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 №2  
«ООП на C#. Абстракция»**  
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 №2**

“ООП на C#. Абстракция”

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 C# и ООП.

C# (читается "си шарп") является объектно-ориентированным языком программирования, разработанным в 2000 году компанией Microsoft. Главной целью создания C# было предоставление разработчикам инструментов для создания приложений на платформе .NET.

C# представляет собой смесь различных языков программирования, таких как C++, Java и Delphi. Этот язык программирования имеет множество преимуществ перед другими языками, таких как более безопасное программирование, более простой и удобный синтаксис, поддержка многопоточности, и т.д.

ООП является ключевым аспектом языка C#. В C# используются такие основные концепции ООП, как инкапсуляция, наследование и полиморфизм.

Инкапсуляция - это механизм, который позволяет скрыть детали реализации объекта от других объектов. Например, в C# можно создать класс, который имеет методы и поля, которые могут быть доступны только изнутри класса, но не доступны из других классов.

Наследование - это механизм, который позволяет создавать новый класс на основе уже существующего класса. Например, можно создать класс "автомобиль", а затем создать подклассы "легковой автомобиль" и "грузовой автомобиль", которые будут иметь дополнительные методы и поля.

Полиморфизм - это механизм, который позволяет объектам иметь различное поведение в зависимости от контекста. Например, можно создать метод "печать", который может быть вызван как для строки, так и для числа, и в каждом случае он будет вести себя по-разному.

Концепция ООП была разработана еще в 1960-х годах и атрибутируется Алану Кэю, одному из создателей языка программирования Simula. Он предложил использовать объекты как основные элементы программирования и создал первый объектно-ориентированный язык программирования - Simula.

Абстракция - это концепция объектно-ориентированного программирования, которая позволяет скрыть детали реализации объекта и предоставить только необходимые для работы интерфейсы. Таким образом, абстракция помогает разработчикам сосредоточиться на существенных аспектах объектов и методов, а не на деталях их реализации.

В C# абстракция достигается через использование абстрактных классов и интерфейсов.

Абстрактный класс - это класс, который не может быть использован для создания объектов, но может содержать абстрактные методы и свойства. Абстрактный метод - это метод, который не имеет реализации и должен быть реализован в производных классах.

Интерфейс - это набор методов и свойств без реализации. Классы могут реализовать интерфейс, чтобы обеспечить реализацию методов и свойств, указанных в интерфейсе.

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

Необходимо решить задачу с использованием методологии ООП. Для чего необходимо подобрать самостоятельно соответствующую проблемную

(предметную/доменную) область, которая базируется на объектах и событиях реального мира (примеры соответствующих предметных областей приведены ниже). Спроектировать классы (собственные пользовательские типы данных)

в языке Java для программного представления данных объектов и основной

логики будущей программной системы. Система должна решать, как минимум, два полезных действия и иметь дополнительно следующие опции:

- не менее 3 разнообразных классов предметной области;

- не менее 5 атрибутов (состояния) и методов (поведения) в классе-сущности;

- не менее 3 методов, которые реализуют бизнес-логику программы, в соответствующих функциональных классах;

- хранить глобальные характеристики системы или характеристики уровня отдельных классов.

На базе спроектированной программной системы реализовать программу и продемонстрировать её работоспособность.

Предметная область: “Рыцарь (Knight). Есть рыцаря, который экипирован соответствующей амуницией. Необходимо определить общую стоимость всей амуниции, а также найти самую дорогую (недорогую) амуницию рыцаря.”

1. Состав проекта

Проект построен на основе MVC. В данном проекте содержатся файлы: Controller.cs, Item.cs, Inventory.cs, MainWindow.xaml. Ниже приведен листинг кода каждого файла.

***Controller.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace Knight

{

public partial class MainWindow : Window

{

private void Button\_Click(object sender, RoutedEventArgs e)

{

try

{

string name = nameField.Text;

string description = descriptionField.Text;

float weight = float.Parse(weightField.Text);

float cost = float.Parse(costField.Text);

Item item = new Item(name, description, weight, cost);

inventory.AddItem(item);

dgInventory.Items.Refresh();

Calculate();

}

catch

{

MessageBox.Show("Всё накрылось медным тазом...");

}

}

private void AddDefaultItems()

{

inventory.AddItem(new Item("Стандартный меч",

"Твой первый меч. Бесценен. Нет, буквально, он никому не нужен.", 2f, 0f));

inventory.AddItem(new Item("Стандартная броня",

"Ты с огромным трудом можешь назвать свою рубашку бронёй, но ладно", 0.5f, 0.1f));

inventory.AddItem(new Item("Зелеье исцеления",

"Хоть что-то полезное. Лечит 10% от макс HP", 0.1f, 10f));

}

public void Calculate()

{

Console.Clear();

WriteLine(String.Format("Стоимость всей амуниции составляет {0} тугр.", inventory.GetOverallPrice().ToString()));

//конечно, можно обойтись без bufferItem, но тогда мы получим фигово читаемую матрёшку,

//где мы вызываем метод в методе. Нам это надо? Я считаю нет

Item bufferItem = inventory.GetCheapestItem();

WriteLine(String.Format("Самый дешёвый предмет - {0} стоимостью {1} тугр.", bufferItem.Name, bufferItem.Price));

bufferItem = inventory.GetExpensiveItem();

WriteLine(String.Format("Самый дорогой предмет - {0} стоимостью {1} тугр.", bufferItem.Name, bufferItem.Price));

}

}

}

***Item.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Knight

{

public class Item

{

public string Name

{

get { return \_name; }

set { if (value != "") \_name = value; }

}

public string Description

{

get { return \_description; }

set { if (value != "") \_description = value; }

}

public float Weight

{

get { return \_weight; }

set { if (value > 0) \_weight = value; }

}

public float Price {

get { return \_price; }

set { if (value >= 0) \_price = value; }

}

private float \_weight = 0;

private float \_price = 0;

private string \_name = "Неизвестный предмет";

private string \_description = "Только богам известно, на что он способен";

public Item(string name, string description)

{

\_name = name;

\_description = description;

}

public Item(string name, string description, float weight, float price)

{

\_name = name;

\_description = description;

\_weight = weight;

\_price = price;

}

public Item()

{

}

public override string ToString()

{

string info = "";

info += "\n Предмет: " + \_name;

info += "\n Описание: " + \_description;

info += "\n Вес: " + \_weight.ToString() + " кг.";

info += "\n Цена: " + \_price.ToString() + " тугр.";

return info;

}

}

}

***Inventory.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Knight

{

public class Inventory

{

public List<Item> Items;

public Inventory()

{

Items = new List<Item>();

}

public void AddItem(Item item)

{

Items.Add(item);

}

public float GetOverallPrice()

{

float sum = 0;

foreach (Item item in Items)

{

sum += item.Price;

}

return sum;

}

public Item GetCheapestItem()

{

Item cheapestItem = Items[0];

foreach(Item item in Items)

{

if(item.Price < cheapestItem.Price)

{

cheapestItem = item;

}

}

return cheapestItem;

}

public Item GetExpensiveItem()

{

Item expensiveItem = Items[0];

foreach (Item item in Items)

{

if (item.Price > expensiveItem.Price)

{

expensiveItem = item;

}

}

return expensiveItem;

}

}

}

***MainWindow.xaml:***

<Window x:Class="Knight.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Knight"

mc:Ignorable="d"

Title="MainWindow" Height="520" Width="800">

<Grid Loaded="Grid\_Loaded">

<TextBox x:Name="Console" HorizontalAlignment="Left" Margin="282,264,0,0" TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="495" Background="White" Height="193" Foreground="Black" BorderBrush="#FF030303"/>

<DataGrid x:Name="dgInventory" AutoGenerateColumns="False" ItemsSource="{Binding inventory}" Margin="0,0,23,245">

<DataGrid.Columns>

<DataGridTextColumn Header="Название" Binding="{Binding Path=Name}"/>

<DataGridTextColumn Header="Описание" Binding="{Binding Path=Description}"/>

<DataGridTextColumn Header="Вес" Binding="{Binding Path=Weight}"/>

<DataGridTextColumn Header="Цена" Binding="{Binding Path=Price}"/>

</DataGrid.Columns>

</DataGrid>

<Grid Background="#FFEEEEEE" HorizontalAlignment="Left" Width="267" Margin="10,264,0,0" Height="193" VerticalAlignment="Top">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="121\*"/>

<ColumnDefinition Width="186\*"/>

</Grid.ColumnDefinitions>

<TextBox x:Name="nameField" HorizontalAlignment="Left" Margin="69,7,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="188" Grid.ColumnSpan="2" Text="Нечто"/>

<TextBox x:Name="descriptionField" HorizontalAlignment="Left" Margin="69,30,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="188" Height="63" Grid.ColumnSpan="2" Text="Неизвестного назначения"/>

<TextBox x:Name="weightField" HorizontalAlignment="Left" Margin="69,98,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="188" Grid.ColumnSpan="2" Text="0"/>

<TextBox x:Name="costField" HorizontalAlignment="Left" Margin="69,121,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="188" Grid.ColumnSpan="2" Text="0"/>

<Label Content="Название" HorizontalAlignment="Left" Margin="3,2,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.121,0.171"/>

<Label Content="Описание" HorizontalAlignment="Left" Margin="3,25,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.121,0.171"/>

<Label Content="Вес" HorizontalAlignment="Left" Margin="4,94,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.121,0.171"/>

<Label Content="Цена" HorizontalAlignment="Left" Margin="4,117,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.121,0.171"/>

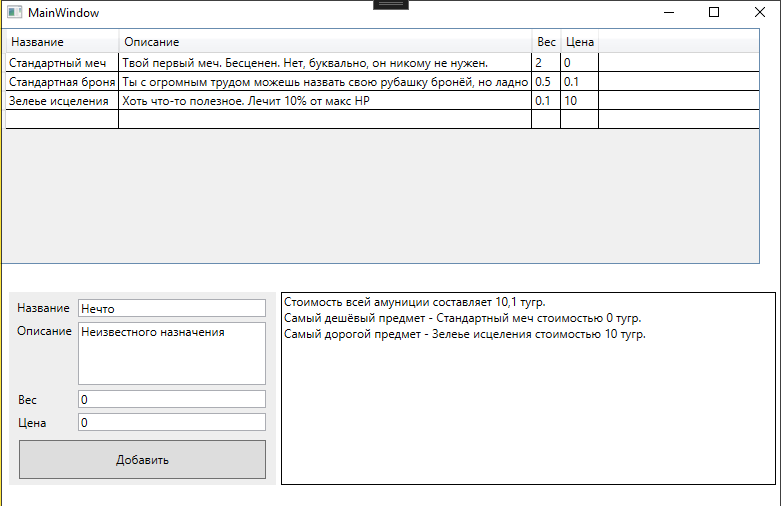
<Button Content="Добавить" HorizontalAlignment="Left" Margin="10,148,0,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Width="247" Height="39" Click="Button\_Click"/>

</Grid>

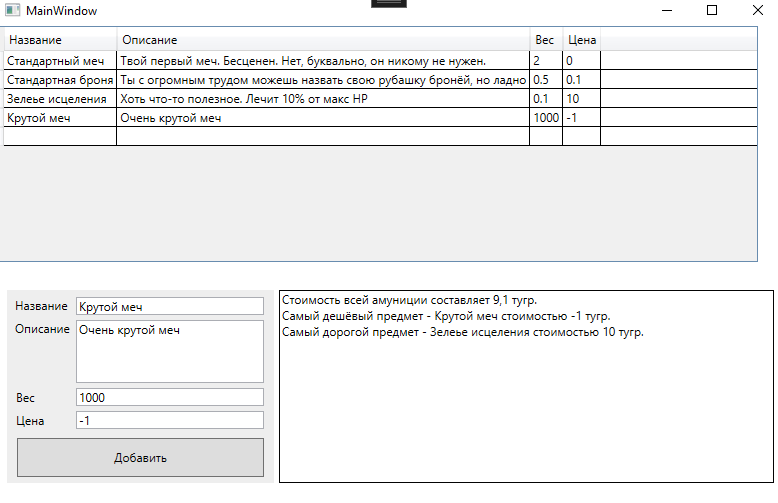
</Grid>

</Window>

1. Cкриншоты работы программы

**

*Скриншот 1. Окно программы*

**

*Скриншот 2. Программа после добавления предмета*



*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