**МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**РОСТОВСКОЙ ОБЛАСТИ**

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РОСТОВСКОЙ ОБЛАСТИ

**«РОСТОВСКИЙ-НА-ДОНУ КОЛЛЕДЖ СВЯЗИ И ИНФОРМАТИКИ»**

КУРСОВОЙ ПРОЕКТ

По дисциплине ОП.05 «Объектно-ориентированное программирование»

Тема: **«**Проектирование библиотеки классов с использованием языка C#»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

РКСИ.К19.09.02.03.ОП.11.18/2850.00ПЗ

Специальность 09.02.03 «Программирование в компьютерных системах»

Выполнил студент \_\_\_\_\_\_\_\_\_\_ Поляков А.В. Группа ПОКС-34

подпись Ф.И.О

Руководитель \_\_\_\_\_\_\_\_\_\_ Лошкарев И.В.

подпись Ф.И.О

Проект защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ «\_\_\_» \_\_\_\_\_\_\_\_\_ 2019 г.

2019 г.

**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**

по дисциплине

**ОП.11 «Объектно-ориентированное программирование»**

студента Поляков А.В.

**Тема «**Проектирование библиотеки классов с использованием языка C#»

**1 Исходные данные**

В качестве предметной области выбрано Научно-просветительное учреждение «Зоопарк»

Дата выдачи задания \_\_\_\_\_\_\_\_ 20\_\_\_ г  Дата сдачи \_\_\_\_\_\_\_\_ .20\_\_г.

Задание рассмотрено на заседании Руководитель проекта

ЦК, протокол №\_\_ от "\_\_\_"\_\_\_\_\_\_\_ 20\_\_\_ г \_\_\_\_\_ Лошкарев И.В.

Подпись Ф.И.О.

Председатель \_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_ "\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г

подпись Ф.И.О.

Задание принял к исполнению "\_\_\_"\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Студент(ка)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**ОТЗЫВ**

**1 Заключение о соответствии курсового проекта (работы) заданной теме**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2 Качество оформление проекта (работы)**

Пояснительная записка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Графическая часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3 Качество работы студента над проектом (работой)**

Соблюдение графика выполнения проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Теоретическая подготовка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Практическая подготовка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Знания современных направлений развития техники\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Умение пользоваться справочной и нормативной документацией\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Инициатива\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Степень самостоятельности\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4 Защита**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

С замечаниями \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание

Оглавление

[Введение 3](#_Toc30187822)

[1 Пояснительная записка 6](#_Toc30187823)

[1.1 Обзор возможностей языка C# и платформы .Net 6](#_Toc30187824)

[1.2 Обзор принципов гибкой разработки ПО 7](#_Toc30187825)

[1.3 Обзор применяемых паттернов архитектуры ПО 14](#_Toc30187826)

[2 Практическая часть 16](#_Toc30187827)

[2.1 Постановка задачи 16](#_Toc30187828)

[2.2 Описание предметной области 16](#_Toc30187829)

[2.3 Проектирование библиотеки классов 17](#_Toc30187830)

[2.4 Реализация библиотеки классов 17](#_Toc30187831)

[3 Заключение 26](#_Toc30187832)

[4 Источники 27](#_Toc30187833)

# **Введение**

ПО или программное обеспечение устанавливается на компьютерную технику или любое интеллектуальное оборудование. Оно необходимо для взаимодействия человека с компьютером.

Программное обеспечение компьютера – это комплекс программ, которые получают, обрабатывают и передают информацию. Последняя впоследствии используется человеком.

ПО используется для создания приложений, игр, браузеров для серфинга в интернете, просмотра фильмов, печати текста и других функций.

Человек может принимать как посредственное, так и прямое отношение к работе над и с ПО.

Из вышеперечисленного следует вывод, что без ПО ни одно интеллектуальное устройство работать с пользователем не будет. Может сложиться впечатление, что программа и ПО это синонимы. Однако есть разница.

**Что такое программа и чем отличается от программного обеспечения**

Программа – это одно приложение, с которым работает пользователь. Программное обеспечение – это комплекс одиночных утилит, которые соединены одной группой для того, чтобы пользователь мог работать с ними для получения глобального результата. В пример можно привести «CCleaner». С помощью этой программы пользователь может почистить компьютер от мусора. А вот операционная система Windows (любая версия) является уже ПО. Стоит отметить, что хоть фактически различие существует, на практике эти понятия обычно считают синонимами.

**Типы и виды ПО с примерами**

Программное обеспечение подразделяется на следующие типы: системное; прикладное; инструментальное. Теперь каждый тип будет разобран по порядку. К системному относятся следующие виды: операционная система Windows, MS DOS, LINUX, Symbian и Android для смартфонов и другие; программы записанные в BIOS материнской платы. Это те утилиты, которые работают с аппаратными средствами. Их называют драйверами; служебное ПО – это такие приложения, которые диагностируют и автоматизируют работу и настраивают систему компьютера. Например, архиваторы, встроенные антивирусы.

К **прикладному** ПО, которое отвечает за исполнение определенных задач поставленных пользователем относятся следующие виды (перечислим распространенные виды):

* Word – текстовый редактор;
* Photoshop, Corel Draw – работа с графикой и изображениями;
* MS Access – работа с базами данных;
* FireFox Mozzila, Google Chrome – браузеры для серфинга в интернет;
* 1С – для ведения систем учета бухгалтерии в организациях;
* Aimp, Kmplayer – видео и аудио проигрыватели;
* FL Studio, Cubase – утилиты для создания собственных музыкальных треков;
* NERO – для записи CD, DVD, Blu-Ray;
* Autocad – программы для 3D и 2D моделирования;
* Aboe Premier, Sony Vegas — видеоредакторы; игровые утилиты.

К **инструментальному** (называют программными средствами) относятся утилиты, с помощью которых создатели предыдущих программ пишут код, придают им внешний облик и работу без багов. Например:

* ассемблеры – превращают обычный код в двоичный;
* компиляторы – переводят текст в двоичный машинный код.

К вышеперечисленным утилитам относятся следующие приложения (приведем для примера, на самом деле их намного больше):

* Microsoft Visual Basic – создает программы в OS Windows;
* Borland C++ Builder – с помощью нее разрабатывали приложения для DOS и Windows.

**Проприетарное**

Это такое программное обеспечение, которое защищено авторскими правами. Только правообладатель сможет изменить исходный код данного ПО, чтобы его улучшить. Только правообладатель может выпускать, и предоставлять право устанавливать его другим для работы с этим ПО. Проприетарное имеет степени защиты авторских прав. Например:

1. Коммерческие. Запрещают распространять утилиты в свободном доступе, модифицировать, бесплатно работать с нею. Оно полностью находится под охраной закона;
2. Другим видом с облегченной степенью защиты является «shareware». Эти приложения приобретаются пользователем бесплатно. Но имеют ограниченный срок действия, после которого предлагается купить полную коммерческую версию;
3. Freeware. Подобен коммерческому, но можно использоваться бесплатно;
4. Существуют коммерческие утилиты с возможностью посмотреть исходный код, но модифицировать его запрещается;
5. Копилефт. Это полностью коммерческие программы, распространяющиеся бесплатны. При чем исходный код открыт с возможностью модификаций и распространения. К таким ПО относятся GNU GPL;
6. BSD License. Программы с такой лицензией распространяются с открытым кодом, но производить модифицированные копии с таким же открытым кодом нельзя.

**Открытое**

Еще один вид ПО – открытое. Это приложения, распространяющиеся с открытым исходным кодом. Его можно изменять, исправлять ошибки, допущенные разработчиками. Такое ПО может быть как платным, так и бесплатным.

**Свободное**

Свободное ПО может распространяться на бесплатной основе. Его можно использовать в институтах, школах, в домашних условиях, в коммерческих и государственных организациях.

**Пиратское**

Незаконным видом программного обеспечения является «пиратское». Данное ПО подразумевает взлом приложения, защищенного авторскими правами. Оно нарушает законы и его производство и использование уголовно наказуемо. На фирмы и пользователей использующие такие программы налагаются штрафы, вплоть до лишения свободы. Взламывается обычно различными «кряками» — утилитами, заточенными под снятие всех ограничений.

# **Пояснительная записка**

## **Обзор возможностей языка C# и платформы .Net**

На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

По сравнению с другими языками C# достаточно молодой, но в то же время он уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 8.0, которая вышла в сентябре 2019 года вместе с релизом .NET Core 3.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

Роль платформы .NET

Когда говорят C#, нередко имеют в виду технологии платформы .NET (Windows Forms, WPF, ASP.NET, Xamarin). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире.

Как-то Билл Гейтс сказал, что платформа .NET - это лучшее, что создала компания Microsoft. Возможно, он был прав. Фреймворк .NET представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

• Поддержка нескольких языков. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), благодаря чему .NET поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в сборку на общем языке CIL (Common Intermediate Language) - своего рода ассемблер платформы .NET. Поэтому мы можем сделать отдельные модули одного приложения на отдельных языках.

• Кроссплатформенность. .NET является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент .NET Core поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

• Мощная библиотека классов. .NET представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов .NET.

• Разнообразие технологий. Общеязыковая среда исполнения CLR и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология ADO.NET и Entity Framework Core. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и UWP, для создания более простых графических приложений - Windows Forms. Для разработки мобильных приложений - Xamarin. Для создания веб-сайтов - ASP.NET и т.д.

Также еще следует отметить такую особенность языка C# и фреймворка .NET, как автоматическая сборка мусора. А это значит, что нам в большинстве случаев не придется, в отличие от С++, заботиться об освобождении памяти. Вышеупомянутая общеязыковая среда CLR сама вызовет сборщик мусора и очистит память.

## **Обзор принципов гибкой разработки ПО**

Методология разработки софта — организация труда, включающая идеологические принципы, план, контроль над процессами, подход к сотрудникам. Выделим 12 видов:

Waterfall — традиционный подход.

RUP (Rational Unified Process) — рациональный.

Agile — общая методология гибкой разработки.

Crystal Clear — подход с уравниванием разработчиков в коллективе.

Spiral — спиральный метод.

DSDM (Dynamic Systems Development Model) — динамическая модель.

FDD (Feature Driven Development) — методология, рассматривающая будущие изменения.

JAD (Joint Application Development) — ориентированный на пользователя подход.

RAD (Rapid Application Development) — модель быстрой разработки.

Scrum — концепция работы в условиях сорванных сроков и идеологического кризиса.

XP (Extreme Programming) — экстремальная разработка в динамической среде.

LD (Lean Development) — метод, предполагающий бережное отношение ко всем участникам процесса.

**Waterfall**

Модель Waterfall относится к классическому пониманию разработки ПО. Весь процесс является жестким и линейным, имеет четкие цели для каждого этапа, новая фаза начинается по завершению предыдущей, нет возврата назад. Преимущества водопадной методологии — децентрализация и строгий контроль над сроками и качеством исполнения.

На практике Waterfall часто не оправдывает ожиданий, поскольку игнорирует динамические изменения. Так, после тестирования очень сложно откатить процесс и заложить функции, не учтенные на стадии разработки. Waterfall неэффективен ещё и потому, что предполагает временные простои сотрудников в рамках одного проекта. Тестирование проводится только в конце разработки, хотя проблемы, найденные на этом этапе — это дорогостоящие исправления.

**RUP**

RUP — итеративный подход, который решает проблемы, которые есть у Waterfall. Чем хорош RUP:

Учитывает изменяющиеся требования. Как бы ни был хорош руководитель проекта, учесть всё в начале невозможно.

Интеграция функций происходит постепенно, то есть каждая «деталь» проходит цикл разработки, проверки и внедрения в проект. Как следствие, снижаются риски и стоимость производства.

Ранний выпуск продукта. ПО выходит с уменьшенной функциональностью, чтобы занять нишу на рынке и противостоять конкурентам, после чего обрастает «мясом».

Повторное использование. При наращивании функциональности проще выделить типовые решения, которые сократят разработку.

Постоянное обучение. Из-за частых итераций разработчики не имеют больших пауз между доработкой кода, поэтому профессиональный рост происходит плавно и безболезненно.

Постоянное улучшение продукта. Итерации позволяют оценить проект не только с точки зрения соответствия плану и ТЗ, но и найти пути увеличения эффективности и качества продукта.

**Agile**

Agile — метод гибкой разработки программного обеспечения, предполагающий большое количество итераций. Документ Agile Manifesto описывает 4 идей и 12 принципов гибкого подхода, коротко его можно описать всего двумя пунктами:

Неформальные отношения важнее задокументированных. То есть устные договоренности между сотрудниками, между заказчиком и исполнителем важнее всего, что отражено в планах, договорах и техническом задании. Иначе говоря, клиент всегда прав.

Работающий продукт — главная оценка прогресса. Важны не инструменты, решения, производительность и изящество, а тот факт, что все запланированные возможности реализованы.

Несмотря на недостатки, Agile стала фундаментальной концепцией для разработки ПО и нашла отражение в других методологиях, речь о которых пойдет далее.

**Crystal Clear**

Методология, созданная для небольших коллективов из 6−10 сотрудников. Также поддерживает принципы гибкой разработки, но имеет чуть больше конкретики. Основная идея, которая и заключена в названии — каждая команда является набором людей с разным уровнем знаний, разными умениями и опытом.

Именно поэтому нет универсального подхода для разработки софта, он должен определяться в процессе общения внутри группы. Там же назначаются роли, инструменты, стандарты. Затем группа принимается за единицу и те же самые вопросы решаются на уровень выше, пока иерархия не дойдет до заказчика.

**Spiral**

Модель спирального жизненного цикла — это сложная организация жизненного цикла ПО, которая фокусируется на раннем выявлении и уменьшении проектных рисков. Разработка начинается в небольшом масштабе, решаются локальные задачи, оцениваются риски и пути их уменьшения. Следующий шаг охватывает более комплексные задачи — следующий виток спирали.

Преимущество подхода не в увеличении скорости разработки, а в снижении уровня возникновения рисков. Успешность спирального метода зависит от добросовестного, внимательного и компетентного управления, а размер проекта не имеет принципиального значения.

**DSDM**

Модель развития динамических систем была разработана в Великобритании в середине 1990-х годов и является эволюционным развитием быстрой разработки приложений (RAD). Основная идея стандартная: при планировании в самом начале невозможно понимать всех тонкостей разработки, поэтому весь процесс — исследовательская работа.

В DSDM тоже присутствует деление на команды, в каждой из которых есть уполномоченный для принятия стратегических решений. В процессе могут участвовать все заинтересованные стороны: пользователи, разработчики, заказчики, руководители. Тестирование проводится на протяжении всего жизненного цикла.

**FDD**

FDD — процесс для обеспечения масштабируемости и повторяемости, при этом поощряющий творчество и инновации. Вот основные принципы:

Разработка каждого крупного проекта должна иметь системность.

Процессы должны быть простыми и проработанными.

Ценность и логичность процесса должна быть ясна каждому члену команды.

Предпочтение отдаётся коротким итеративным циклам разработки. Это уменьшает количество ошибок и позволяет быстрее наращивать функциональность.

FDD регламентирует время, которое должно затрачиваться на каждый из процессов. Организационной деятельности в цикле должна занимать не более 23−25%, в то время как на непосредственную разработку, сборку и тестирование функций необходимо тратить 75−77% времени.

**JAD**

JAD — это методология, нацеленная на максимальную занятость в разработке конечного пользователя. Происходит это посредством встреч и проведения совместных семинаров. JAD была придумана в 1970-х годах сотрудниками IBM и нацелена на бизнес в целом. Однако со временем данная концепция стала успешно применяться и для разработки программного обеспечения.

В отличие от подхода Waterfall, JAD приводит к сокращению времени разработки, большей удовлетворенности клиентов и экономии средств на изучении рынка. С другой стороны, это требует большой клиентской выборки и необходимости разработчиков работать не со строгими требованиями ТЗ, а с постоянно меняющимся мнением.

**RAD**

RAD — методология, которая во главу угла ставит скорость и удобство разработки. Одно из главных условий — использование языка быстрой разработки. Это название абстрактного языка программирования, с помощью которого программист способен решать задачи быстрее, чем с представителями третьего поколения (C / C ++, Pascal или Fortran). Вот ещё несколько пунктов концепции:

Использование фокус-групп для сбора требований.

Прототипирование и пользовательское тестирование конструкций.

Повторное использование программных компонентов.

Использование плана, не включающего переработку, или дизайн следующей версии продукта.

Проведение неформальных совещаний по запросу одной из сторон.

RAD предполагает использование целого комплекса инструментов помимо языка быстрой разработки: системы сбора требований, среды разработки, фреймворки, программы для группового общения, ПО для тестирования.

**Scrum**

Scrum — гибкий метод управления проектами, целью которого является повышение производительности труда в командах, ранее парализованных более тяжелыми методологическими процессами. В основе концепции лежат «спринты». Спринт — короткая итерация, строго ограниченная по времени (обычно 2−4 недели). В это время минимизируется длительность совещаний, но увеличивается их частота (они называются «схватками»).

Благодаря этому контроль за выполнением становится более гибким, а разработчики быстрее реагируют на возникающие проблемы. Традиционное планирование отходит на второй план, его место занимает журнал спринтов.

**XP**

Экстремальное программирование — возможность вести разработку в условиях постоянно меняющихся требований. Вот несколько признаков:

Игра в планирование. В начале проекта есть только приблизительный план, после каждой итерации его чёткость возрастает.

Высокая частота релизов. Новая версия продукта имеет незначительные изменения по сравнению с предыдущей, но время на выпуск при этом минимально.

Контакт с клиентом. Для удовлетворения требований конечной аудитории необходимо оперативное реагирование на замечания и пожелания.

Рефакторинг. Улучшение качества кода без уменьшения функциональности.

Стандарт выполнения кода. Или применяются общие правила, или разногласия в оформлении не подлежат обсуждению и критике.

Коллективная ответственность. Несмотря на то, что каждый член команды выполняет свой участок работ, за код в целом отвечает весь коллектив.

**LD**

Бережливая разработка ПО — ещё одно ответвление гибкой методологии, предполагающее сохранение высокого морально-функционального состояния разработчиков. Это выражается в:

Поощрении сотрудников за успешную работу.

Изменении текущих задач только по мере необходимости или по запросу заказчика.

Строгом выполнении плана: всё, что сверх — считается потерями времени и ресурсов.

Внедрении общей концепции «Мыслить широко, делать мало, ошибаться быстро, учиться стремительно».

В условиях короткого дайджеста трудно раскрыть все преимущества и недостатки методологий, показать эффективные области применения. О наиболее актуальных на сегодняшний день концепциях мы поговорим отдельно. О каких именно? Оставляйте свои пожелания в комментариях.

**Немного истории**

Кто придумал паттерны? Это хороший, но не совсем корректный вопрос, так как паттерны не придумывают, а, скорее, «открывают». Это не какие-то супер-оригинальные решения, а наоборот, часто встречающиеся, типовые решения одной и той же проблемы.

Концепцию паттернов впервые описал Кристофер Александер в книге «Язык шаблонов. Города. Здания. Строительство». В книге описан «язык» для проектирования окружающей среды, единицы которого — шаблоны (или паттерны, что ближе к оригинальному термину patterns) — отвечают на архитектурные вопросы: какой высоты сделать окна, сколько этажей должно быть в здании, какую площадь в микрорайоне отвести под деревья и газоны.

Идея показалась заманчивой четвёрке авторов: Эриху Гамме, Ричарду Хелму, Ральфу Джонсону, Джону Влиссидесу. В 1995 году они написали книгу «Приемы объектно-ориентированного проектирования. Паттерны проектирования», в которую вошли 23 паттерна, решающие различные проблемы объектно-ориентированного дизайна. Название книги было слишком длинным, чтобы кто-то смог всерьёз его запомнить. Поэтому вскоре все стали называть её «book by the gang of four», то есть «книга от банды четырёх», а затем и вовсе «GoF book».

С тех пор были найдены десятки других объектных паттернов. «Паттерновый» подход стал популярен и в других областях программирования, поэтому сейчас можно встретить всевозможные паттерны и за пределами объектного проектирования.

## **Обзор применяемых паттернов архитектуры ПО**

Паттерн проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерны часто путают с алгоритмами, ведь оба понятия описывают типовые решения каких-то известных проблем. Но если алгоритм — это чёткий набор действий, то паттерн — это высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Если привести аналогии, то алгоритм — это кулинарный рецепт с чёткими шагами, а паттерн — инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации.

Из чего состоит паттерн

Описания паттернов обычно очень формальны и чаще всего состоят из таких пунктов:

проблема, которую решает паттерн;

мотивации к решению проблемы способом, который предлагает паттерн;

структуры классов, составляющих решение;

примера на одном из языков программирования;

особенностей реализации в различных контекстах;

связей с другими паттернами.

**Зачем знать паттерны?**

Вы можете вполне успешно работать, не зная ни одного паттерна. Более того, вы могли уже не раз реализовать какой-то из паттернов, даже не подозревая об этом.

Но осознанное владение инструментом как раз и отличает профессионала от любителя. Вы можете забить гвоздь молотком, а можете и дрелью, если сильно постараетесь. Но профессионал знает, что главная фишка дрели совсем не в этом. Итак, зачем же знать паттерны?

Проверенные решения. Вы тратите меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений вы смогли бы додуматься и сами, но многие могут быть для вас открытием.

Стандартизация кода. Вы делаете меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.

Общий программистский словарь. Вы произносите название паттерна, вместо того, чтобы час объяснять другим программистам, какой крутой дизайн вы придумали и какие классы для этого нужны.

**Существует три основных группы паттернов**

**Порождающие**

Отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов.

**Структурные**

Отвечают за построение удобных в поддержке иерархий классов.

**Поведенческие**

Решают задачи эффективного и безопасного взаимодействия между объектами программы.

# **Практическая часть**

## **Постановка задачи**

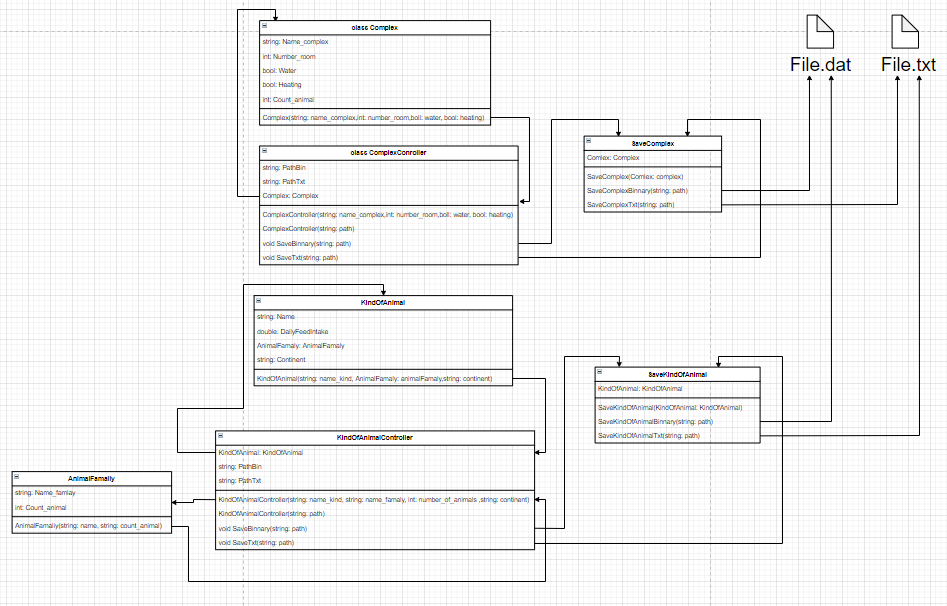
Разработка архитектуры программы для обработки информации по заданным исходным данным.

Тема: «Зоопарк»

## **Описание предметной области**

В качестве предметной области выбрано Научно-просветительное учреждение «Зоопарк» в задачи которого входят демонстрация разнообразия животного мира, распространение знаний о природе, пропаганда охраны животных и сохранение генофонда редких и исчезающих видов животных.  
При создании объекта комплекс его можно сохранить, а также загрузить данные. На данный момент объект может только сохранять и загружать данные в бинарном и текстовом виде. Аналогичные действия может совершать объект «Вид животного». В дальнейшем решение этого проекта можно расширять.

## **Проектирование библиотеки классов**



## **Реализация библиотеки классов**

Класс «Complex» его суть заключается выполнение роли модели.

public class Complex

{

Объявление полей:

/// <summary>

/// Название комплекса.

/// </summary>

public string Name\_complex { get; }

/// <summary>

/// Номер помещения.

/// </summary>

public int Number\_room { get; }

/// <summary>

/// Наличие водоема.

/// </summary>

public bool Water { get; set; }

/// <summary>

/// Наличие отопления.

/// </summary>

public bool Heating { get; set; }

/// <summary>

/// Колличество животных в помещении

/// </summary>

public int Count\_animal { get; set; }

/// <summary>

/// Создание комалекса

/// </summary>

/// <param name="name\_complex">Название комплекса</param>

/// <param name="number\_room">Номер помещения</param>

/// <param name="water">Наличие водоема</param>

/// <param name="heating">Наличие отопления</param>

/// <param name="count\_animal">Колличество животных в помещении</param>

Объявление методов:

public Complex(string name\_complex, int number\_room, bool water, bool heating,int count\_animal) – Конструктор класса Complex

{

Name\_complex = name\_complex;

Number\_room = number\_room;

Water = water;

Heating = heating;

Count\_animal = count\_animal;

}

}

Класс «ComplexController» его суть заключается в выполнение роли контролера класса «Complex».

/// <summary>

/// Контроллер комплекса

/// </summary>

public class ComplexController

{

Объявление полей:

/// <summary>

/// Путь бинарного файла

/// </summary>

public string PathBin { get; set; }

/// <summary>

/// Путь текстового файла

/// </summary>

public string PathTxt { get; set; }

/// <summary>

/// Комплекс

/// </summary>

public Complex Complex { get; set; }

/// <summary>

/// Создание Комплекса

/// </summary>

/// <param name="complex\_name">Название комплекса</param>

/// <param name="number\_room">Номер помещения</param>

/// <param name="water">Наличие водоема</param>

/// <param name="heating">Наличие отопления</param>

/// <param name="count\_animal">Колличество животных в помещении</param>

Объявление методов:

public ComplexController(string complex\_name, int number\_room,bool water,bool heating,int count\_animal) – Конструктор класса ComplexController

{

#region Проверка данных

if (string.IsNullOrWhiteSpace(complex\_name))

{

throw new ArgumentNullException("Имя комплекса не может быть пустым", nameof(complex\_name));

}

if(number\_room<0)

{

throw new ArgumentException("Номер помещения не может быть отрицательным либо null", nameof(number\_room));

}

if(water!=true||water!=false)

{

throw new ArgumentException("Неверно введенные данные", nameof(water));

}

if (heating != true || heating != false)

{

throw new ArgumentException("Неверно введенные данные", nameof(heating));

}

if(count\_animal<0)

{

throw new ArgumentException("Колличество животных не может быть отрицательным", nameof(count\_animal));

}

#endregion

Complex = new Complex(complex\_name,number\_room,water,heating,count\_animal);

}

/// <summary>

/// Сохранение в бинарном виде

/// </summary>

/// <param name="path">Путь</param>

public void SaveBinnary(string path) – Создание объекта класса SaveComplex

{

if (string.IsNullOrWhiteSpace(path))

{

throw new ArgumentNullException("Путь не может быть пустым или null", nameof(path));

}

PathBin = path;

var sc = new SaveComplex(Complex);

sc.SaveComplexBinnary(path);

}

/// <summary>

/// Сохранение в текстовом виде

/// </summary>

/// <param name="path"></param>

public void SaveTxt(string path) – Создание объекта класса SaveComplex

{

if (string.IsNullOrWhiteSpace(path))

{

throw new ArgumentNullException("Путь не может быть пустым или null", nameof(path));

}

PathTxt = path;

var sc = new SaveComplex(Complex);

sc.SaveComplexTxt(path);

}

/// <summary>

/// Создание комплекса

/// </summary>

public ComplexController(string path) – Метод открытия файла если он существует

{

if (string.IsNullOrWhiteSpace(path))

{

throw new ArgumentNullException("Путь не может быть пустым или null", nameof(path));

}

var formatter = new BinaryFormatter();

using (FileStream fstream = new FileStream($"{path}", FileMode.Open))

{

if (formatter.Deserialize(fstream) is Complex complex)

{

Complex = complex;

}

}

// TODO: Что то с лоад если не открывается

}

}

Класс «KindOfAnimal» его суть заключается выполнение роли модели.

public class KindOfAnimal

{

Объявление полей:

/// <summary>

/// Название вида

/// </summary>

public string Name\_kind { get; set; }

// TODO: Суточное потребления корма

/// <summary>

/// Суточное потребление корма

/// </summary>

public double DailyFeedIntake { get { return DailyFeedIntake; }set { DailyFeedIntake = 320\* Math.Pow(100, 0.75); } }

/// <summary>

/// Название симейства

/// </summary>

public AnimalFamily AnimalFamily { get; set; }

/// <summary>

/// Континент обитания

/// </summary>

public string Continent { get; set; }

/// <summary>

/// Создание вида животного

/// </summary>

/// <param name="name\_kind">Название вида</param>

/// <param name="animalFamily">Названия семейства</param>

/// <param name="continent">Континент обитания</param>

Объявление методов:

public KindOfAnimal(string name\_kind, AnimalFamily animalFamily,string continent) – Конструктор класса KindOfanimall

{

Name\_kind = name\_kind;

AnimalFamily = animalFamily;

Continent = continent;

}

}

Класс «KindOfAnimalController» его суть заключается в выполнение роли контролера класса «KIndOfAnimal».

/// <summary>

/// Вид животного

/// </summary>

public class KindOfAnimalController

{

Объявление полей:

/// <summary>

/// Название вида

/// </summary>

public KindOfAnimal KindOfAnimal { get; set; }

/// <summary>

/// Путь бинарного файла

/// </summary>

public string PathBin { get; set; }

/// <summary>

/// Путь к текстовому файлу

/// </summary>

public string PathTxt { get; set; }

/// <summary>

/// Создание вида животного

/// </summary>

/// <param name="name\_kind">Название вида</param>

/// <param name="name\_famaly">Название семейства</param>

/// <param name="number\_of\_animals">Численность</param>

/// <param name="continent">Континент</param>

Объявление методов:

public KindOfAnimalController(string name\_kind, string name\_famaly, int number\_of\_animals, string continent) – Конструктор класса KindOfAnimalController

{

#region Проверка данных

if (string.IsNullOrWhiteSpace(name\_kind))

{

throw new ArgumentNullException("Имя вида не может быть пустым или null", nameof(name\_kind));

}

if (string.IsNullOrWhiteSpace(name\_famaly))

{

throw new ArgumentNullException("Имя семейства не может быть пустым или null", nameof(name\_famaly));

}

if(number\_of\_animals<0)

{

throw new ArgumentException("Численность животных не может быть отрицательной или null", nameof(number\_of\_animals));

}

if(string.IsNullOrWhiteSpace(continent))

{

throw new ArgumentNullException("Имя континента не может быть пустым или null", nameof(continent));

}

#endregion

AnimalFamily animalfamyly = new AnimalFamily(name\_famaly, number\_of\_animals);

KindOfAnimal = new KindOfAnimal(name\_kind, animalfamyly, continent);

}

/// <summary>

/// Создание вида животного с помощью чтения бинарного файла

/// </summary>

/// <param name="path"></param>

public KindOfAnimalController(string path) - Метод открытия файла если он существует

{

if (string.IsNullOrWhiteSpace(path))

{

throw new ArgumentNullException("Путь не может быть пустым или null", nameof(path));

}

var formatter = new BinaryFormatter();

using (FileStream fstream = new FileStream($"{path}", FileMode.Open))

{

if (formatter.Deserialize(fstream) is KindOfAnimal kindOfAnimal)

{

KindOfAnimal = kindOfAnimal;

}

}

// TODO: Что то с лоад если не открывается

}

/// <summary>

/// Сохранение в бинарном виде

/// </summary>

/// <param name="path">Путь</param>

public void SaveBinnary(string path) – Создание объекта класса SaveKindOfAnimal

{

if (string.IsNullOrWhiteSpace(path))

{

throw new ArgumentNullException("Путь не может быть пустым или null", nameof(path));

}

PathBin = path;

var sc = new SaveKindAnimal(KindOfAnimal);

sc.SaveKindOfAnimalBinnary(path);

}

/// <summary>

/// Сохранение в текстовом виде

/// </summary>

/// <param name="path"></param>

public void SaveTxt(string path) – Создание объекта класса SaveKindOfAnimal

{

if (string.IsNullOrWhiteSpace(path))

{

throw new ArgumentNullException("Путь не может быть пустым или null", nameof(path));

}

PathTxt = path;

var sc = new SaveKindAnimal(KindOfAnimal);

sc.SaveKindOfAnimalTxt(path);

}

}

Класс «AnimalFamaly» описывает семейство животного

/// <summary>

/// Семейство животных

/// </summary>

public class AnimalFamily

{

Объявление полей:

/// <summary>

/// Название семейства.

/// </summary>

public string Name\_famaly { get; set; }

/// <summary>

/// Численность.

/// </summary>

public int Count\_animal { get; set; }

/// <summary>

/// Создание семейства.

/// </summary>

/// <param name="name">Название семейства</param>

/// <param name="count\_animal">Численность</param>

Объявление методов:

public AnimalFamily(string name,int count\_animal) – Создание объекта класса AnimalFamaly

{

Name\_famaly = name;

Count\_animal = count\_animal;

}

}

Класс «SaveComplex» роль реализации записи объекта с разным расширением.

/// <summary>

/// Сохранение данных комплекса

/// </summary>

public class SaveComplex

{

Объявление полей:

/// <summary>

/// Комплекс

/// </summary>

public Complex Complex { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="complex">Комплекс</param>

Объявление методов:

public SaveComplex(Complex complex) – Конструктор класса SaveComplex

{

Complex = complex;

}

/// <summary>

/// Сохранение в бинарном виде

/// </summary>

/// <param name="path">Путь</param>

public void SaveComplexBinnary(string path) – Сохранение файла в бинарном виде

{

var formatter = new BinaryFormatter();

using (FileStream fstream = new FileStream($"{path}", FileMode.OpenOrCreate))

{

formatter.Serialize(fstream, Complex);

}

}

/// <summary>

/// Сохранение в текстовом виде

/// </summary>

/// <param name="path">Путь</param>

public void SaveComplexTxt(string path) – Сохранение файла в текстовом виде

{

using(var sw = new StreamWriter($"{path}",true,System.Text.Encoding.Default))

{

sw.WriteLine(Complex.Name\_complex);

sw.WriteLine(Complex.Number\_room);

sw.WriteLine(Complex.Water);

sw.WriteLine(Complex.Heating);

sw.WriteLine(Complex.Count\_animal);

}

}

}

Класс «SaveKindOfAnimal» роль реализации записи объекта с разным расширением.

/// <summary>

/// Сохранение вида животного

/// </summary>

public class SaveKindAnimal

{

Объявление полей:

/// <summary>

/// Вид животного

/// </summary>

public KindOfAnimal KindOfAnimal { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="kindOfAnimal">Вид животного</param>

/// <param name="kindOfAnimal">Вид животного</param>

Объявление методов:

public SaveKindAnimal(KindOfAnimal kindOfAnimal) – Конструктор класса SaveKindOfAnimal

{

KindOfAnimal = kindOfAnimal;

}

/// <summary>

/// Сохранение в бинарном виде

/// </summary>

/// <param name="path">Путь</param>

public void SaveKindOfAnimalBinnary(string path) – Сохранение файла в бинарном виде

{

var formatter = new BinaryFormatter();

using (FileStream fstream = new FileStream($"{path}", FileMode.OpenOrCreate))

{

formatter.Serialize(fstream, KindOfAnimal);

}

}

/// <summary>

/// Сохранение в текстовом виде

/// </summary>

/// <param name="path">Путь</param>

public void SaveKindOfAnimalTxt(string path) – Сохранение файла в текстовом виде

{

using (var sw = new StreamWriter($"{path}", true, System.Text.Encoding.Default))

{

sw.WriteLine(KindOfAnimal.Name\_kind);

sw.WriteLine(KindOfAnimal.AnimalFamily.Name\_famaly);

sw.WriteLine(KindOfAnimal.AnimalFamily.Count\_animal);

sw.WriteLine(KindOfAnimal.Continent);

}

}

}

# **Заключение**

В данной курсовой работе были рассмотрены такие темы как возможности языка C#, принципы разработки ПО и о применяемых паттернах в архитектуре ПО. Cледует отметить то, что изучение языков программирования очень помогает развивать логические и аналитические навыки человека, которые так необходимы для специалиста по защите информации.

Инструментальной средой разработки программ стала MS Visual Studio 2019

По окончанию курсовой работы можно сказать, что мной были изучены и закреплены следующие задачи:

* ставить задачу и разрабатывать алгоритм её решения
* Понимание сущности и социальной значимости своей будущей профессии, проявлять к ней устойчивый интерес
* Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
* Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием.

# **Источники**

Мартин Роберт "Принципы, паттерны и методики гибкой разработки на языке C#".

<https://metanit.com/>

<https://refactoring.guru/ru>

[http://composs.ru/](http://composs.ru/chto_takoe_programmnoe_obespechenie/)

<https://ru.wikipedia.org/>

<https://geekbrains.ru/posts/methodologies>

<https://docs.microsoft.com/ru-ru/dotnet/csharp/>