МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
РОСТОВСКОЙ ОБЛАСТИ
ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
РОСТОВСКОЙ ОБЛАСТИ
«РОСТОВСКИЙ-НА-ДОНУ КОЛЛЕДЖ СВЯЗИ И ИНФОРМАТИКИ»

КУРСОВОЙ ПРОЕКТ

По профессиональной дисциплине ОП.11 Объектно-ориентированное программирование
Тема Библиотека классов, описывающая сотрудников предприятия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

РКСИ.К22.09.02.03.ОП.18.3885.00ПЗ
(шифр)

Специальность 09.02.03 «Программирование в компьютерных системах»

Выполнил студентка _____ Кривко Д.А. Группа ПОКС-31
подпись Ф.И.О.

Руководитель _____ Бельчич Д.С.
подпись Ф.И.О.

Проект защищен с оценкой _____ «___»_____2022 г.

2022г.

МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
РОСТОВСКОЙ ОБЛАСТИ

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
РОСТОВСКОЙ ОБЛАСТИ
«РОСТОВСКИЙ-НА-ДОНУ КОЛЛЕДЖ СВЯЗИ И ИНФОРМАТИКИ»

ЗАДАНИЕ
на курсовой проект (работу)

По профессиональной дисциплине_____
ОП.11 «ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»
Студенту Кривко Д.А. группы ПОКС-31

Тема «Библиотека классов, описывающая сотрудников предприятия.

1 Исходные данные
Разработать библиотеку классов, описывающую различных сотрудников предприятия.

2. Содержание пояснительной записки

Наименование разделов проекта	Объем в %	Срок выполнения
Введение	6	20.12.2022
Теоретические основы	38	20.12.2022
Практическая часть	44	20.12.2022
Заключение	6	20.12.2022
Список использованных источников и литературы	6	20.12.2022

3. Содержание графической части
Реализация функцианального каталога предметов для игрового приложения

Дата выдачи задания «___»_____20___г.
Задание рассмотрено
на заседании ЦК программирования,
протокол №__ от «___»_____20___г.
Председатель ЦК
_____ _____
подпись Ф.И.О.
«___» _____20___ г

Дата сдачи _____20___г.
Руководитель проекта
_____ Бельчич Д.С.
подпись Ф.И.О.
Студент _____
(подпись)

Содержание

Введение

Актуальность исследуемой темы состоит в том, что для игровых приложений в жанре RPG(от англ. Role-Playing Game) требуется каталог товаров что может продавтать продавец.

Объектом исследования данной курсовой работы является информация и ее вывод пользователю. Предметом исследования будут предметы и их классификация в каталоге товаров, их стоимость и характеристки.

Для изучения этой темы информационной базой послужили различные сетевые информационные ресурсы.

В первой главе будет описано наполнение предметов, различные виды и характеристики, а после будет произведена постановка задачи на разработку библиотеки классов реализующую каталог для пгрового приложения. После того как будут обозначены основные определения будет необходимо описать все операции, а также рамки, в которых они могут быть проведены. Также будет наглядно изображен процесс проведения каждой задачи. После будет произведена постановка задачи, в которой будет определены основные функциональные требования к разрабатываемой программе, а также выделить основные задачи, которые потребуется решить при разработке библиотеки классов для игрового приложения. Для того чтобы можно было определить функциональные требования необходимо будет провести анализ каких-либо известных предметов добавленных в игру.

Во второй главе будет разобрано определение библиотеки классов и всех данных, связанных с классов, такие как поля класса, методы, используемые в классах и прочее. Для этого будут использованы таблицы с описанием каждого метода, и краткое описание каждой используемой структурой, а после будет проведена формализация этих данных. Как только будет проведена формализация данных, необходимо выбрать

основные средства разработки для максимальной реализации разрабатываемой программы. После основной упор будет распределён на создание библиотеки классов. Это необходимо для проработки каждого условия, для проведения операций вывода информации о товарах каталога и для того, чтобы не допустить появлению ошибок уже в тестировании библиотеки классов.

1 Анализ предметной области

1.1 Определение игрового приложения, виды и области применения

Задание данного курсового проекта состоит в том, чтобы разработать библиотеку классов для реализации игрового приложения. В текущей работе потребуется создать библиотеку классов.

Компьютерная игра — компьютерная программа, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра. В настоящее время, в ряде случаев, вместо компьютерная игра может использоваться видеоигра, то есть данные термины могут употребляться как синонимы и быть взаимозаменяемыми.

В работе не будет использоваться система игрового приложения, в угоду удобства разработки библиотеки классов, а также, не углубляясь слишком детально в различия данных средств.

На Рисунке 1 изображены жанры компьютерных игр.

Рисунок 1 – Жанры компьютерных игр.

В проекте система классов будет сведена к главным и отличающимся характеристикам. Для этого будет разработана библиотека классов, которая будет выводить данные о предметах игры в. С помощью данного описания можно рассмотреть различные предметы и удостоверится в разнообразии.

1.2 Постановка задачи

Разработать библиотеку классов, каталога товаров для реализации игрового приложения.

После анализа предметной области необходимо определить функциональные требования к разрабатываемой программе, а также

выделить основные задачи, которые потребуется решить при разработке библиотеки классов реализующую библиотеку классов для вывода информации о игровом приложении.

Библиотека классов должна быть реализована по разработанной во второй главе диаграмме классов.

Для определения всех требований нужно определить основные спецификации сотрудника, и какую задачу он выполняет.

В проекте будут использованные основные операции вывода информации, а именно: вывод названия предметов, а так же их характеристик и стоимости в магазине.

Операция, к которой будут определены требования, будет вывод характеристик. Для проведения данной операции должно выполняться главное условие. А именно это условие будут заключаться уже в приложении тем, что операция должна будет выполняться только для определенного вида выбранного предмета.

Для операции вывода данных только об данном виде, а именно правильный ввод данных по запросу, иначе программа сообщит пользователю о не верном формате и прервет операцию.

После того как определились все требования к основной операции, необходимо определить требования к процессу определения используемой классификации, заключается в возможности пользователем выбрать нужную ему классификации и после этого иметь возможность активировать операции вывода. Также одним из основных требований к приложению необходимость использовать библиотеки классов для каждой из операций, а после использовать их. Каждый класс должен быть представленным в разработанной в следующей главе диаграмме классов, а после использоваться для проведения всех операций. После определения всех требований к функциональной части системы библиотеки классов необходимо определить порядок последующей реализации библиотек классов с использованием диаграммы классов и кодированием каждой

операции, разработка которых будет осуществляться во второй главе.

2 Создание библиотеки классов

2.1 Определение библиотеки классов и формализация в библиотеку классов.

Для определения библиотеки классов необходимо сначала разобрать понятие класса, его свойства и методы, иерархии, а также наследование, для дальнейшей реализации в виде курсового проекта и программного кода. После определения класса используемых методов свойств необходимо будет формализовать эти данные в проектирование диаграммы классов, которая будет использована для разработки приложения. Библиотека классов определяет типы и методы, которые могут быть вызваны из любого приложения.

Класс — это понятие в объектно-ориентированном программировании, который является моделью для создания объектов определённого типа, описывающая их структуру (набор полей и их начальное состояние) и определяющая алгоритмы (функции или методы) для работы с этими объектами. Класс служит средством для введения абстрактных типов данных в программный проект. Другие описатели абстрактных типов данных — метаклассы, интерфейсы, структуры, перечисления, — характеризуются какими-то своими особенностями. Суть отличия классов состоит в том, что при задании типа данных, класс определяет одновременно как интерфейс, так и реализацию для всех своих экземпляров поэтому вызов метода-конструктора обязателен. Идея использования классов пришла из работ по базам знаний, имеющих отношение к исследованиям по искусственному интеллекту. Используемые человеком классификации в зоологии, ботанике, химии, деталях машин, несут в себе основную идею, что любую вещь всегда можно представить частным случаем некоторого более общего понятия. Класс — в объектно-

ориентированном программировании, представляет собой некий шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов. Класс является наиболее распространенной разновидностью ссылочного типа. Класс состоит из основного ключевого слова, а также имя класса должно с допустимым именем идентификатора используемого языка библиотеки классов в приложении. Оставшаяся часть определения — это тело класса, в котором задаются данные и поведение. Поля, методы, свойства и события в классе собирательно и называются членами класса.

Поле — это переменная, которая является членом класса или структуры. У поля есть единственный модификатор readonly который предотвращает изменение поля после его создания.

Присваивать значение полю, допускающему только чтение, можно только в его объявлении или внутри конструктора типа, где оно определено. Метод класса выполняет действие, представленное в виде последовательности операторов. Метод может получать входные данные из вызывающего кода посредством указания параметров и возвращать выходные данные обратно вызывающему коду за счет указания возвращаемого типа. Для метода может быть определен возвращаемый тип void, который говорит о том, что метод никакого значения не возвращает.

Для классов могут быть использованы последующие методы:

Методы, сжатые до выражений суть данного метода, заключается в том, что он состоит из единственного выражения;

Локальные методы. Данные методы можно определять внутри другого метода. Преимуществом локальных методов в том, что они могут обращаться к локальным переменным и параметрам охватывающего метода. Локальные методы могут появляться внутри функций других видов, таких как средства доступа к свойствам, конструкторы и т. д. Локальные методы можно даже помещать внутрь других локальных методов и лямбда-выражений, которые используют блок операторов. Локальные методы могут

быть итераторными или асинхронными;

Статические локальные методы. Добавление модификатора static который к локальному методу запрещает ему видеть локальные переменные и параметры охватывающего метода. Это помогает ослабить связность и предотвращает случайную ссылку в локальном методе на переменные из содержащего метода;

Локальные методы и операторы верхнего уровня. Любые методы, которые объявляются в операторах верхнего уровня, трактуются как локальные методы. Таким образом, они могут обращаться к переменным в операторах верхнего уровня;

Конструктор — это метод, имя которого совпадает с именем его типа. Сигнатура метода содержит только необязательный Модификатор доступа, имя метода и список параметров. Конструктор включает тип возвращаемого значения, и они выполняют код инициализации класса или структуры. Конструктор определяется подобно методу за исключением того, что вместо имени метода и возвращаемого типа указывается имя типа, к которому относится этот конструктор. Конструкторы, содержащие единственный оператор, также можно записывать как члены, сжатые до выражений.

Перегрузка конструкторов Класс или структура может перегружать конструкторы. Один перегруженный конструктор способен вызывать другой, используя ключевое слово this. Ключевое слово this ссылается на текущий экземпляр класса, а также используется в качестве модификатора первого параметра метода расширения.

Неявные конструкторы без параметров в работе автоматически генерируют для класса открытый конструктор без параметров, если и только если в нем не определено ни одного конструктора. Однако после определения хотя бы одного конструктора конструктор без параметров больше автоматически не генерируется. Когда один конструктор вызывает другой, то первым выполняется вызванный конструктор. Конструкторы не

обязательно должны быть открытыми. Распространенной причиной наличия неоткрытого конструктора является управление созданием экземпляров через вызов статического метода. Статический метод может использоваться для возвращения объекта из пула вместо создания нового объекта или для возвращения экземпляров разных подклассов на основе входных аргументов. Снаружи свойства выглядят похожими на поля, но подобно методам внутренне они содержат логику. Для классов могут быть использованы следующие свойства:

Свойства только для чтения и вычисляемые свойства. Свойство разрешает только чтение, если для него указано лишь средство доступа get, и только запись, если определено лишь средство доступа set. Свойства только для записи используются редко. Свойство обычно имеет отдельное поддерживающее поле, предназначенное для хранения внутренних данных. Тем не менее, свойство может также возвращать значение, вычисленное на основе других данных;

Автоматические свойства. Наиболее распространенная реализация свойства предусматривает наличие средств доступа get совместно либо лишь со средством доступа set, которые просто читают и записывают в закрытое поле того же типа, что и свойство. Объявление автоматического свойства указывает компилятору на необходимость предоставления такой реализации;

Все поля, методы и остальные компоненты класса имеют модификаторы доступа. Модификаторы доступа позволяют задать допустимую область видимости для компонентов класса. То есть модификаторы доступа определяют контекст, в котором можно употреблять данную переменную или метод.

В языке C# применяются модификаторы доступа, указанные в таблице 1.

Таблица 1 – Модификаторы доступа класса

Объявленная доступность	Значение
private	закрытый или приватный компонент класса или структуры. Приватный компонент доступен только в рамках своего класса или структуры.
private protected	компонент класса доступен из любого места в своем классе или в производных классах, которые определены в той же сборке
file	добавлен в версии C# 11 и применяется к типам, например, классам и структурам. Класс или структура с таким модификатором доступны только из текущего файла кода.
protected	такой компонент класса доступен из любого места в своем классе или в производных классах. При этом производные классы могут располагаться в других сборках.
internal	компоненты класса или структуры доступен из любого места кода в той же сборке, однако он недоступен для других программ и сборок
protected internal	совмещает функционал двух модификаторов protected и internal. Такой компонент класса доступен из любого места в текущей сборке и из производных классов, которые могут располагаться в других сборках.
public	публичный, общедоступный компонент класса или структуры. Такой компонент доступен из любого места в коде, а также из других программ и сборок.

Модификаторы уровня доступа различаются и на модификаторы доступа, разделенные на различные типы. В каждый тип входят вышеуказанные модификаторы. Для того чтобы показать все возможные типы ниже будет предоставлена таблица 2, и в ней будут показаны члены типа, уровень доступности для данного типа и доступные для них уровни

модификаторов.

Таблица 2 – Уровни доступности по типам

Члены типа	Уровень доступности членов по умолчанию	Допустимые объявленные уровни доступности члена
enum	public	-
class	private	public protected internal protected internal private
interface	public	-
struct	private	public internal private

Следующим модификаторами являются модификаторы, которые используются для того, чтобы определить можно было определить тип класса по его наследованию. Данные модификаторы также используются для того, чтобы указать наследование класса от модификатора, используемого в приложении. Однако для того, чтобы определить данные модификаторы необходимо определить, что такое наследование в классах, используемых в ООП. В объектно-ориентированного программирования используются расширенная реализация парадигмы объектной ориентации, которая включает инкапсуляцию, наследование и полиморфизм.

Класс может быть унаследован от другого класса с целью расширения или настройки первоначального класса.

Инкапсуляция означает создание вокруг объекта границы, предназначенной для отделения внешнего поведения от внутренних деталей реализации. часто называется третьим столпом объектно-ориентированного программирования после инкапсуляции и наследования.

В основе работы полиморфизма лежит тот факт, что подклассы

обладают всеми характеристиками своего базового класса. Однако обратное утверждение не будет верным.

Наследование является одним из фундаментальных атрибутов объектно-ориентированного программирования. Оно позволяет определить производный класс, который использует, расширяет или изменяет возможности родительского класса. Класс, члены которого наследуются, называется базовым классом. Класс, который наследует члены базового класса, называется производным классом. Наследование от класса позволяет повторно использовать функциональность данного класса вместо ее построения с нуля. Класс может наследоваться только от одного класса, но сам может быть унаследован множеством классов, формируя иерархию классов. После определения понятия наследования теперь необходимо определить используемые модификаторы, связанные с наследованием и их назначением в приложении. Данные модификаторы будут описаны в 3 таблице.

Таблица 3 – Модификаторы классов

Модификатор	Назначение
abstract	Указывает на то, что класс предназначен только для использования в качестве базового класса для других классов.
async	Указывает, что измененный метод, лямбда-выражение либо анонимный метод является асинхронным
const	Указывает на то, что значение поля или локальной переменной не может быть изменено.
event	Объявляет событие.
extern	Указывает на то, что объявляется метод с внешней реализацией.
new	Скрывает наследуемый член от члена базового класса
override	Указывает на то, что создается новая реализация виртуального члена, унаследованного от базового класса.

В приложении будет использоваться модификатор класса abstract для того, чтобы использовать абстрактный класс. Класс, объявленный как абстрактный, не разрешает создавать свои экземпляры. Взамен можно создавать только экземпляры его конкретных подклассов. В абстрактных

классах имеется возможность определять абстрактные члены. Абстрактные члены похожи на виртуальные члены за исключением того, что они не предоставляют стандартные реализации. Реализация должна обеспечиваться подклассом, если только подкласс тоже не объявлен как abstract. Последними модификаторами классов какие будут представлены и используемые в приложении являются модификаторы полей. В работе будут использованы несколько полей, один из которых ответственен за статичность. В таблице 4 будут предоставлены все модификаторы, используемые для полей, такие модификаторы необходимы при разработке любого класса.

Таблица 4 – Модификаторы полей

Модификатор	Назначение
partial	определение разделяемых классов, структур и методов в рамках одной сборки.
readonly	объявление поля, которому можно присваивать значения только на этапе объявления или с помощью конструктора этого же класса.
sealed	указывает на то, что нельзя создавать производные классы от этого класса
static	объявление члена, который принадлежит всему типу, а не конкретному объекту
unsafe	объявление небезопасного контекста
virtual	объявление обычного метода или метода доступа, реализацию которых можно переопределить в производном классе.
volatile	указывает на то, что поле может быть изменено в программе операционной системой, оборудованием, параллельным потоком и т. д

UML – унифицированный язык моделирования – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

2.2 Выбор средств разработки

Средства разработки программного обеспечения – совокупность приемов, методов, методик, а также набор инструментальных программ (компиляторы, прикладные/системные библиотеки и т. д.), используемых разработчиком для создания программного кода приложения, отвечающего заданным требованиям. Разработка программ – сложный процесс, основной целью которого является создание, сопровождение программного кода, обеспечивающего необходимый уровень надежности и качества. Для достижения основной цели разработки программ используются средства разработки программного обеспечения. В зависимости от предметной области и задач, поставленных перед разработчиками, разработка программ может представлять собой достаточно сложный, поэтапный процесс, в котором задействовано большое количество участников и разнообразных средств.

Для того чтобы определить, когда и в каких случаях какие средства применяются, выделяют следующие основные этапы разработки программного обеспечения:

- Проектирование приложения;
- Реализация программного кода приложения;
- Тестирование приложения.

На этапе реализации программного кода выполняется кодирование отдельных компонентов программы в соответствии с разработанным проектом, в случае кода необходимо выделить следующие основные виды средств какие будут использованы в приложении:

- Языки программирования;
- Средства создания пользовательского интерфейса;
- Средства получения исполняемого кода;
- Отладчики.

Язык программирования — формальный язык, предназначенный для

записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель под её управлением. В приложении будет использоваться язык программирования C#. C# является универсальным, безопасным в отношении типов, объектно-ориентированным языком программирования. Цель C# заключается в обеспечении продуктивности работы программистов. Для этого в языке соблюдается баланс между простотой, выразительностью и производительностью. Язык C# нейтрален в отношении платформ и работает с рядом исполняющих сред, специфических для платформ. В языке C# предлагается расширенная реализация парадигмы объектной ориентации, которая включает инкапсуляцию, наследование и полиморфизм. Так как данный язык программирования является ориентированным на ООП у него есть множество особенностей, которые могут быть в различных приложениях и преимуществом.

- Унифицированная система типов. Фундаментальным строительным блоком в C# является инкапсулированная единица данных и функций, которая называется типом. Язык C# имеет унифицированную систему типов, которые в итоге разделяют общий базовый тип.

- Классы и интерфейсы. В рамках традиционной объектно-ориентированной парадигмы единственной разновидностью типа считается класс. В языке C# присутствуют типы других видов, одним из которых является интерфейс. Интерфейс похож на класс, не позволяющий хранить данные. Это означает, что он способен определять только поведение, что делает возможным множественное наследование, а также отделение спецификации от реализации.

- Свойства, методы и события. В чистой объектно-ориентированной парадигме все функции представляют собой методы. В языке C# методы являются только одной разновидностью функций-членов, куда также относятся свойства и события (помимо прочих). Свойства — это функции-

члены, которые инкапсулируют фрагмент состояния объекта, такой как цвет кнопки или текст метки. События — это функции-члены, упрощающие выполнение действий при изменении состояния объекта. Так как данный язык программирования поддерживает базовые принципы ООП и имеют различные дополнительные особенности дает этому языку преимущество при разработке библиотеки классов.

Следующим средством разработки, какой является средство создания пользовательского интерфейса, средством получения исполняемого кода и средством для отладки будет использоваться интегрированная среда разработки Visual Studio.

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом.

Данная интегрированная среда разработки поддерживает в своем функционале шаблоны для языка C#, которые используют для разработки приложений с различным функционалом.

2.3 Программная реализация библиотеки классов

Финальная разработка системы классов будет проводиться

На основе разработанной выше диаграммы классов при помощи выбранных средств разработки.

Реализация приложения будет организована по следующему шаблону:

- Создание консольного приложения;

- Создание классов;

- Реализация классов;

- Разработка меню для взаимодействия с пользователем.

Для того чтобы приступить к разработке библиотеки классов необходимо для начала создать консольное приложение. На рисунках 2-3 изображен процесс создания консольного приложения, а на рисунке будет показан пример добавления любого элемента в приложение.

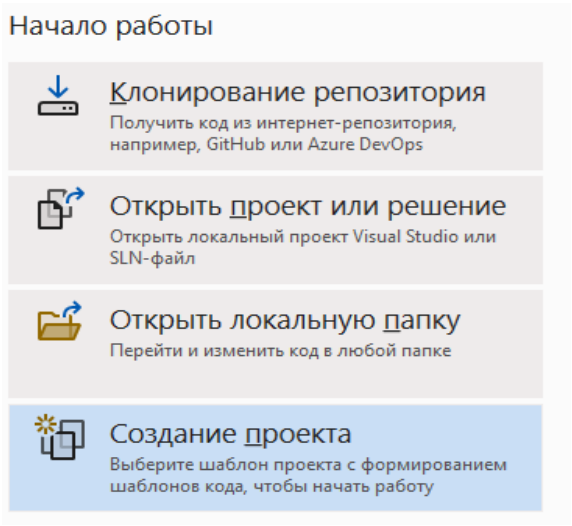


Рисунок 2 – создание проекта в MS Visual Studio

Далее система предоставит нам выбор – какой вид проекта мы хотим создать. Нас интересует консольное приложение. Для разработки консольного приложения лучше всего использовать стандартный шаблон от компании Майкрософт, то есть .Net Framework. Данное решение уменьшит возможности использование некоторых специальных модулей и свойств, однако даст возможность использовать кроссплатформенность приложения.

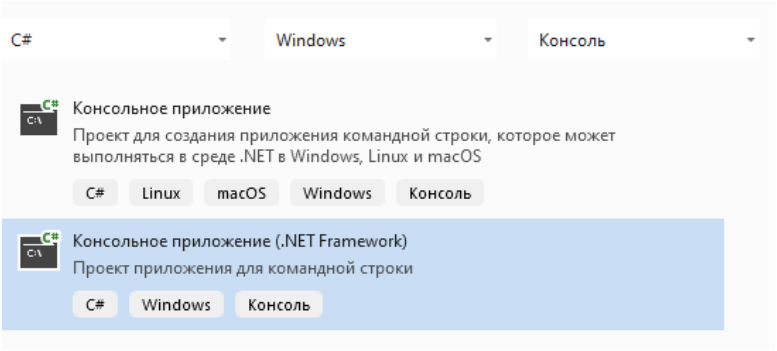


Рисунок 3 – создание консольного приложения в MS Visual Studio

Далее продемонстрируем добавление элементов в проект. Таким образом далее будут создаваться каждый из классов.

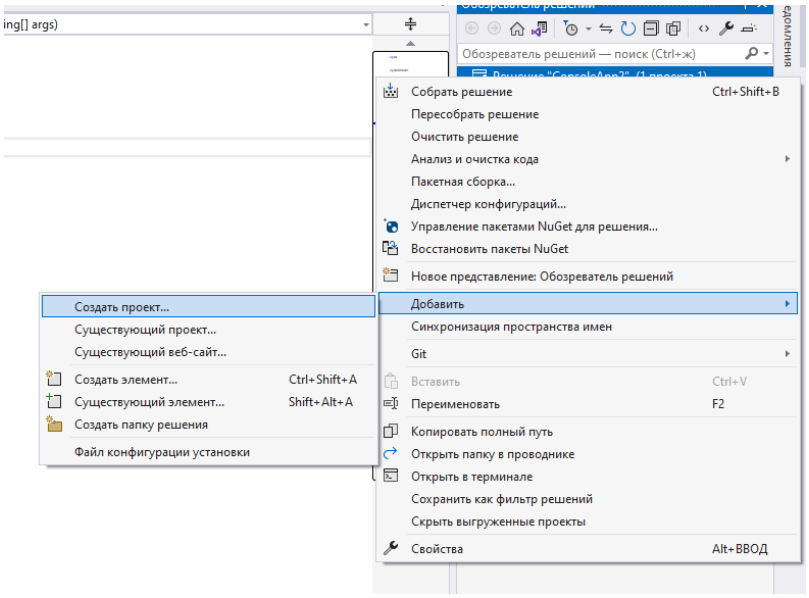


Рисунок 4 – добавление в проект элементов

Далее создадим каждый из классов приложения:

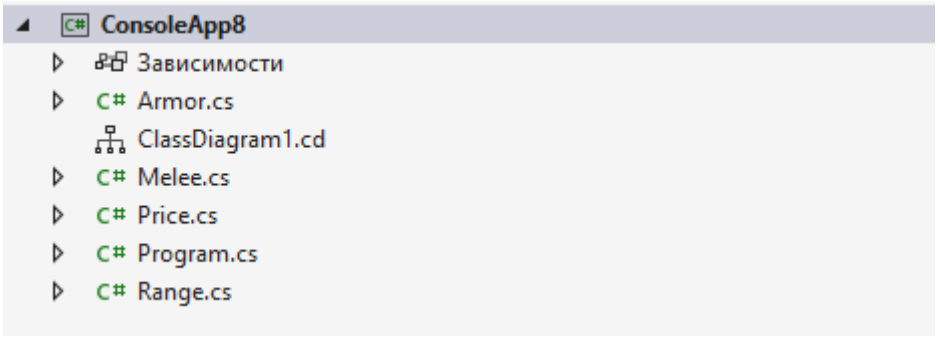


Рисунок 5 – готовая структура проекта

Задачей следующего этапа разработки приложения будет наполнение кодом в соответствии с диаграммой классов, классов приложения с использованием описанных методов и их наследования.

Начнем с класса RPG_Shop Он является абстрактным, так как по сути объекта в реальности «RPG_Shop». Так как класс абстрактный, то создать его экземпляр нельзя, поэтому данный класс служит неким шаблоном общих характеристик для других производных классов. В данном случае общими характеристиками являются свойства Armor, Melee, Price, Range. Так как класс абстрактный, экземпляр создать нельзя, поэтому выводить информацию об экземпляре не представляется возможным. Данный метод служит шаблоном общим для производных классов и должен быть обязательно реализован в дочерних классах.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp8
{
    Ссылок: 5
    public class Armor
    {
        public string Helmet;
        public string Chestplate;
        public string Leggings;
        public string Boots;

        Ссылок: 1
        public Armor(string helmet, string chestplate, string leggings, string boots)
        {
            this.Helmet = helmet;
            this.Chestplate = chestplate;
            this.Leggings = leggings;
            this.Boots = boots;
        }

        Ссылок: 1
        public void Print()
        {
            Console.WriteLine("_____________________ ");
            Console.WriteLine("Доспехи              | ");
            Console.WriteLine("_____________________|_____________________________");
            Console.WriteLine($"Шлем Бебры           || {Helmet}   |"  );
            Console.WriteLine("_____________________||_____________________________|");
            Console.WriteLine($"Нагрудник Саши Камня  || {Chestplate}  |");
            Console.WriteLine("_____________________||_____________________________|");
            Console.WriteLine($"Поножи Спанч-Боба     || {Leggings} |");
            Console.WriteLine("_____________________||_____________________________|");
            Console.WriteLine($"Сапоги Гермеса        ||{Boots}   |");
            Console.WriteLine("_____________________||_____________________________|");
        }
    }
}
```

Рисунок 6 – реализация класса Armor

Обратимся к классу Armor. Броня имеет свой вид, поэтому может отображаться по выбору.

```
Ссылок: 1
public void Print()
{
    Console.WriteLine("_____________________ ");
    Console.WriteLine("Доспехи              | ");
    Console.WriteLine("_____________________| _____________________________");
    Console.WriteLine($"Шлем Бебры           || {Helmet}   |"  );
    Console.WriteLine("_____________________||_____________________________|");
    Console.WriteLine($"Нагрудник Саши Камня  || {Chestplate}  |");
    Console.WriteLine("_____________________||_____________________________|");
    Console.WriteLine($"Поножи Спанч-Боба     || {Leggings} |");
    Console.WriteLine("_____________________||_____________________________|");
    Console.WriteLine($"Сапоги Гермеса        ||{Boots}   |");
    Console.WriteLine("_____________________||_____________________________|");
}
```

Рисунок 7 – ответ на запрос пользователя

Здесь же только инициализируются уникальные свойства для класса.

Как уже было сказано ранее, производные класс должны реализовывать метод , поэтому данный метод мы переопределяем и

«подгоняем» под параметры нашего дочернего класса.

```
{
    z:
    Console.WriteLine("_______________________________");
    Console.WriteLine("Выбери вид товара             |");
    Console.WriteLine("_______________________________|");
    Console.WriteLine("1-Доспехи                     |");
    Console.WriteLine("_______________________________|");
    Console.WriteLine("2-Оружие ближнего боя         |");
    Console.WriteLine("_______________________________|");
    Console.WriteLine("3-Оружие дальнего боя         |");
    Console.WriteLine("_______________________________|");
    Console.WriteLine("4-Закрыть                     |");
    Console.WriteLine("_______________________________|");
```

Рисунок 8 – переопределение метода

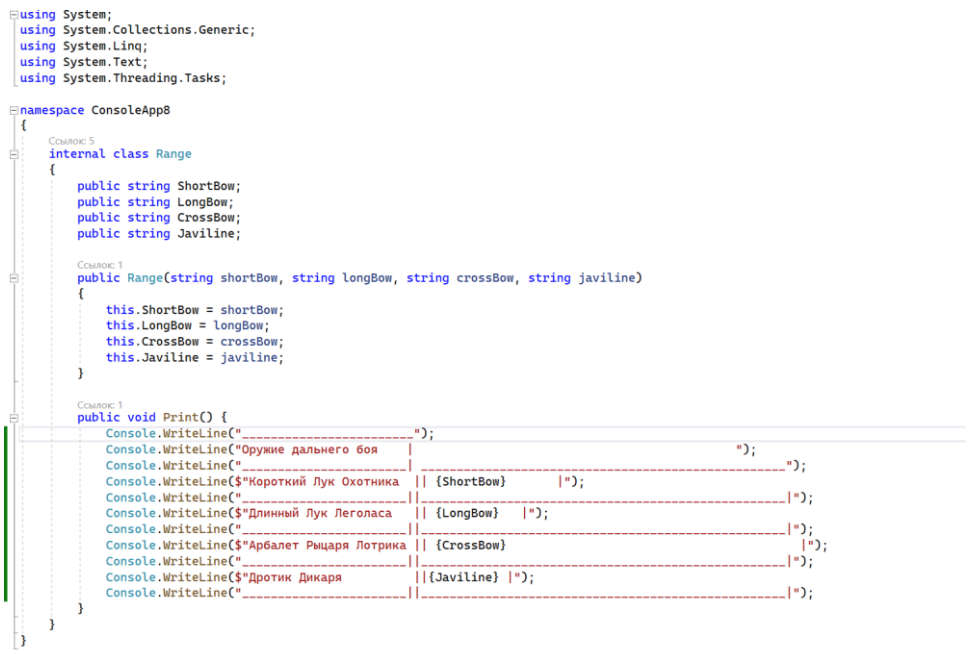
Подобным образом создаем класс Range(), Melee(), Price() Рисунки 9-.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp8
{
    Ссылок: 5
    public class Melee
    {
        public string LongSword;
        public string ShortSword;
        public string Axe;
        public string Spear;

        Ссылок: 1
        public Melee(string longsword, string shortsword, string axe, string spear)
        {
            this.LongSword = longsword;
            this.ShortSword = shortsword;
            this.Axe = axe;
            this.Spear = spear;
        }
        Ссылок: 1
        public void Print()
        {
            Console.WriteLine("_______________________________");
            Console.WriteLine("Оружие ближнего боя           |");
            Console.WriteLine("______________________________|_____________________________________");
            Console.WriteLine($"Двуручный Меч Короля Артура  || {LongSword}            |");
            Console.WriteLine("______________________________||_____________________________________|");
            Console.WriteLine($"Гладиус Центуриона           || {ShortSword}       |");
            Console.WriteLine("______________________________||_____________________________________|");
            Console.WriteLine($"Топор Кентавра               || {Axe}           |");
            Console.WriteLine("______________________________||_____________________________________|");
            Console.WriteLine($"Крылатое Копье               ||{Spear}          |");
            Console.WriteLine("______________________________||_____________________________________|");
        }
    }
}
```

Рисунок 9 – реализация метода и класса Melee



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp8
{
    Ссылок: 5
    internal class Range
    {
        public string ShortBow;
        public string LongBow;
        public string CrossBow;
        public string Javiline;

        Ссылок: 1
        public Range(string shortBow, string longBow, string crossBow, string javiline)
        {
            this.ShortBow = shortBow;
            this.LongBow = longBow;
            this.CrossBow = crossBow;
            this.Javiline = javiline;
        }

        Ссылок: 1
        public void Print() {
            Console.WriteLine("_________________________");
            Console.WriteLine("Оружие дальнего боя    |                                                  ");
            Console.WriteLine("_____________________| _____________________________________________________");
            Console.WriteLine($"Короткий Лук Охотника  || {ShortBow}       |");
            Console.WriteLine("_____________________||_____________________________________________________|");
            Console.WriteLine($"Длинный Лук Леголаса   || {LongBow}   |");
            Console.WriteLine("_____________________||_____________________________________________________|");
            Console.WriteLine($"Арбалет Рыцаря Лотрика || {CrossBow}                                          |");
            Console.WriteLine("_____________________||_____________________________________________________|");
            Console.WriteLine($"Дротик Дикаря          ||{Javiline} |");
            Console.WriteLine("_____________________||_____________________________________________________|");
        }
    }
}
```

Рисунок 10 – реализация метода и класса Range

Рисунок 10 – реализация метода и класса Price

3 Тестирование библиотеки классов

3.1 Разработка оконного приложения для тестирования

Тестирование приложений — это процесс в которым проверяют основные функциональные возможности приложения. Они должны выполняться быстро, поскольку цель таких тестов — убедиться, что основные возможности системы работают как запланировано.

В приложении для тестирования будет использоваться консольное приложение, которое будет считаться для пользователя оконным приложением. Для того чтобы разработать данное консольное приложение необходимо использовать метод main для составления основного функционала и для взаимодействия уже пользователя с разработанными выше операциями, а также является цикличной что позволит не закрываться приложения после выполнения одной операции. Для выхода из программы создано отдельное действие.

В методе также должно будет реализовано меню выбора действий. В самом начале будет выводиться в консоль список всех видов предметов, а также функций приложения. На рисунке 11 изображен список, который выводится на консоль при помощи системного метода Console.WriteLine().

```
Console.WriteLine("_______________________________");
Console.WriteLine("Выбери вид товара             |");
Console.WriteLine("_______________________________|");
Console.WriteLine("1-Доспехи                     |");
Console.WriteLine("_______________________________|");
Console.WriteLine("2-Оружие ближнего боя         |");
Console.WriteLine("_______________________________|");
Console.WriteLine("3-Оружие дальнего боя         |");
Console.WriteLine("_______________________________|");
Console.WriteLine("4-Закрыть                     |");
Console.WriteLine("_______________________________|");
```

Рисунок 11 - Код списка со всеми видами предметов

После будет реализована система выбора необходимого действия при помощи оператора switch…case, и для этого будут разработаны специальные структуры, в которые будут входить код программы для вывода определённого вида предмета, стоимость и характеристики.

Первой структурой, которая будет включать в себя код, будет в приложении отображаться цифрой 1 и будет реализовывать класс Armor в консольном приложении. Данная структура изображена на рисунке 12.

```
case 1:
    {

        armor.Print();
        price.Print();
        Console.WriteLine("--------------------");
        Console.WriteLine("Назад-1, Закрыть-2");
        Console.WriteLine("--------------------");
        b = Convert.ToInt32(Console.ReadLine());

        switch (b)
        {
            case 1:
                goto z;
            case 2:
                Environment.Exit(0);
                break;
        }
        break;
    }
```

Рисунок 12 – Код структуры с экземпляром класса Armor

Структура данного кода будет идентичной для Брони. Данная структура заключаться в одинаковом алгоритме вывода информации. Для того чтобы была возможность использовать операции вывода в данном оконном приложении в коде используется экземпляр класса, который после реализует необходимый класс, в данном случае реализуется класс Armor. Из-за крайне схожего кода блоки с видами предметов и рассматриваться не будут.

Второй и последней описанной структурой в приложении будет

структура, которая отвечает за возврат к выбору или закрытия консольного приложения. Последней она является, так как данная структура также используется во всех структурах, а отличается лишь данными. В данном случае алгоритм будет заключаться в выводе вопроса пользователю, в зависимости от ответа которого вернет его к экрану выбора, либо закроет консоль. На рисунке 13 изображен код данной структуры.

```
Console.WriteLine("--------------------");
Console.WriteLine("Назад-1, Закрыть-2");
Console.WriteLine("--------------------");
b = Convert.ToInt32(Console.ReadLine());
switch (b)
{

    case 1:
        goto z;
    case 2:
        Environment.Exit(0);
        break;
}
break;
```

Рисунок 13 - Код структуры с возвратом к выбору

После описания всех структур и кодирования их функционала разработка оконного приложения завершена. Следующим этапом для тестирования приложения будет выбор методики тестирования и после проведение тестирования по выбранному наиболее эффективному способу.

3.2 Определение методики тестирования

Методология тестирования программного обеспечения определяется как различные подходы, стратегии и типы тестирования для тестирования

приложения.

Тестирование различается на функциональное и нефункциональное.

Функциональное тестирование включает проверку, все ли бизнес-требования выполняются приложением без каких-либо ошибок. Он включает в себя все возможные типы тестов, чтобы проверить, ведет ли каждая часть приложения так, как ожидается, в соответствии с различными требованиями.

Нефункциональное тестирование выполняется для проверки производительности, удобства использования, надежности, совместимости приложения.

В проекте будет использовано функциональное тестирование. Из-за того, что проект разрабатывается для взаимодействия пользователя с небольшим функционалом, будет проще провести тестирование на работу всех операций вывода каждого класса.

Требования, которые нужно будет учитывать при тестировании:

Использовать все основные спецификации;

Спецификации с вводом значений должны выполняться только при выполнении условий, заданных приложением;

После определения требований можно приступить к проведению функционального тестирования приложения.

3.3 Тестирование системы библиотек классов

Первой для тестирования спецификацией вывода в проекте будет спецификация вывода товаров. Для проведения тестирования будет тип «Доспехи». Для проведения данной операции будет необходимо написать цифру один, а после будет предоставлен список брони. На рисунке 14 изображен результат вывода информации о сотрудниках.

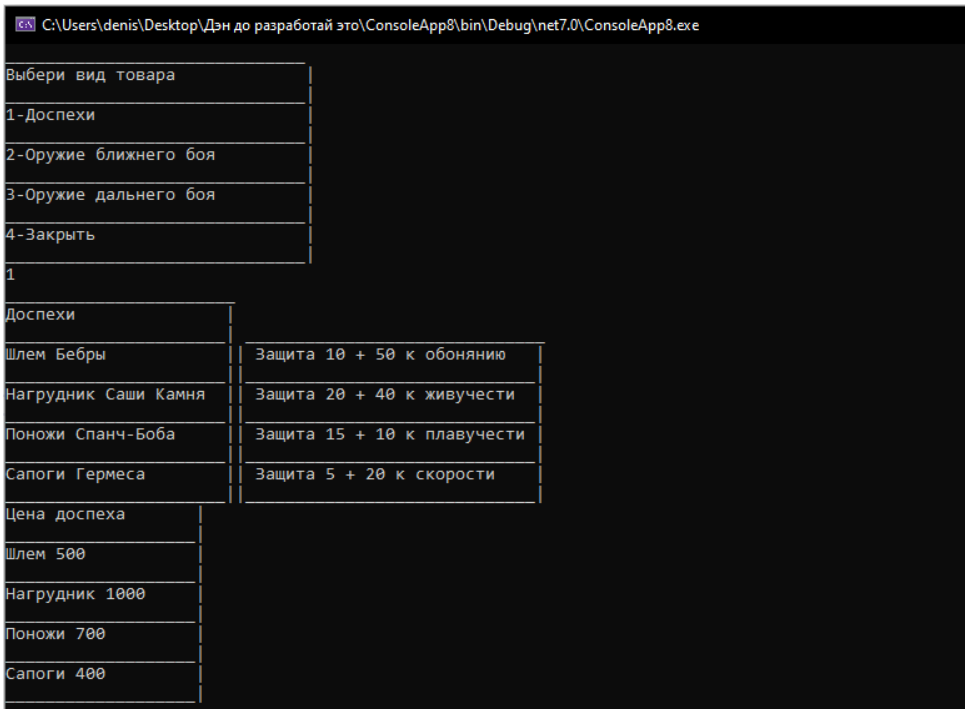


Рисунок 14 – Тестирование спецификации вывода брони

Второй спецификацией, которая будет протестированная в приложении, будет просмотр всех орудий ближнего боя этого вида. Для проведения тестирования будет использоваться спецификация входные данные. На рисунке 15 изображен результат данной операции.

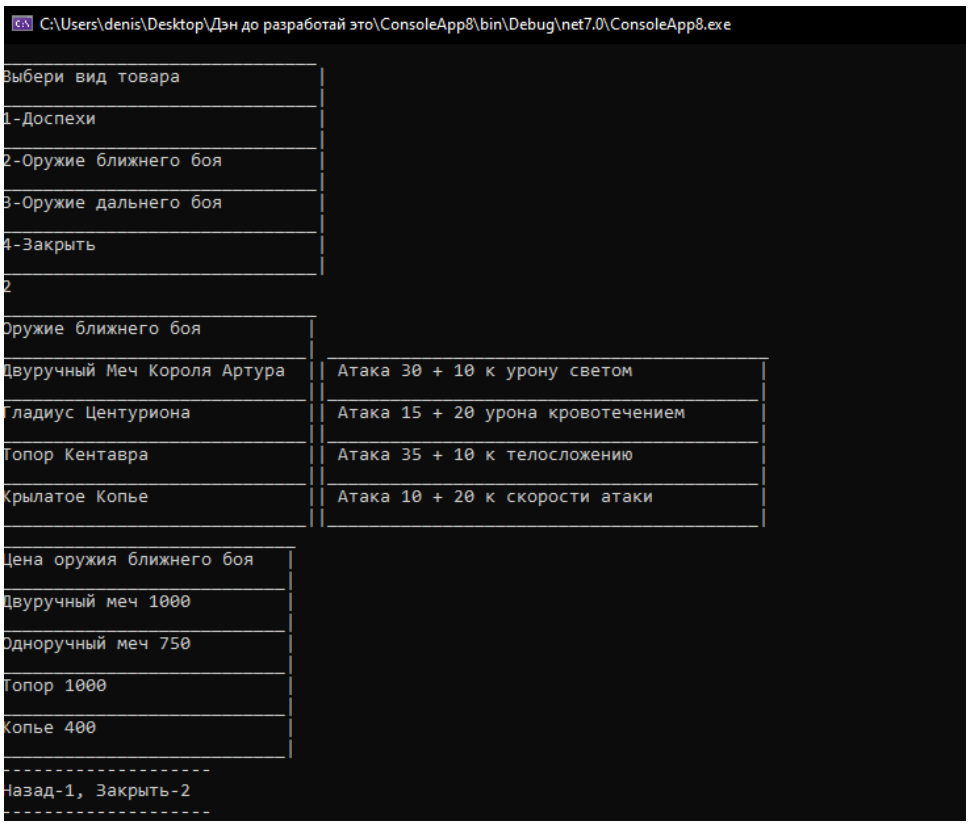


Рисунок 15 – Тестирование вывода всех с орудий ближнего боя в них.

И последней проверкой будет вывод орудий дальнего боя. Примером его выполнения смотреть в рисунке 16

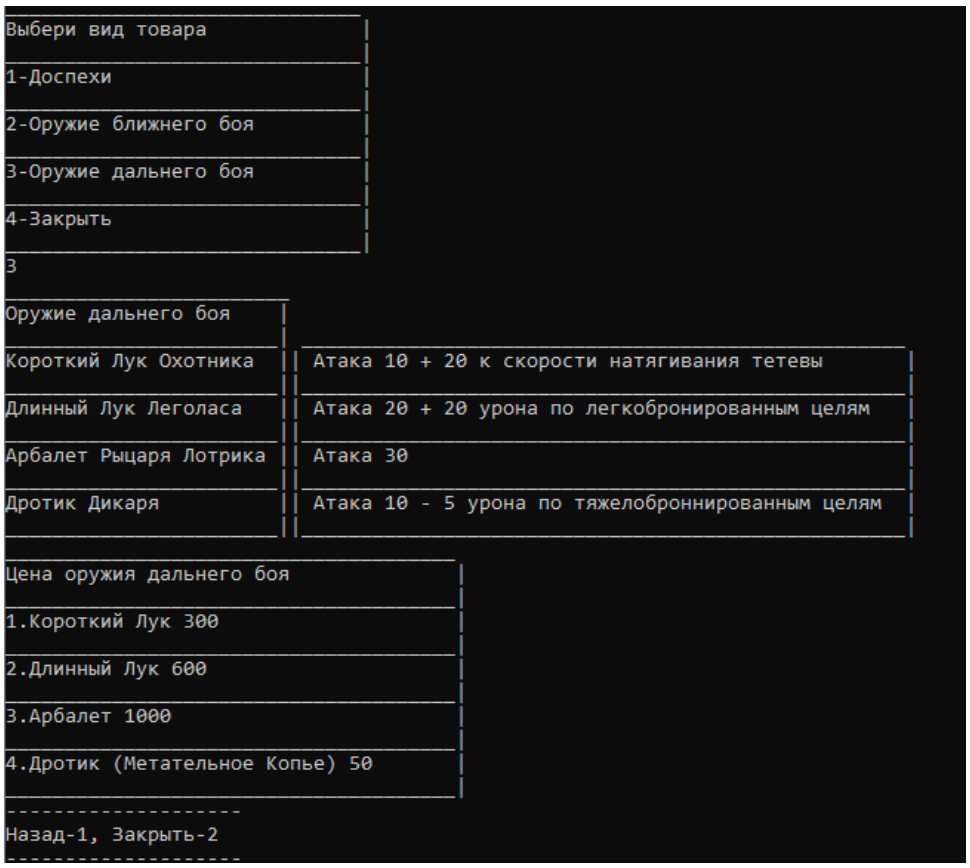


Рисунок 16 – Тестирование создания нового рабочего.

После проверки всех операций, а также самого консольного приложения, который в работе выступает в роли оконного приложения, можно прийти к выводу того, что все функциональные части соответствуют требованиям и тестирование приложения завершено.

Заключение

Целями данной работы были анализ, программная создание и тестирование библиотеки классов.

Целью анализа было понять предметную область и место ее применения, что было выполнено при разборе понятия нужд игрового приложения.

Целью программной было понять, что такое библиотека классов, её основные составляющие и как они между собой взаимодействуют, что было выполнено при создании UML таблицы библиотеки классов.

Целью тестирования было выявление ошибок в программном коде библиотеки классов и их устранения, что было показано выше со всеми методами.

По итогу выполнения данного курсового проекта была разработана и реализована библиотека классов предоставляющая основной функционал классов, которые используются для вывода описания различных предметов в внутреигровом магазине.

Данное приложение может быть усовершенствовано, из-за этого оно использует в проекте простые решение, конструкции, которые могут быть дополнены либо изменены в соответствии с требованиями заказчика или компании. Библиотека классов, реализованная в оконном приложении, которым выступает консоль, не нуждается в установке из-за того, что просто запускается от одного единого ярлыка после отладки приложения. Также приложения просто в использовании, ибо сразу отображается его основой функционал в виде списка. Само приложение не нуждается в установке дополнительного программного обеспечения и, может быть, распространённо на всех операционных системах, которые реализует шаблон, используемый при реализации приложения.

На данном этапе развитие функционала библиотеки класса завершено. В случае же продолжения разработки в сторону расширения

функционала следует добавить новые классы, которые будут отвечать за новые спецификации, к примеру, как , заработная плата, , которые будут необходимы для пользователя. Также можно было бы изменить шаблон приложения на многооконное. Данное изменение шаблона позволит изменить дизайн, и возможности различных операций, к примеру покупка товаров в магазине, их продажа, отображение кошелька и системы скидок для игрока.

Исходный код библиотеки классов, состоящий из различных классов с операциями подробно прокомментирован и изображен в данной работе, но само приложение может быть дополнено человеком, который знает язык программирования C# и принципы объектно-ориентированного программирования. Для основных возможностей приложения, реализующего операции вывода описания различных типов предмета для обычного пользователя в виде студента весь функционал приложения является удовлетворительным.

Приложение А

Класс Armor:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp8
{
    public class Armor
    {
        public string Helmet;
        public string Chestplate;
        public string Leggings;
        public string Boots;

        public Armor(string helmet, string chestplate, string leggings, string
boots)
```

```
        {
            this.Helmet = helmet;
            this.Chestplate = chestplate;
            this.Leggings = leggings;
            this.Boots = boots;
        }

        public void Print()
        {
            Console.WriteLine("_____ ");
            Console.WriteLine("Доспехи          | ");
            Console.WriteLine("_____|
_____");
            Console.WriteLine($"Шлем Бебры        || {Helmet}   |"  );

Console.WriteLine("_____||_____
__|");
            Console.WriteLine($"Нагрудник Саши Камня  || {Chestplate}  |");

Console.WriteLine("_____||_____
__|");
            Console.WriteLine($"Поножи Спанч-Боба    || {Leggings} |");

Console.WriteLine("_____||_____
__|");
            Console.WriteLine($"Сапоги Гермеса      ||{Boots}   |");

Console.WriteLine("_____||_____
__|");
        }
    }
}
```

Класс Melee:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp8
{
    public class Armor
    {
```

```
        public string Helmet;
        public string Chestplate;
        public string Leggings;
        public string Boots;

        public Armor(string helmet, string chestplate, string leggings, string
boots)
        {
            this.Helmet = helmet;
            this.Chestplate = chestplate;
            this.Leggings = leggings;
            this.Boots = boots;
        }

        public void Print()
        {
            Console.WriteLine("_____ ");
            Console.WriteLine("Доспехи          | ");
            Console.WriteLine("_____|
_____");
            Console.WriteLine($"Шлем Бебры       || {Helmet}   |" );

Console.WriteLine("_____||_____
___|");
            Console.WriteLine($"Нагрудник Саши Камня  || {Chestplate}
|");

Console.WriteLine("_____||_____
___|");
            Console.WriteLine($"Поножи Спанч-Боба    || {Leggings} |");

Console.WriteLine("_____||_____
___|");
            Console.WriteLine($"Сапоги Гермеса     ||{Boots}   |");

Console.WriteLine("_____||_____
___|");
        }
    }
}
```

Класс Range:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace ConsoleApp8
{
    internal class Range
    {
        public string ShortBow;
        public string LongBow;
        public string CrossBow;
        public string Javiline;

        public Range(string shortBow, string longBow, string crossBow,
string javiline)
        {
            this.ShortBow = shortBow;
            this.LongBow = longBow;
            this.CrossBow = crossBow;
            this.Javiline = javiline;
        }

        public void Print() {
            Console.WriteLine("_____________________________");
            Console.WriteLine("Оружие дальнего боя    |
");
            Console.WriteLine("_____________________|
_____________________________________________");
            Console.WriteLine($"Короткий Лук Охотника  || {ShortBow}
|");

Console.WriteLine("_____________________||_____________________________
_____________________________|");
            Console.WriteLine($"Длинный Лук Леголаса   || {LongBow}
|");

Console.WriteLine("_____________________||_____________________________
_____________________________|");
            Console.WriteLine($"Арбалет Рыцаря Лотрика || {CrossBow}
|");

Console.WriteLine("_____________________||_____________________________
_____________________________|");
            Console.WriteLine($"Дротик Дикаря          ||{Javiline} |");

Console.WriteLine("_____________________||_____________________________
_____________________________|");
        }
```

```
    }
}
```

Класс Price:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp8
{
    internal class Range
    {
        public string ShortBow;
        public string LongBow;
        public string CrossBow;
        public string Javiline;

        public Range(string shortBow, string longBow, string crossBow,
string javiline)
        {
            this.ShortBow = shortBow;
            this.LongBow = longBow;
            this.CrossBow = crossBow;
            this.Javiline = javiline;
        }

        public void Print() {
            Console.WriteLine("_____");
            Console.WriteLine("Оружие дальнего боя    |
");
            Console.WriteLine("_____|
_____");
            Console.WriteLine($"Короткий Лук Охотника  || {ShortBow}
|");

Console.WriteLine("_____||_____
_____|");
            Console.WriteLine($"Длинный Лук Леголаса   || {LongBow}
|");

Console.WriteLine("_____||_____
_____|");
```

```
                Console.WriteLine($"Арбалет Рыцаря Лотрика || {CrossBow}
|");

Console.WriteLine("_____||_____
_____|");
                Console.WriteLine($"Дротик Дикаря          ||{Javiline} |");

Console.WriteLine("_____||_____
_____|");
            }
        }
    }
```

Класс RPG_Shop:

```
using System;
using System.Security.Authentication.ExtendedProtection;
using System.Security.Cryptography.X509Certificates;

namespace ConsoleApp8
{

    abstract class RPG_Shop
    {

        private static Armor GetArmor() {
            Armor Armor = new Armor ("Защита 10 + 50 к обонянию",
"Защита 20 + 40 к живучести", "Защита 15 + 10 к плавучести", " Защита 5 +
20 к скорости ");

            return Armor;
        }
        private static Melee GetMelee()
        {
            Melee Melee = new Melee("Атака 30 + 10 к урону светом",
"Атака 15 + 20 урона кровотечением", "Атака 35 + 10 к телосложению ", "
Атака 10 + 20 к скорости атаки ");

            return Melee;
        }
        private static Range GetRange()
        {
            Range Range = new Range("Атака 10 + 20 к скорости
натягивания тетевы", "Атака 20 + 20 урона по легкобронированным целям",
"Атака 30 ", " Атака 10 - 5 урона по тяжелоброннированным целям ");
```

```
    return Range;
}
private static Price GetPrice()
{
    Price Price = new Price( 500, 1000, 700 ,400);

    return Price;
}

private static Price2 GetPrice2()
{
    Price2 Price2 = new Price2( 1000, 750, 1000, 400);

    return Price2;
}

private static Price3 GetPrice3()
{
    Price3 Price3 = new Price3(300, 600, 1000, 50);

    return Price3;
}
private static void Main(string [] args)

{
    z:
    Console.WriteLine("_____");
    Console.WriteLine("Выбери вид товара           |");
    Console.WriteLine("_____|");
    Console.WriteLine("1-Доспехи                   |");
    Console.WriteLine("_____|");
    Console.WriteLine("2-Оружие ближнего боя       |");
    Console.WriteLine("_____|");
    Console.WriteLine("3-Оружие дальнего боя       |");
    Console.WriteLine("_____|");
    Console.WriteLine("4-Закрыть                   |");
    Console.WriteLine("_____|");

    Armor armor = GetArmor();
    Price price = GetPrice();
    Melee melee = GetMelee();
    Price2 price2 = GetPrice2();
    Range range= GetRange();
    Price3 price3 = GetPrice3();
    int a,b;
```

```
a = Convert.ToInt32(Console.ReadLine());
switch (a)
{

   case 1:
      {

         armor.Print();
         price.Print();
         Console.WriteLine("--------------------");
         Console.WriteLine("Назад-1, Закрыть-2");
         Console.WriteLine("--------------------");
         b = Convert.ToInt32(Console.ReadLine());

         switch (b)
         {
            case 1:
               goto z;
            case 2:
               Environment.Exit(0);
               break;
         }
         break;
      }
   case 2:
      melee.Print();
      price2.Print();
      Console.WriteLine("--------------------");
      Console.WriteLine("Назад-1, Закрыть-2");
      Console.WriteLine("--------------------");
      b = Convert.ToInt32(Console.ReadLine());
      switch (b)
      {

         case 1:
            goto z;
         case 2:
            Environment.Exit(0);
            break;
      }
      break;

   case 3:
      range.Print();
      price3.Print();
      Console.WriteLine("--------------------");
```

```
                Console.WriteLine("Назад-1, Закрыть-2");
                Console.WriteLine("--------------------");
                b = Convert.ToInt32(Console.ReadLine());
                switch (b)
                {

                    case 1:
                        goto z;
                    case 2:
                        Environment.Exit(0);
                        break;
                }
                break;
            case 4:
                Environment.Exit(0);
                break;
        }
    }
  }
}
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

Интернет-ресурс:

1. C# и .Net | Определение классов – Режим доступа: https://metanit.com/sharp/tutorial/3.1.php
2. C# и .Net | Определение интерфейса – Режим доступа: https://metanit.com/sharp/tutorial/3.9.php
3. C# и .Net | Создание конструкторов – Режим доступа: https://metanit.com/sharp/tutorial/3.35.php
4. C# и .Net | Структуры – Режим доступа: https://metanit.com/sharp/tutorial/2.13.php
5. C# и .Net | Типы значений и ссылочные типы – Режим доступа: https://metanit.com/sharp/tutorial/2.16.php