Лабораторная работа № 2 Модульное тестирование

Цели:

1. Обеспечение корректности отдельных модулей: Модульные тесты позволяют проверить, что каждая отдельная часть программы (например, класс или функция) работает правильно и соответствует требованиям.

2. Выявление ошибок на ранних этапах: Тесты помогают обнаруживать ошибки и недочеты в коде еще на стадии разработки, что упрощает их исправление и снижает вероятность появления ошибок в готовом продукте.

3. Снижение риска регрессий: После внесения изменений в код модульные тесты обеспечивают уверенность в том, что существующая функциональность не была нарушена.

4. Ускорение разработки: Наличие полной базы тестов позволяет разработчикам работать быстрее, зная, что их изменения не ломают уже проверенные части кода.

5. Повышение качества кода: Применение модульных тестов способствует созданию более структурированного, понятного и поддерживаемого кода.

6. Автоматизация тестирования: Модульные тесты можно автоматизировать, что экономит время и ресурсы, особенно при частых изменениях кода.

Задание:

1. Продолжаем работу с проектом созданным на тему из лабораторной №1.
2. Разработать тесты для каждого модуля.

Пример:

Модульные тесты для программы управления товарами в магазине с использованием MSTest:

1. Тесты для класса `Product`

```csharp
using Microsoft.VisualStudio.TestTools.UnitTesting;
```

```
namespace ProductManager.Tests
{
    [TestClass]
    public class ProductTests
    {
        [TestMethod]
        public void IsAvailable_WhenQuantityGreaterThanZero_ReturnsTrue()
        {
            // Arrange
            var product = new Product("Test", 100, 5);

            // Act
            var result = product.IsAvailable();

            // Assert
            Assert.IsTrue(result);
        }

        [TestMethod]
        public void IsAvailable_WhenQuantityEqualsZero_ReturnsFalse()
        {
            // Arrange
            var product = new Product("Test", 100, 0);

            // Act
            var result = product.IsAvailable();

            // Assert
            Assert.IsFalse(result);
        }

        [TestMethod]
        public void ToString_ReturnsCorrectFormat()
        {
            // Arrange
            var product = new Product("Test", 100, 5);

            // Act
            var result = product.ToString();

            // Assert
            StringAssert.Contains(result, "Товар: Test");
            StringAssert.Contains(result, "Количество: 5");
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void Constructor_WhenPriceIsNegative_ThrowsArgumentException()
```

```
        {
            // Act
            new Product("Test", -100, 5);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void Constructor_WhenQuantityIsNegative_ThrowsArgumentException()
        {
            // Act
            new Product("Test", 100, -5);
        }
    }
}
```

2. Тесты для класса `MainWindow` (логика форм)

```csharp
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Windows.Forms;

namespace ProductManager.Tests
{
    [TestClass]
    public class MainWindowTests
    {
        private MainWindow _mainWindow;
        private List<Product> _products;

        [TestInitialize]
        public void SetUp()
        {
            _mainWindow = new MainWindow();
            _products = new List<Product>();
        }

        [TestMethod]
        public void AddButton_Click_WhenValidData_AddsProduct()
        {
            // Arrange
            _mainWindow.nameTextBox.Text = "Test";
            _mainWindow.numericUpDown1.Value = 100;
            _mainWindow.numericUpDown2.Value = 5;

            // Act
            _mainWindow.addButton_Click(null, EventArgs.Empty);
```

```
            // Assert
            Assert.AreEqual(1, _mainWindow.products.Count);
        }

        [TestMethod]
        public void AddButton_Click_WhenNegativePrice_ShowsError()
        {
            // Arrange
            _mainWindow.nameTextBox.Text = "Test";
            _mainWindow.numericUpDown1.Value = -100;
            _mainWindow.numericUpDown2.Value = 5;

            // Act
            _mainWindow.addButton_Click(null, EventArgs.Empty);

            // Assert
            Assert.AreEqual(0, _mainWindow.products.Count);
        }

        [TestMethod]
        public void RemoveButton_Click_WhenItemSelected_RemovesProduct()
        {
            // Arrange
            _mainWindow.products.Add(new Product("Test", 100, 5));
            _mainWindow.productsListBox.SetSelected(0, true);

            // Act
            _mainWindow.removeButton_Click(null, EventArgs.Empty);

            // Assert
            Assert.AreEqual(0, _mainWindow.products.Count);
        }

        [TestMethod]
        public void CheckButton_Click_WhenItemSelected_ChecksAvailability()
        {
            // Arrange
            _mainWindow.products.Add(new Product("Test", 100, 5));
            _mainWindow.productsListBox.SetSelected(0, true);

            // Act
            _mainWindow.checkButton_Click(null, EventArgs.Empty);

            // Assert
            // Проверить, что показано соответствующее сообщение
        }
    }
}
```

```

#### 3. Тесты для проверки граничных значений

```csharp
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace ProductManager.Tests
{
    [TestClass]
    public class BoundaryConditionTests
    {
        [TestMethod]
        public void PriceZero_IsNotAllowed()
        {
            // Act and Assert
            Assert.Throws<ArgumentException>(() => new Product("Test", 0, 5));
        }

        [TestMethod]
        public void QuantityZero_IsAllowedButNotAvailable()
        {
            // Arrange
            var product = new Product("Test", 100, 0);

            // Act and Assert
            Assert.IsFalse(product.IsAvailable());
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void NegativePrice_IsNotAllowed()
        {
            // Act
            new Product("Test", -100, 5);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void NegativeQuantity_IsNotAllowed()
        {
            // Act
            new Product("Test", 100, -5);
        }
    }
}
```
```

4. Тесты для UI элементов

```csharp
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Windows.Forms;

namespace ProductManager.Tests
{
    [TestClass]
    public class UITests
    {
        private MainWindow _mainWindow;

        [TestInitialize]
        public void SetUp()
        {
            _mainWindow = new MainWindow();
        }

        [TestMethod]
        public void NameTextBox_IsVisibleAndEnabled()
        {
            // Assert
            Assert.IsTrue(_mainWindow.nameTextBox.Visible);
            Assert.IsTrue(_mainWindow.nameTextBox.Enabled);
        }

        [TestMethod]
        public void AddButton_IsVisibleAndEnabled()
        {
            // Assert
            Assert.IsTrue(_mainWindow.addButton.Visible);
            Assert.IsTrue(_mainWindow.addButton.Enabled);
        }

        [TestMethod]
        public void ProductsListBox_IsVisibleAndEnabled()
        {
            // Assert
            Assert.IsTrue(_mainWindow.productsListBox.Visible);
            Assert.IsTrue(_mainWindow.productsListBox.Enabled);
        }
    }
}
```

Объяснение:
1. Тесты для класса `Product`:

- Проверяют корректность методов `IsAvailable()` и `ToString()`.
- Проверяют, что конструктор выбрасывает исключение при отрицательных значениях цены или количества.

2. Тесты для класса `MainWindow`:
- Проверяют обработку кликов на кнопки добавления, удаления и проверки.
- Проверяют, что при вводе неверных данных отображается сообщение об ошибке.

3. Тесты для проверки граничных значений:
- Проверяют, что цена и количество не могут быть отрицательными или нулевыми.

4. Тесты для UI элементов:
- Проверяют, что элементы управления (TextBox, Button, ListBox) видимы и доступны.

Эти тесты покрывают основные функции программы и помогают убедиться, что она работает корректно.

Варианты

1. Управление банковским счётом
2. Конвертер валют
3. Управление задачами
4. Календарь событий
5. Управление книгами в библиотеке
6. Управление заметками
7. Управление контактами
8. Управление покупками
9. Управление задачами с приоритетом
10. Управление продажами
11. Управление инвентарём
12. Управление заказами
13. Управление сотрудниками
14. Управление проектами
15. Управление клиентами
16. Управление доставкой
17. Управление отчётами
18. Управление резервированием
19. Управление бюджетом
20. Управление файлами и папками
21. Управление системой безопасности данных
22. Управление здоровьем
23. Управление музыкальной коллекцией
24. Управление путешествиями
25. Управление фотографиями

26. Управление рецептами и планированием меню
27. Управление обучением и курсами
28. Управление резюме и поиском работы
29. Управление встречами и мероприятиями
30. Управление техникой и оборудованием
31. Управление автомобилем и его обслуживанием
32. Управление энергопотреблением
33. Управление личными целями и задачами
34. Управление коллекцией игр
35. Управление доставкой