**Task 3.1 – IdentifiableObject**

**IdentifiableObject.cs**

using System;

using System.Collections.Generic;

namespace SwinAdventure

{

public class IdentifiableObject

{

//Collection class to store identifiers

private List<string> \_identifiers;

// Constructor: Initializes the object with an array of identifiers.

public IdentifiableObject(string[] idents)

{

\_identifiers = new List<string>();

foreach (string id in idents)

{

AddIdentifier(id);

}

}

// Checks if a given 'id' is in the list (case-insensitive).

public bool AreYou(string id)

{

return \_identifiers.Contains(id.ToLower());

}

// Add FirstId property

// Gets the first identifier, or an empty string if the list is empty.

public string FirstId

{

get

{

if (\_identifiers.Count > 0)

{

return \_identifiers[0];

}

else

{

return "";

}

}

}

// AddIdentifier Method

// Adds a new identifier to the list in lowercase.

public void AddIdentifier(string id)

{

\_identifiers.Add(id.ToLower());

}

// RemoveIdentifier Method

// Removes an identifier from the list.

public void RemoveIdentifier(string id)

{

\_identifiers.Remove(id.ToLower());

}

// PrivilegeEscalation Method

// Replaces the first ID if the correct PIN is provided.

public void PrivilegeEscalation(string pin)

{

if (pin == "4881" && \_identifiers.Count > 0)

{

\_identifiers[0] = "TUTE01";

}

}

}

}

**Task 3.1 – Item**

**Item.cs**

using System;

using System.Collections.Generic;

namespace SwinAdventure

{

public class Item

{

//Collection class to store identifiers

private List<string> \_identifiers;

// Add private fields

private string \_name;

private string \_description;

// Constructor for the Item.

public Item(string[] idents, string name, string desc)

{

\_identifiers = new List<string>(idents);

\_name = name;

\_description = desc;

}

public bool AreYou(string id)

{

return \_identifiers.Contains(id.ToLower());

}

public void PrivilegeEscalation(string pin)

{

\_identifiers[0] = "TUTE01";

}

public string FirstId

{

get { return \_identifiers[0]; }

}

// A read-only property to get the item's name.

public string Name

{

get { return \_name; }

}

// A read-only property that formats a short description.

public string ShortDescription

{

get { return "a " + \_name + " (" + \_identifiers[0] + ")"; }

}

// A read-only property to get the item's full description.

public string LongDescription

{

get { return \_description; }

}

}

}

**Task 3.2 – IdentifiableObjectTests**

**IdentifiableObjectTests.cs**

using NUnit.Framework;

using SwinAdventure;

namespace SwinAdventure.Tests

{

[TestFixture]

public class IdentifiableObjectTests

{

private IdentifiableObject \_testObject;

private string \_studentID = "105684881";

private string \_firstName = "Min Thu Kyaw";

private string \_familyName = "Khaung";

[SetUp]

public void Setup()

{

// Initialize the test object with sample identifiers.

\_testObject = new IdentifiableObject(new string[] { \_studentID, \_firstName, \_familyName });

}

[Test]

public void TestAreYou()

{

// Test that it responds True when identifier matches

Assert.That(\_testObject.AreYou(\_studentID), Is.True);

Assert.That(\_testObject.AreYou(\_firstName), Is.True);

Assert.That(\_testObject.AreYou(\_familyName), Is.True);

}

[Test]

public void TestNotAreYou()

{

// Test that it responds False when identifier doesn't match

Assert.That(\_testObject.AreYou("1O5684881"), Is.False);

Assert.That(\_testObject.AreYou("Taaj"), Is.False);

Assert.That(\_testObject.AreYou("Jack"), Is.False);

}

[Test]

public void TestCaseSensitive()

{

// Test that matching is case insensitive

Assert.That(\_testObject.AreYou(\_firstName.ToUpper()), Is.True);

Assert.That(\_testObject.AreYou(\_familyName.ToLower()), Is.True);

Assert.That(\_testObject.AreYou("min Thu kyaW"), Is.True);

Assert.That(\_testObject.AreYou("MiN ThU KyAW"), Is.True);

}

[Test]

public void TestFirstID()

{

// Test that first id returns the first identifier

Assert.That(\_testObject.FirstId, Is.EqualTo(\_studentID));

}

[Test]

public void TestFirstIDWithNoIDs()

{

// Test empty string is returned when no identifiers

IdentifiableObject emptyObject = new IdentifiableObject(new string[] { });

Assert.That(emptyObject.FirstId, Is.EqualTo(""));

}

[Test]

public void TestAddID()

{

// Test that identifiers can be added

\_testObject.AddIdentifier("TestID");

Assert.That(\_testObject.AreYou("TestID"), Is.True);

Assert.That(\_testObject.AreYou("testid"), Is.True);

}

[Test]

public void TestPrivilegeEscalation()

{

// Test privilege escalation with correct PIN

\_testObject.PrivilegeEscalation("4881");

Assert.That(\_testObject.FirstId, Is.EqualTo("TUTE01"));

// Test with wrong PIN

IdentifiableObject testObject2 = new IdentifiableObject(new string[] { "test", "object" });

testObject2.PrivilegeEscalation("1234");

Assert.That(testObject2.FirstId, Is.EqualTo("test")); // Should remain unchanged

}

}

}

**Task 3.2 - ItemTests**

**ItemTests.cs**

using NUnit.Framework;

using SwinAdventure;

namespace SwinAdventure.Tests

{

[TestFixture]

public class ItemTests

{

private Item \_testItem;

[SetUp]

public void Setup()

{

// Initialize the test item with sample identifiers.

\_testItem = new Item(new string[] { "sword", "bronze sword" }, "bronze sword", "A short sword cast from bronze");

}

[Test]

public void TestItemIsIdentifiable()

{

// Test that item responds correctly to AreYou requests

Assert.That(\_testItem.AreYou("sword"), Is.True);

Assert.That(\_testItem.AreYou("bronze sword"), Is.True);

Assert.That(\_testItem.AreYou("SWORD"), Is.True);

Assert.That(\_testItem.AreYou("axe"), Is.False);

}

[Test]

public void TestShortDescription()

{

// Test short description format: "a name (first id)"

Assert.That(\_testItem.ShortDescription, Is.EqualTo("a bronze sword (sword)"));

}

[Test]

public void TestFullDescription()

{

// Test that full description returns the item's description

Assert.That(\_testItem.LongDescription, Is.EqualTo("A short sword cast from bronze"));

}

[Test]

public void TestPrivilegeEscalation()

{

// Test privilege escalation with correct PIN

\_testItem.PrivilegeEscalation("4881");

Assert.That(\_testItem.FirstId, Is.EqualTo("TUTE01"));

}

}

}