Swinburne University of Technology

COS20007 Object Oriented Programming

Case Study - Iteration 2 - Players Items and Inventory

PDF generated at 11:43 on Friday 22nd September, 2023

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

namespace Iteration2
{

// The GameObject class serves as a base class for game objects

    public abstract class GameObject : IdentifiableObject
    {

// Private fields to store the name and description of the game object

        private string _name;
        private string _description;


// constructor for creating a GameObject
// takes an array of identifiers, name and description as parameters
// calls the base class ( identifiableobject )constructor with the ids

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


// property to return the name of the game object

        public string Name
        {
            get { return _name; }
        }


// property to return the short description of the game object

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

//virtual property to return the full description of the game object

        public virtual string FullDescription
        {
```

```
54              get
55              {
56
57  // returns the description stored in the private field
58
59                  return _description;
60              }
61
62          }
63      }
64  }
65
```

```
using Iteration2;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace Iteration2
{

// The Player class represents a player character in the game,inherited from GameObject

    public class Player : GameObject
    {

// Private field to store the inventory of the player

        private Inventory _inventory;

// constructor for creating a player with the given name and description

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

// initialise the inventory

            _inventory = new Inventory();
        }


// Locates a game object based on a its id

        public GameObject Locate(string id)
        {
            if (AreYou(id))
            {

//Returns the player if the id matches the player's id

                return this;
            }

// Searches the inventory for an item with the given id

            return _inventory.Fetch(id);
        }


// Overrides the FullDescription property of the GameObject class to provide a custom description of the player

```

```
52          public override string FullDescription
53          {
54              get
55              {
56
57  // returns the description indicating the player's name and the itema in their
    ↪  inventory
58
59                  return $"You are {Name}, {base.FullDescription} \nyou are
    ↪  carrying:\n\t" + _inventory.ItemList;
60              }
61          }
62
63
64  // Property to acess the inventory of the player
65
66          public Inventory Inventory
67          {
68              get
69
70  // Allows external code to acess the inventory
71  //
72              { return _inventory; }
73          }
74
75      }
76  }
```

```
using Iteration2;

namespace TestProject3
{
    [TestFixture]
    public class TestPlayer
    {
        Player player;
        Item stool;

        [SetUp]
        public void Setup()
        {
            player = new Player("Aaryan", "the student");
            stool = new Item(new string[] { "stool" }, "50cm stool", "This is a stool");
            player.Inventory.Put(stool);
        }

        [Test]
        public void TestIsIdentifiable()
        {
            Assert.That(player.AreYou("me"), Is.True);
            Assert.That(player.AreYou("inventory"), Is.True);
        }
        [Test]
        public void TestLocateItems()
        {
            Assert.That(player.Locate("stool"), Is.SameAs(stool));
            Assert.That(player.Inventory.HasItem("stool"), Is.True);
        }
        [Test]
        public void TestLocateItself()
        {
            Assert.That(player.Locate("me"), Is.SameAs(player));
            Assert.That(player.Locate("inventory"), Is.SameAs(player));
        }
        [Test]
        public void TestLocateNothing()
        {
            Assert.That(player.Locate("ssswor"), Is.SameAs(null));

        }
        [Test]
        public void TestFullDescription()
        {
            Assert.That(player.FullDescription, Is.EqualTo("You are Aaryan, the student \nyou are carrying:\n\ta 50cm stool (stool)"));
        }
    }
}
```

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

namespace Iteration2
{

// The Item class represents an item in the game, inherited from GameObject

    public class Item : GameObject
    {

// Constructor for creating a new item object with identifiers, name and description

        public Item(string[] idents, string name, string descrption) : base(idents,
 ↪  name, descrption)
        {
// The constructor of the base class is called with the provided parameters
        }
    }
}
```

```
using Iteration2;
using NUnit.Framework;
using System;

namespace TestProject
{
    [TestFixture]

    public class Tests
    {
        Item sword;
        [SetUp]
        public void Setup()
        {
            sword = new Item(new string[] { "stool" }, "50cm stool", "This is a
↪   stool");
        }
        [Test]
        public void TestItemIsIdentifiable()
        {
            Assert.That(sword.AreYou("stool"), Is.True);
            Assert.That(sword.AreYou("katana"), Is.False);
        }
        [Test]
        public void TestShortDescription()
        {
            Assert.That(sword.ShortDescription, Is.EqualTo("a 50cm stool (stool)"));
        }
        [Test]
        public void TestFullDescription()
        {
            Assert.That(sword.FullDescription, Is.EqualTo("This is a stool"));
        }
    }
}
```

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

namespace Iteration2
{

// the inventory class represents a collection of items that player can carry

    public class Inventory
    {

// private field to store the list of items

        private List<Item> _items;


// constructor to create a new empty inventory

        public Inventory()
        {

// Initialise the items list as an empty list of Item objects

            _items = new List<Item>();
        }


// check if the inventory contain an item with the given id

        public bool HasItem(string id)
        {
            foreach (var item in _items)
            {

// calls the AreYou method of the item to check if it has the given id

                if (item.AreYou(id))
                {

//return true if a matching item is found

                    return true;
                }

            }

// return false if no matching item is found

            return false;
        }

```

```
54
55  //adds an item to the inventory
56
57          public void Put(Item item)
58          {
59
60  // adds the provided item to the inventory list
61
62              _items.Add(item);
63          }
64
65
66  // removes and returns an item from the inventory with the given id
67
68          public Item Take(string id)
69          {
70
71  //  calls the Fetch method to find the item with the given id
72
73              Item item = this.Fetch(id);
74
75  // if the item is found, remove it from the inventory list and return it
76
77              if (item != null)
78              {
79                  _items.Remove(item);
80              }
81
82  // if the item is not found, return null
83
84              return item;
85          }
86
87  // Retrives item from the inventory based on its id
88
89          public Item Fetch(string id)
90          {
91              foreach (Item item in _items)
92              {
93
94  // calls the AreYou method of the item to check if it has the given id
95
96                  if (item.AreYou(id))
97                  {
98
99  // return the item if it has the given id
100
101                     return item;
102                 }
103             }
104
105 // return null if no matching item is found
106
```

```
107             return null;
108         }
109 
110 // Property to get a string containing the short description of each item in the
    ↪  inventory
111 
112         public string ItemList
113         {
114             get
115             {
116                 string itemList = "";
117                 foreach (Item item in _items)
118                 {
119 
120 // concatenate the short description of each item with a new line character
121 
122                     itemList += $"{item.ShortDescription}";
123                 }
124 
125 // return the concatenated as a string
126 
127                 return itemList;
128             }
129         }
130     }
131 }
```

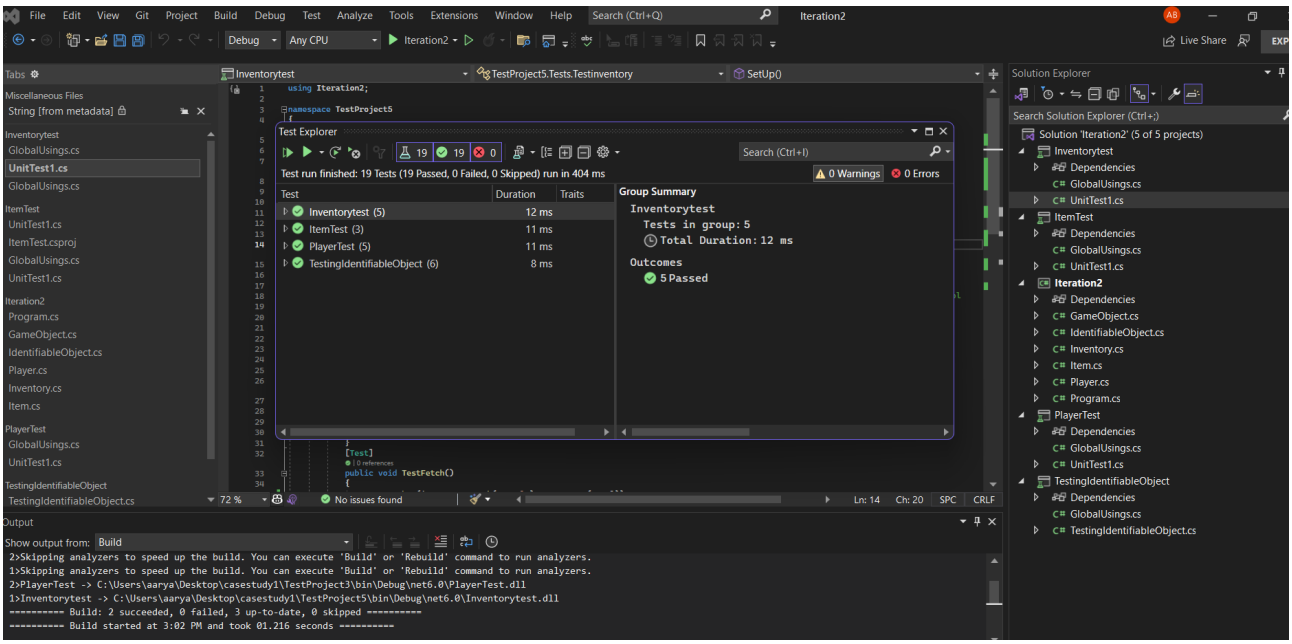
```
using Iteration2;

namespace TestProject5
{
    public class Tests
    {
        [TestFixture]
        public class Testinventory
        {
            Inventory inventory;
            Item stool;
            Item pistol;
            [SetUp]
            public void SetUp()
            {
                inventory = new Inventory();
                pistol = new Item(new string[] { "pistol" }, "50cal pistol", "This is a 50cal pistol");
                stool = new Item(new string[] { "stool" }, "50cm stool", "This is a stool");

                inventory.Put(stool);
                inventory.Put(pistol);
            }
            [Test]
            public void TestHasItem()
            {
                Assert.IsTrue(inventory.HasItem("stool"));
                Assert.IsTrue(inventory.HasItem("pistol"));
            }
            [Test]
            public void TestFetch()
            {
                Assert.That(inventory.Fetch("stool"), Is.SameAs(stool));
                Assert.That(inventory.HasItem("stool"), Is.True);

            }
            [Test]
            public void TestTake()
            {
                Assert.That(inventory.Take("stool"), Is.SameAs(stool));
                Assert.That(inventory.HasItem("stool"), Is.False);
            }
            [Test]
            public void TestItemList()
            {
                //Assert.That(inventory.ItemList, Is.EqualTo("a 50cm stool (stool)\na 50cal pistol (pistol)\n"));
                Assert.IsTrue(inventory.ItemList.Contains("stool"));
                Assert.IsTrue(inventory.ItemList.Contains("pistol"));
            }
            [Test]
            public void TestNoItem()
```

```
51              {
52                  Assert.That(inventory.HasItem("katana"), Is.False);
53              }
54          }
55      }
56  }
```


Test Explorer
Test run finished: 19 Tests (19 Passed, 0 Failed, 0 Skipped) run in 404 ms
Inventorytest (5) 12 ms
ItemTest (3) 11 ms
PlayerTest (5) 11 ms
TestingIdentifiableObject (6) 8 ms
Group Summary
Inventorytest
Tests in group: 5
Total Duration: 12 ms
Outcomes
5 Passed
Output
Show output from: Build
2>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
2>PlayerTest -> C:\Users\aarya\Desktop\casestudy1\TestProject3\bin\Debug\net6.0\PlayerTest.dll
1>Inventorytest -> C:\Users\aarya\Desktop\casestudy1\TestProject5\bin\Debug\net6.0\Inventorytest.dll
========== Build: 2 succeeded, 0 failed, 3 up-to-date, 0 skipped ==========
========== Build started at 3:02 PM and took 01.216 seconds ==========