School of Science, Computing and Engineering Technologies

Object-Oriented Programming

Week 3: NUnit Test with Identifiable Objects and Items

Overview

In this week, there are **two assessable tasks** 3.1 and 3.2. **Each task contributes 2%** to your final grade. Noting that you need to complete these tasks before coming to your allocated lab. In the lab, there will be verification tasks and short interview to verify your understanding.

Object-oriented programming makes best sense with larger programs including many consecutive development iterations. The case study we provide here is SwinAdventure game. The game will be your opportunity to create a larger program and better understand how the OOP can make it easier to create complex software solutions.

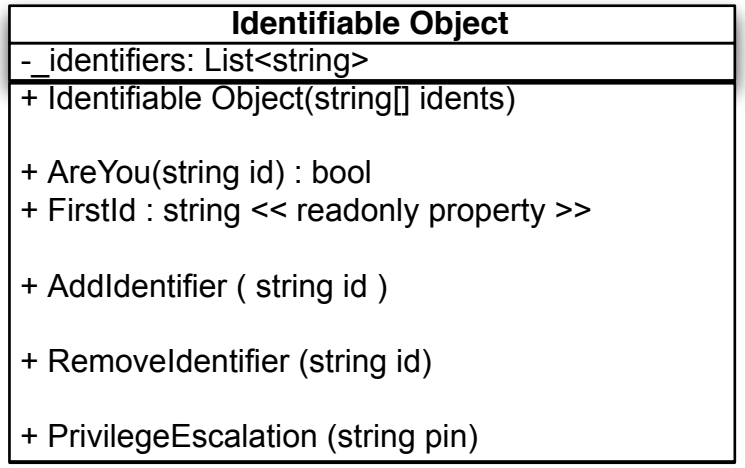
Purposes Task 3.1	Practice interpreting UML class diagrams and writing unit tests
(pages 2-4)	Understand the case study program and implement iteration 1. **The task contains personalized requirements.**
Task 3.2 (pages 5-6)	Learn to apply object-oriented programming techniques related to the concept of abstraction and conduct the NUnit test. Understand the case study program and implement iteration 2. **The task contains personalized requirements**

Note: If you are unable to complete theses tasks, you will struggle in this unit. Your tutor may be able assist you with suggestions on how increase your knowledge. You can also participate in HelpDesk sessions

Task 3.1. Instructions - Iteration 1: Identifiable Object

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

2. In this Iteration 1 you will create an Identifiable Object which will become a foundation for many of the objects in the Swin-Adventure. Identifiable objects have a list of identifiers, and know if they are identified by a certain identifier.
3. The UML diagram for the Identifiable Object follows, and the unit tests to create are shown on the following page.



The player needs to be able to "identify" a number of things in the game. This includes commands the user will perform, items they will interact with, locations, paths, etc. The **Identifiable Object** role was created to encapsulate this functionality.

- Identifiable Object
  - The **constructor** adds identifiers to the Identifiable Object from the passed in array.
  - **Are You** checks if the passed in identifier is in the _identifiers
  - **First Id** returns the first identifier from _identifiers, or an empty string if the object has identifiers
  - **Add Identifier** converts the identifier to lower case and stores it in _identifiers
  - **Privilege Escalation** checks if the inputed pin is the same as the last 4 digits of your student ID. If yes, replace the first identifer from the _identifiers by your tutorial ID. Otherwise, do nothing.
- **Remove Identifier** converts the identifier to lower case and remove it in _identifiers if exists

Example Identifiable Object:

```
IdentifiableObject id =
new IdentifiableObject( new string[] { "007", "James", "Bond"} );
```

Use the following unit tests to create the Identifiable Object class and ensure that it is working successfully, using your details to identify you as a player in the game.

This means we want you to use your personal details such as your student number, your first name, and your family name rather than “007”, “James”, and “Bond”.

Identifiable Object Unit Tests		
Test Number	**Test Case Label**	**Test Description**
1	Test Are You	Check that it responds "True" to the "Are You" message where the request matches one of the object's identifiers. eg. An Identifiable Object with identifiers <<Your student ID>>, <<“Your name>> and <<Your family name>> can be identified. by (calling Are You) “<<Your student ID>>” and “<<“Your name>>”. If you wish you may use your tutor’s surname (family name).
2	Test Not Are You	Check that it responds "False" to the "Are You" message where the request does not match one of the object's identifiers. To create a mismatch for your <<Student ID>> change any zeros (0) to the letter “O”. eg. An Identifiable Object for a band with identifiers “John”, “Paul”, “George” and “Ringo” WOULD NOT be identified by (calling Are You) “Jagger” or “Swift”.
3	Test Case Sensitive	Check that it responds "True" to the "Are You" message where the request matches one of the object's identifiers where there is a mismatch in case. eg. An Identifiable Object initialised with identifiers “James” and “Bond” would be identified by (calling Are You) “JAMES” and “bOnD”.

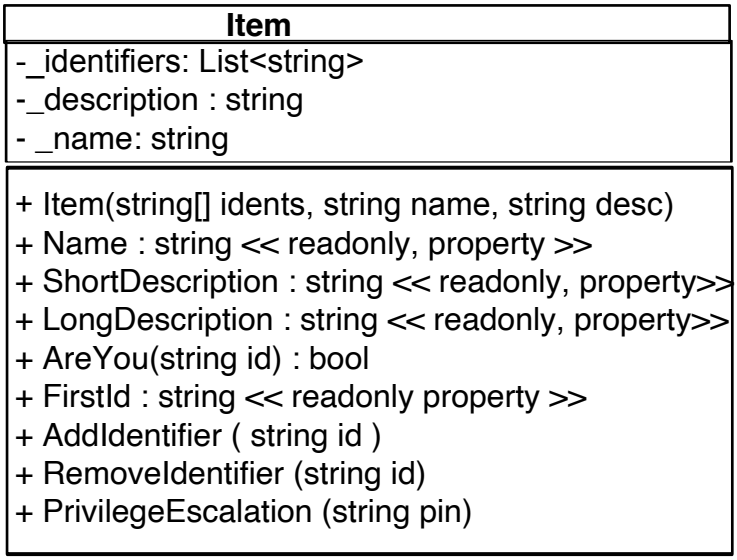
Identifiable Object Unit Tests (continued)		
Test Number	**Test Case Label**	**Test Description**
4	Test First ID	Check that the first id returns the first identifier in the list of identifiers.
5	Test First ID With No IDs	Check that an empty string is returned if there are no identifiers in the list of identifiers. e.g., An Identifiable Object with no identifiers has “” as its FirstID.
6	Test Add ID	Check that you can add identifiers to the object. eg. An Identifiable Object created with identifiers “Seekers” and “Athol”, “Keith”, “Bruce” can have "Mary" added and then be identified by (calling Are You) with “Seekers”,“Keith”, and "Mary".
7	Test Privilege Escalation	Check that you can escalate your lab instance number or ID to be the first id. eg. An Identifiable Object with identifiers “007” and “James” can be replaced by newly updated identifiers "XXX" and "James", where "XXX" would be your tutorial’s ID. This can be verified by using the Test First ID case.

Note: At this point there will not be a "program" as such, just a set of unit tests that help demonstrate that your solution is moving towards completion.
You should have a game project and a test project co-locate under the same C# VS Code solution.

Once your tests are working correctly save and backup your projects

Task 3.2. Instructions - Iteration 2 - Items

The goal of this iteration will be to create the Item class. This class presents the "Items" that a player can interact with in the game. In addition, you may think of the player has an inventory and the inventory has a collection of items. We will create the player and the inventory classes in next iterations.



- Item
  - _identifiers: The list of identifiers in the item
  - _name: The name of the item.
  - _description - a longer textual description of the item
  - Short Description - returns a short description made up of the name and the first id of the game object. This is used when referring to an object, rather than directly examin-ing the object.
  - Long Description - By default this is just the description.
  - Are You checks if the passed in identifier is in the _identifiers
  - First Id returns the first identifier from _identifiers, or an empty string if the object has identifiers
  - Add Identifier converts the identifier to lower case and stores it in _identifiers
  - Remove Identifier converts the identifier to lower case and remove it in _identifiers if exists
  - Privilege Escalation checks if the inputed pin
    is the same as the last 4 digits of your student ID.
    If yes, replace the first identifer from the _identifiers
    by your tutorial/lab ID. Otherwise, do nothing.

Item Unit Tests	
Test Item is Identifiable	The item responds correctly to "Are You" requests based on the identifiers it is created with.
Test Short Description	The object's short description returns the string "a name (first id)" eg: a bronze sword (sword)
Test Full Description	Returns the item's description.
Test Privilege Escalarion	The item returns correctly the first ID as your tutorial ID if the inputed pin matches the last 4 digits of your student ID.

Implement above test cases.

Note: At this point there will not be a "program" as such, just a set of unit tests that help demonstrate that your solution is moving towards completion.

You can implement the Iteration 2 as new .cs files co-located in the same projects that you created in Iteration 1.

When you arrive at your lab, you will receive the verification tasks.
See you very soon.