МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ  
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и  
технологий

**Отчёт  
по лабораторной работе №6  
«Unit-тесты»**  
по дисциплине: «Конструирование программного обеспечения»

Выполнили: ст. гр. 10701121

Васенков А.Д.

Иванов М.Ю.

Мажейко Е.Д.

Принял: Станкевич С.Н.

Минск 2023

**Лабораторная работа №6**

“Unit-тесты”

Цель: Научиться грамотно анализировать предметную область и с помощью абстракции выделять существенные детали, на базе которых в дальнейшем проектируются классы и объекты будущей программной системы согласно методологии ООП, а также практически закрепить данные навыки при решении соответствующих задач (бизнес проблем).

1. Сведения о Unit-тестах.

Модули взаимодействуют и обеспечивают работу программы или приложения. Чтобы проверить, правильно ли написан модуль, проводят юнит-тесты, или модульное тестирование, — проверку не всего приложения, а одного модуля. Пример юнит-теста — проверка функции подсчёта общей стоимости заказа.

Модульное тестирование проводят сразу после написания кода. Проверить работу кнопки в готовом приложении не получится, потому что на неё уже влияют другие модули. Главная причина написания юнит-тестов — тестирование отдельных модулей. Поскольку каждый модуль пишется отдельно, тестировать его тоже можно изолированно, без связки с другими. Получается простая схема: программист написал модуль → протестировал его → продолжил разработку для связи с другими модулями и других тестов.

Если пропустить этап юнит-тестирования, в следующий раз не получится понять, что именно вызвало ошибку: какой-то из модулей или неправильно настроенная интеграция между ними. Придётся разбираться, тратить время и всё равно тестировать отдельные юниты.

Для примера представим автомобиль. Его «юниты» — это двигатель, подача бензина, зажигание. Можно проверить их по отдельности и ещё до сборки увидеть поломки и починить их. А можно собрать автомобиль, не протестировав юниты, — и он не поедет. Придётся всё разбирать и проверять каждую деталь.

1. Требования

1) Необходимо спроектировать и реализовать UML-диаграмму взаимодействия классов и объектов разрабатываемой программной системы с отображением всех связей (отношений) между классами и объектами.

2) При проектировании и разработке системы необходимо полностью использовать своё объектно-ориентированное воображение и по максимум использовать возможности, которые предоставляет язык программирования C# для реализации ООП-методологии.

3) Основные классы системы должны быть самодостаточными, т.е. не зависеть, к примеру, от консоли! Любые типы отношений между классами должны применяться обосновано и лишь тогда, когда это имеет смысл.

4) При выполнения задания необходимо по максимуму пытаться разрабатывать универсальный, масштабируемый, легко поддерживаемый и читаемый код.

5) Также рекомендуется придерживаться Single Responsibility Principle, SRP (принципа единственной ответственности): у каждого пакета, класса или метода должна быть только одна ответственность (цель), т.е. должна быть только одна причина изменить в дальнейшем соответствующий блок кода.

6) Создаваемые классы необходимо грамотно разложить по соответствующим пакетам, которые должны иметь «адекватные» названия и быть вложены в указанные стартовые пакеты: by.bntu.fitr.poisit.nameofstudent.

7) В соответствующих компонентах бизнес-логики необходимо предусмотреть «защиту от дурака».

8) На базе спроектированной программной системы реализуйте простейшее интерактивное консольное приложение. Используйте при реализации архитектурный шаблона проектирования Model-View-Controller, MVC.

9) Программа должна обязательно быть снабжена комментариями, в которых необходимо указать краткое предназначение программы, номер лабораторной работы и её название, версию программы, ФИО разработчиков, название бригады (если есть), номер группы и дату разработки. Исходный текст классов и демонстрационной программы рекомендуется также снабжать поясняющими краткими комментариями.

10) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем.

11) Интерфейс программы и комментарии должны быть на английском языке.

12) Необходимо проверить все тестовые случаи работы основной бизнес-логики программы.

1. Постановка задачи

Составить unit-тесты.

1. Состав проекта

using System.Windows;

namespace Tests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestFood()

{

Knight.Model.Food food\_1 = new Knight.Model.Food("-", "-", 200, 200, 200);

Knight.Model.Food food\_2 = new Knight.Model.Food("-", "-", 100, 100, 100);

food\_1.Use();

MessageBox.Show("s");

Assert.AreEqual(food\_1.Price, food\_2.Price);

Assert.AreEqual(food\_1.Weight, food\_2.Weight);

Assert.AreEqual(food\_1.Satiety, food\_2.Satiety);

}

[TestMethod]

public void TestWeapon()

{

Knight.Model.Weapon weapon\_1 = new Knight.Model.Weapon("-", "-", 200, 200, 200);

Knight.Model.Weapon weapon\_2 = new Knight.Model.Weapon("-", "-", 200, 200, 198);

weapon\_1.Use();

Assert.AreEqual(weapon\_2.Damage, weapon\_1.Damage);

}

[TestMethod]

public void TestCloth()

{

Knight.Model.Cloth cloth\_1 = new Knight.Model.Cloth("-", "-", 200, 200, "Шерсть");

Knight.Model.Cloth cloth\_2 = new Knight.Model.Cloth("-", "-", 200, 200, "Шерсть");

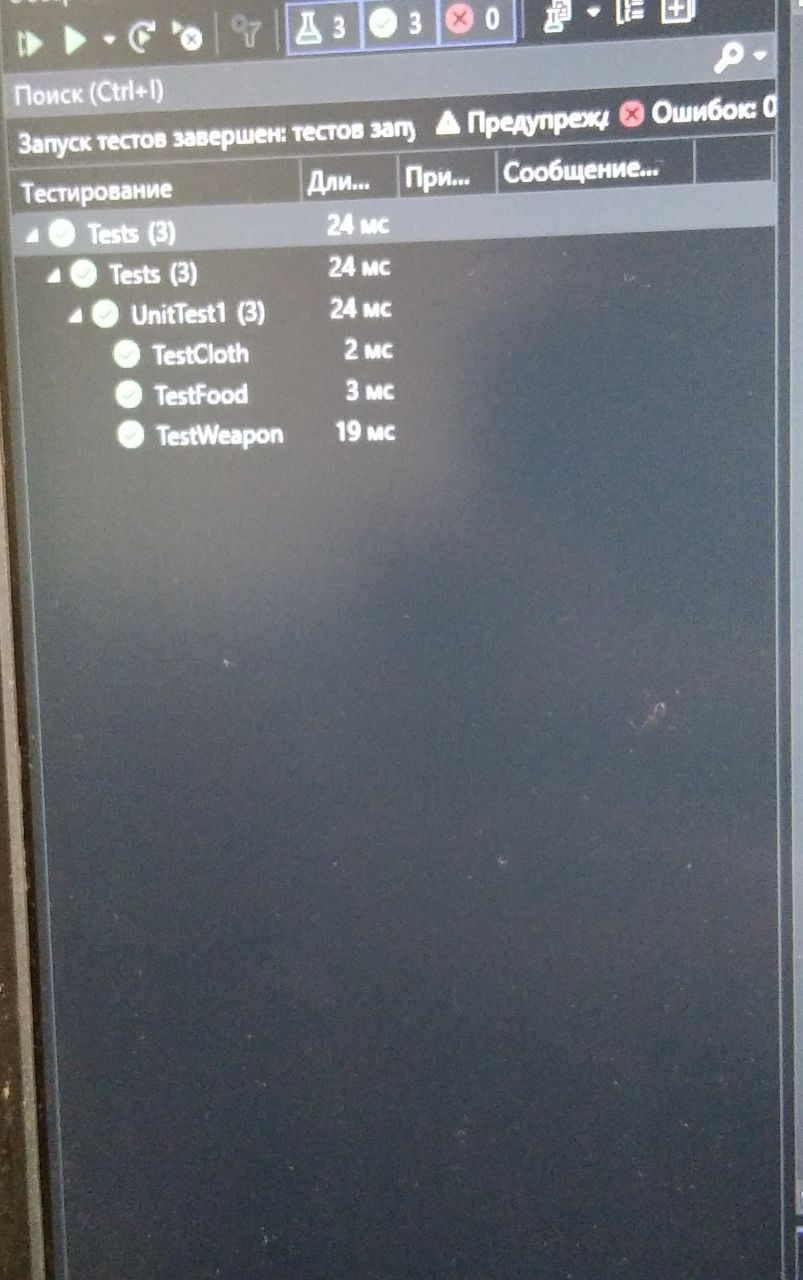
cloth\_1.Use();

Assert.AreEqual(cloth\_2.ToString(), cloth\_1.ToString());

}

}

1. Cкриншоты работы программы



*Скриншот 1. Результат тестов*



*Рисунок 1. UML-диаграмма программы.*

1. Вывод

C# - мощный и популярный язык программирования, который обладает полной поддержкой ООП и предоставляет разработчикам возможности для создания абстракций и объектов, абстракция же, как концепция ООП, помогает скрыть детали реализации объектов и предоставлять только необходимый набор методов и свойств для взаимодействия с ними

1. Использованные источники

<https://en.wikipedia.org/wiki/Object-oriented_programming>

<https://en.wikipedia.org/wiki/C_Sharp_(programming_language)>

https://en.wikipedia.org/wiki/Abstraction\_(computer\_science)

https://chat.openai.com/chat