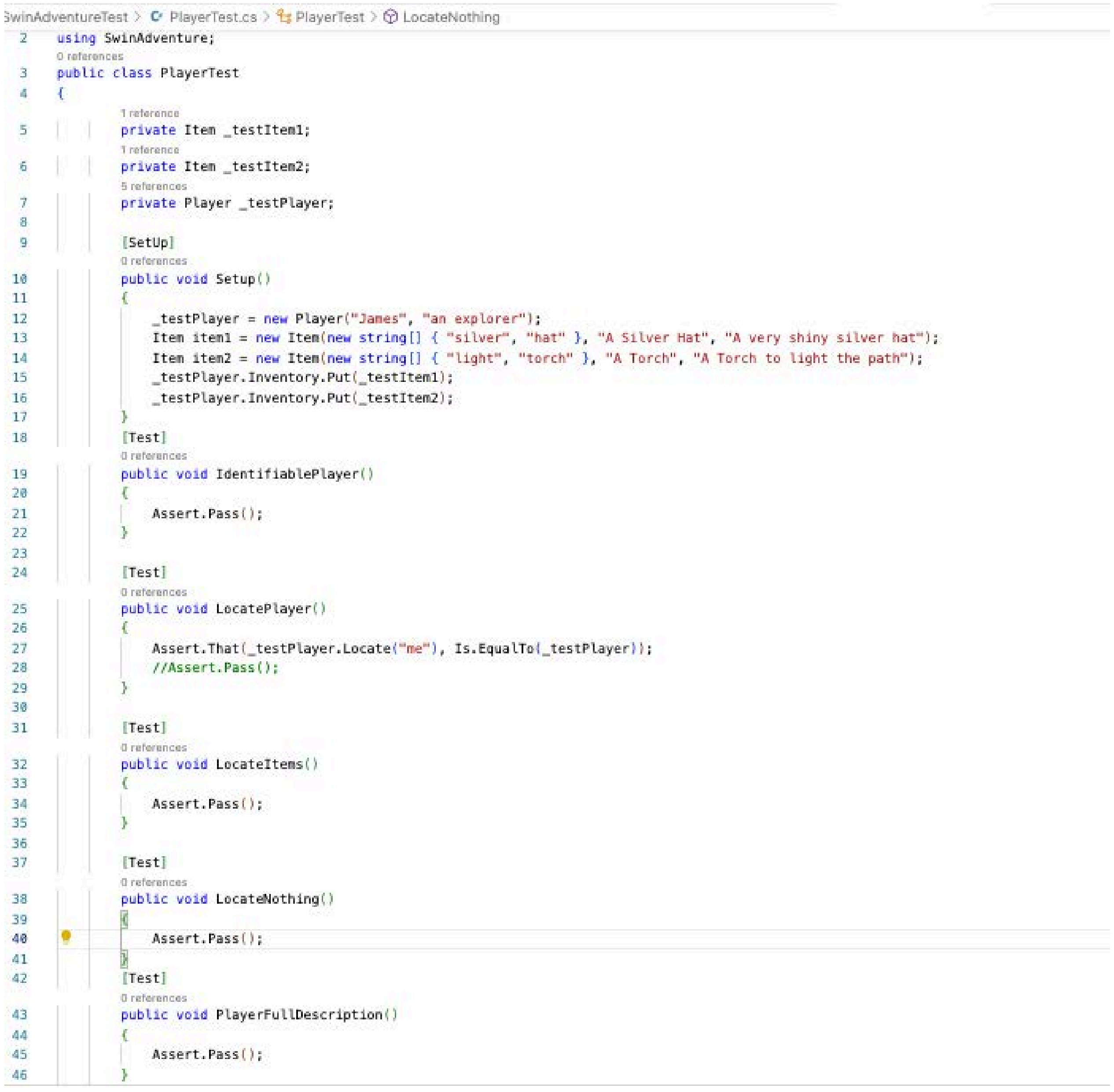


```
LECTURE7_SWINADVENTUREDEMO
  SwinAdventure
    > bin
    > obj
    GameObject.cs
    IdentifiableObject.cs
    Inventory.cs
    Item.cs
    Player.cs
    Program.cs
    SwinAdventure.csproj
    TestPlayer.txt
  SwinAdventureTest
    > bin
    > obj
    InventoryTest.cs
    ItemTest.cs
    PlayerTest.cs
    SwinAdventureTest.csproj
    TestIdentifiableObject.cs
    UnitTest1.cs
  Lecture7_SwinAdventureDemo.sln
```

9. In the **PlayerTest.cs,** please develop following test cases using the provided sample source code shown in Step 7.

Player Unit Tests	
Test Player is Identifiable	The player responds correctly to "Are You" requests based on its default identifiers (me and inventory).
Test Player Locates Items	The player can locate items in its inventory, this returns items the player has and the item remains in the player's inventory.
Test Player Locates itself	The player returns itself if asked to locate "me" or "inventory".
Test Player Locates nothing	The player returns a null/nil object if asked to locate something it does not have.
Test Player Full Description	The player's full description contains the text “You are (the player’s name), (the player’s description). You are carrying:" and the short descriptions of the items the player has (from its inventory's item list)

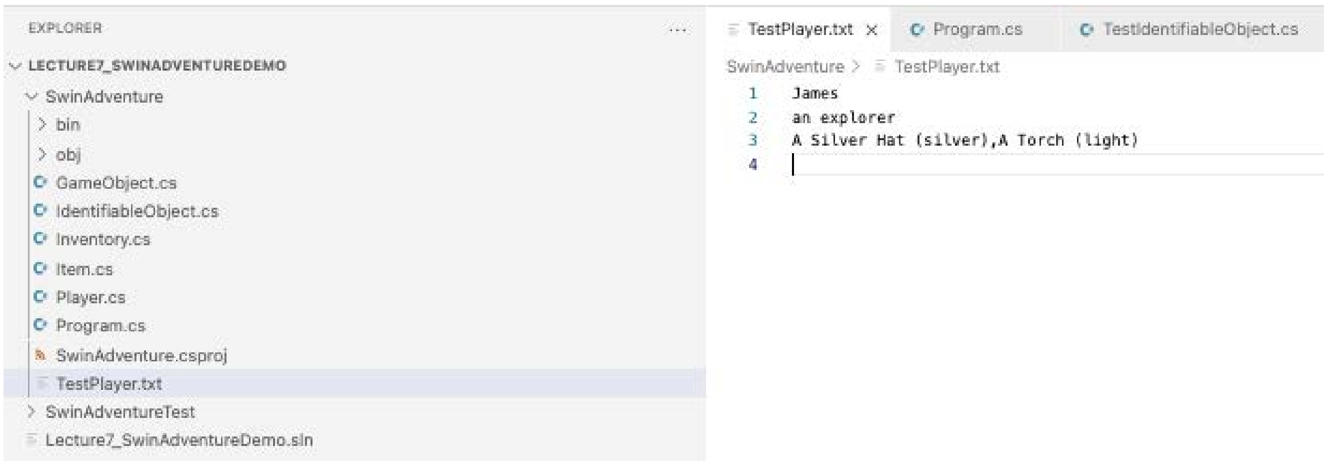
Please see the below template for the test cases. The source code is in Assignment module on Canvas.


SwinAdventureTest > PlayerTest.cs > PlayerTest > LocateNothing

```
using SwinAdventure;
public class PlayerTest
{
    private Item _testItem1;
    private Item _testItem2;
    private Player _testPlayer;

    [SetUp]
    public void Setup()
    {
        _testPlayer = new Player("James", "an explorer");
        Item item1 = new Item(new string[] { "silver", "hat" }, "A Silver Hat", "A very shiny silver hat");
        Item item2 = new Item(new string[] { "light", "torch" }, "A Torch", "A Torch to light the path");
        _testPlayer.Inventory.Put(_testItem1);
        _testPlayer.Inventory.Put(_testItem2);
    }
    [Test]
    public void IdentifiablePlayer()
    {
        Assert.Pass();
    }

    [Test]
    public void LocatePlayer()
    {
        Assert.That(_testPlayer.Locate("me"), Is.EqualTo(_testPlayer));
        //Assert.Pass();
    }

    [Test]
    public void LocateItems()
    {
        Assert.Pass();
    }

    [Test]
    public void LocateNothing()
    {
        Assert.Pass();
    }
    [Test]
    public void PlayerFullDescription()
    {
        Assert.Pass();
    }
```

10. The task 8.2 requires the SwinAdventure program can store the information of the player into a file. Then, the program can read the file content to display the information of the player into console.

The below screenshot demonstrates the content of the file **TestPlayer.txt**

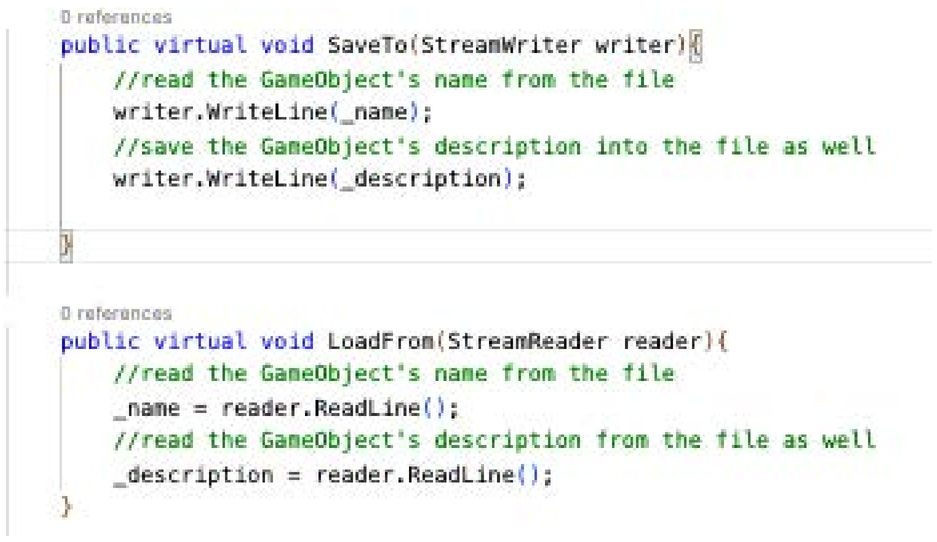


```
James
an explorer
A Silver Hat (silver),A Torch (light)

```

In the **TestPlayer.txt**, the first line displays the name of the player. The second line is about the description of the player. The third line is about the description of items in the inventory of the player. The description of each item is separated by a comma. The above screenshot shows that the play has two items.

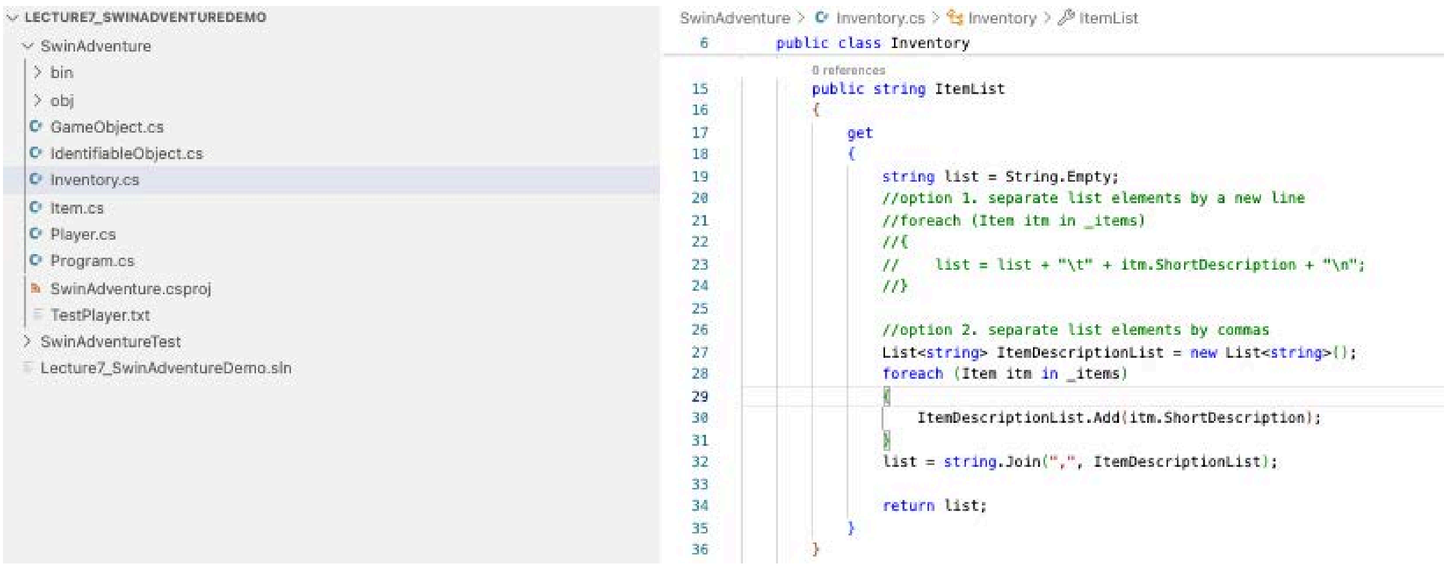
11. From the above UML class diagram at page 2, it shows that the Player class inherits from the GameObject. Hence, the GameObject class can stores some generalised information of the player class including the name and the description of the player.

12. Open the GameObject file and add following two methods



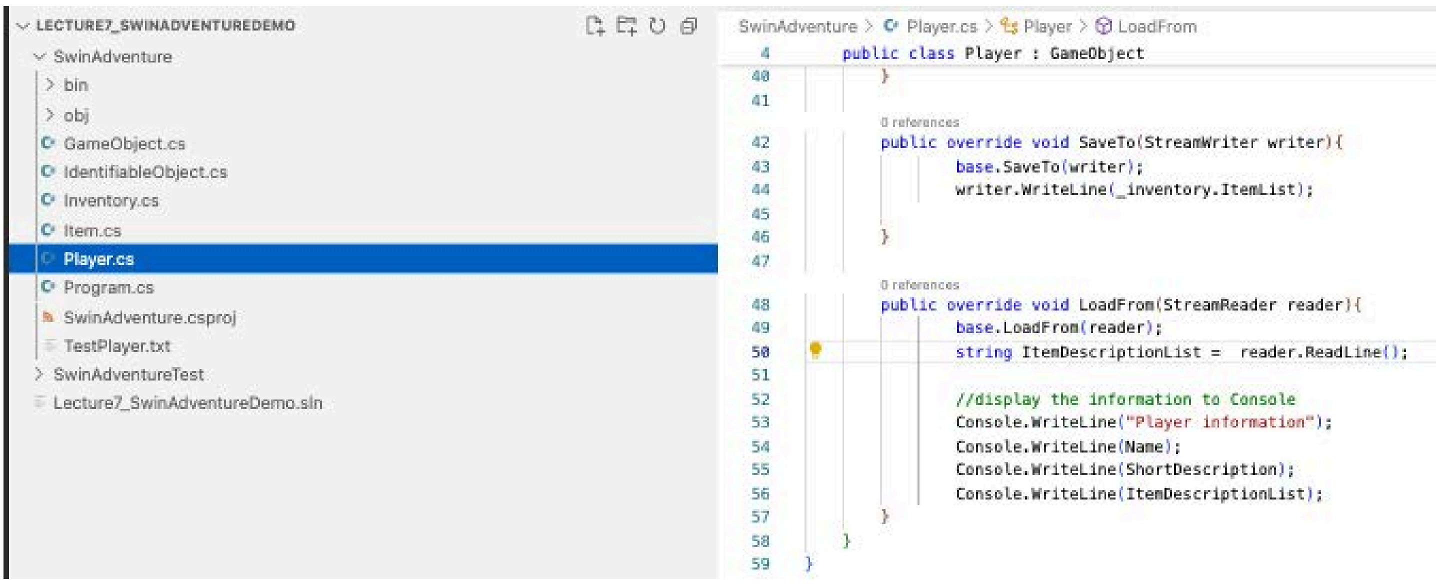
```
0 references
public virtual void SaveTo(StreamWriter writer){
    //read the GameObject's name from the file
    writer.WriteLine(_name);
    //save the GameObject's description into the file as well
    writer.WriteLine(_description);

}

0 references
public virtual void LoadFrom(StreamReader reader){
    //read the GameObject's name from the file
    _name = reader.ReadLine();
    //read the GameObject's description from the file as well
    _description = reader.ReadLine();
}
```

13. Open the **Inventory.cs** and modify the ItemList property to allow the returned list is formated by commas.



```
    public class Inventory
        0 references
        public string ItemList
        {
            get
            {
                string list = String.Empty;
                //option 1. separate list elements by a new line
                //foreach (Item itm in _items)
                //{
                //    list = list + "\t" + itm.ShortDescription + "\n";
                //}

                //option 2. separate list elements by commas
                List<string> ItemDescriptionList = new List<string>();
                foreach (Item itm in _items)
                {
                    ItemDescriptionList.Add(itm.ShortDescription);
                }
                list = string.Join(",", ItemDescriptionList);

                return list;
            }
        }
```

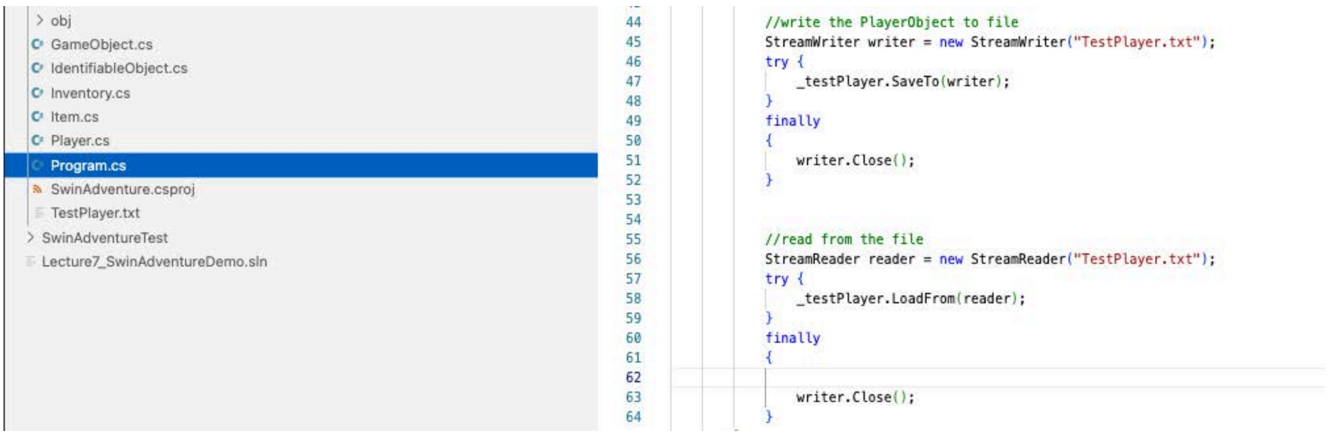
14. Open the **Player.cs** and add two overriding methods as follows. These methods override the base methods in the **GameObject.cs**



```
    public class Player : GameObject
        }

        0 references
        public override void SaveTo(StreamWriter writer){
                base.SaveTo(writer);
                writer.WriteLine(_inventory.ItemList);

        }

        0 references
        public override void LoadFrom(StreamReader reader){
                base.LoadFrom(reader);
                string ItemDescriptionList =  reader.ReadLine();

                //display the information to Console
                Console.WriteLine("Player information");
                Console.WriteLine(Name);
                Console.WriteLine(ShortDescription);
                Console.WriteLine(ItemDescriptionList);
        }
    }
}
```

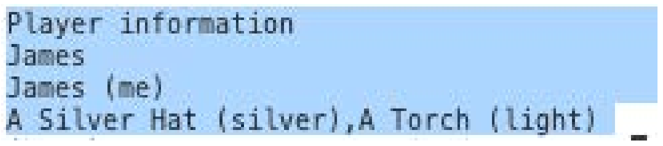
15. Open the **Program.cs** and test the above methods. The sample source code is in the next page.

16. The code in the screenshots writes the player information into the file and later read the file content again



```
//write the PlayerObject to file
StreamWriter writer = new StreamWriter("TestPlayer.txt");
try {
    _testPlayer.SaveTo(writer);
}
finally
{
    writer.Close();
}


//read from the file
StreamReader reader = new StreamReader("TestPlayer.txt");
try {
    _testPlayer.LoadFrom(reader);
}
finally
{

    writer.Close();
}
```

17. The terminal should display following result



```
Player information
James
James (me)
A Silver Hat (silver),A Torch (light)
```

See you in the lab