4.2 iteration 2
Source code
IdentifiableObject.cs

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace SwinAdventure3

{

  public class Identifiable_Object

  {

    private List<string> _identifiers; //its a private class of List that contains a key type and value to store

    public Identifiable_Object(string[] idents) //we set our class here as public which can be used to be called for the other test function later

    //parameter idents, idents.Length refers to the length of the array (number of element) in the (idents array)

    {

      _identifiers = new List<string>();  //now we call a new object to place new list

      int i = 0; //increment

      while (i < idents.Length) //if idents kengths is lower than i perform increment

      {

        AddIdentifier(idents[i]);

        i++; //+1,+1

      }

    }

    public bool AreYou(string id) //bool is like decimal, AreYou is store for string,id

    {
```

```
        return _identifiers.Contains(id.ToLower()); //
    }

    public string FirstId //return first id
    {
      get
      {
        if (_identifiers.Count > 0) //more than 0 then return first identifiers
        {
          return _identifiers[0]; //call first ID
        }
        else
        {
          return ""; //call null
        }
      }
    }

    public void AddIdentifier(string id)
    {
      _identifiers.Add(id.ToLower()); //to store a third identifiers
    }
  }
}
```

Gameobject.cs

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

```
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace SwinAdventure3
{
    public abstract class GameObject : Identifiable_Object
    {
        private string _description, _name;

        public GameObject(string[] idents) : base(idents)
        {
        }

        public GameObject(string[] idents, string name, string desc) : base(idents)
        {
            _name = name;
            _description = desc;

        }
        public string Name
        {
            get { return _name; }

        }
        virtual public string FullDescription
        {
```

```
        get
        {
            return _description;
        }
    }
    public string ShortDescription
    {
        get
        {
            return $"{_name} ({FirstId})";


        }



    }
  }
}
```

Item.cs

```
using SwinAdventure3;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace SwinAdventure3
{
   public class Item : GameObject
   {
      public Item(string[] idents, string name, string desc) : base(idents, name, desc)
//initialise the item lists
      {


      }
   }
}
```

Player.cs

```
using SwinAdventure3;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace SwinAdventure3
{
   public class Player : GameObject
   {
      private Inventory _inventory;
      public Player(string name, string desc) : base(new string[] { "Me", "Inventory " },
name, desc) //overide new info for name and description
      {
         _inventory = new 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; //display our name is carrying itemlist which it varries between
total list length
        }
    }
    public Inventory Inventory
    {
        get => _inventory;
    }


}
```

```
}
```

Inventory.cs

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace SwinAdventure3

{

   public class Inventory

   {

      private List<Item> _items;



      public Inventory()

      {

         _items = new List<Item>(); //call and intialize the list to be executed by the Item.cs

      }


      public bool HasItem(String id)

      {

         foreach (Item i in _items)

         {

            if (i.AreYou(id))

            {

               return true;
```

```
        }
    }
    return false;
}
public void Put(Item itm) //add an item
{

    _items.Add(itm);

}
public Item take(String id) //remove and item
{
    Item takeItem = this.Fetch(id);

    if (takeItem != null)
    {
        // Now it's safe to remove the item outside of the iteration
        _items.Remove(takeItem);
        return takeItem;
    }
    else
    {
        // Return null if the item is not found
        Console.WriteLine("Item not found.");
        return null;
    }
```

```
}



public Item Fetch(String id) //get the item
{
   foreach (Item i in _items)
   {
      if (i.AreYou(id))
      {
         return i;
      }
   }
   return null;
}



public string ItemList
{
   get
   {
      string Listitm = "";
      foreach (Item i in _items)
      {
         Listitm = Listitm + i.ShortDescription;
      }
      return Listitm;
   }
```

```
    }
  }
}
```

UNIT TESTING
TestPlayer.cs

```
using SwinAdventure3;

namespace IdentifiableObjectTesting
{
  public class TestPlayer
  {
    Player player = new Player("Anna", "A wizard");
    Item Gun = new Item(new string[] { "Gun" }, "a gun", "this is a Gun");
    Item Katana = new Item(new string[] { "Katana" }, "a Katana", "this is a katana");




    [SetUp]
    public void Setup()
    {

    }
    [Test]
    public void IdentifiablePlayer()
    {
```

```
    Assert.IsFalse(player.AreYou("Me") && player.AreYou("Inventory"));
}
[Test]
public void LocateItem()
{
    var result = false;
    player.Inventory.Put(Gun);
    var Itemlocate = player.Locate("Gun");
    if (Gun == Itemlocate)
    {
        result = true;
    }
    Assert.IsTrue(result);
}
[Test]
public void locateItself()
{
    var Me = player.Locate("Me");
    var Invent = player.Locate("Inventory");
    var result = false;

    if (Me == player || Invent == player)
    {
        result = true;
    }
    Assert.IsTrue(result);
    ;
```

```
    }
    [Test]
    public void locateNon()
    {
        var me = player.Locate("Me");
        Assert.AreEqual(me, player);

    }
    [Test]
    public void FullDescription()
    {
        player.Inventory.Put(Gun);
        player.Inventory.Put(Katana);
        string ExpectedOutput = "You are Anna, A wizard.\nYou are carrying\na gun (gun)a Katana (katana)";
        Assert.AreEqual(player.FullDescription, ExpectedOutput);

    }
  }
}
```

TestItem.cs

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

```
using SwinAdventure3;

namespace IdentifiableObjectTesting
{
   public class TestItem
   {
      Item Gun = new Item(new string[] { "Gun" }, "a Gun", "this is a Gun");
      Item Katana = new Item(new string[] { "Katana" }, "a Katana", "this is a katana");



      [SetUp]
      public void Setup()
      {
      }
      [Test]
      public void TestItemIdentifiable()
      {
         var result = Gun.AreYou("Gun");
         Assert.IsTrue(result);


         var result1 = Katana.AreYou("Katana");
         Assert.IsTrue(result1);



      }
[Test]
      public void ShortDescription()
      {
```

```
        Assert.AreEqual(Gun.ShortDescription, "This is a legendary gun");
        Assert.AreNotEqual(Katana.ShortDescription, "This is a legendary Katana");
    }
[Test]
    public void FullDescription()
    {
        Assert.AreEqual(Gun.FullDescription, "This gun is crafted with the finest Gun powder in the market");
        Assert.AreNotEqual(Katana.FullDescription, "This Katana is forged by the finest swordsmiths");
    }
  }
}
```

TestInv.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;
using SwinAdventure3;

namespace IdentifiableObjectTestingInv
{
  public class TestInventory
  {
    Item Gun = new Item(new string[] { "Gun" }, "a Gun", "this is a Gun");
    Item Katana = new Item(new string[] { "Katana" }, "a Katana", "this is a katana");
```

```
[SetUp]

public void Setup()
{

}
[Test]
public void TestFindThem()
{
   Inventory i = new Inventory();
   i.Put(Gun);
   Assert.IsTrue(i.HasItem(Gun.FirstId)); //check first object "gun" if have then is true

}
[Test]
public void TestNotFindItem()
{
   Inventory i = new Inventory();
   i.Put(Gun);
   Assert.IsFalse(i.HasItem(Katana.FirstId)); //check gun to compare with chosen object in this case is "katana" then IsFalse
}
[Test]
public void TestFetchItem()
{
   Inventory i = new Inventory();
```

```
        i.Put(Gun);
        Item fetchItem = i.Fetch(Gun.FirstId);

        Assert.IsTrue(fetchItem == Gun);
        Assert.IsTrue(i.HasItem(Gun.FirstId));
    }
    [Test]
    public void TestTakenItem()
    {
        Inventory i = new Inventory();
        i.Put(Gun);
        i.take(Gun.FirstId);
        Assert.IsFalse(i.HasItem(Gun.FirstId));
    }
    [Test]
    public void ItemList()
    {
        Inventory i = new Inventory();
        i.Put(Gun);
        i.Put(Katana);
        Assert.IsTrue(i.HasItem(Gun.FirstId));
        Assert.IsTrue(i.HasItem(Katana.FirstId));

        String expctOutput = "a Gun (gun)" + "a Katana (katana)";
        Assert.That(i.ItemList, Is.EqualTo(expctOutput));
    }
}
```

```
}
```

objecttesting.cs

```
using NUnit.Framework;
using SwinAdventure3;



namespace IdentifiableObjectTesting
{
    public class Objecttesting
    {
        private Identifiable_Object _testObject;
        private string _teststring;
        private string[] _teststringArray;

        private Identifiable_Object _testObject_emp;
        private string _teststring_emp;
        private string[] _teststringArray_emp;




        [SetUp]
        public void Setup()
        {
            _teststring = "Anna";
            string[] _teststringArray = new string[] { "Anna", "Bryan" };
            _testObject = new Identifiable_Object(_teststringArray);
            _testObject.AddIdentifier(_teststring);
```

```
    _teststring_emp = "";
    _teststringArray_emp = new string[] { };
    _testObject_emp = new Identifiable_Object(_teststringArray_emp);
    _testObject_emp.AddIdentifier(_teststring_emp);

}

[Test]
public void TestAreYou()
{

    Assert.IsTrue(_testObject.AreYou(_teststring));

}

[Test]
public void TestNotAreYou()
{
    Assert.IsFalse(_testObject.AreYou("Jack"));

}

[Test]
public void TestCaseSensitive()
{
    Assert.IsTrue(_testObject.AreYou("Anna"));
}
```

```
    [Test]
    public void TestFirstId()
    {
      Assert.AreEqual("anna", _testObject.FirstId);
      Assert.AreNotEqual("Jack", _testObject.FirstId);
    }

    [Test]
    public void TestFirstIdWithNoId()
    {

      Assert.AreEqual("", _testObject_emp.FirstId);  //assert that firstId is equal to ""(null) if no id
    }

    [Test]
    public void TestAddId()
    {
      _testObject.AddIdentifier("Max");
      _testObject.AddIdentifier("Andrew");
      Assert.IsTrue(_testObject.AreYou("Max"));
      Assert.IsTrue(_testObject.AreYou("Andrew"));
    }
  }
}
```

OUTPUT

*ignore the identifiableObjectTestingBag I was testing for iteration 3 when I noticed major errors on my iterations 2 code