**Введение**

На учебную практику была поставлена задача разработать программный продукт на тему: «Консольное приложение «Страховая компания».

Цель проекта заключается в создании приложения для упрощения работы сотрудников страховых компаний.

Создаваемая программа будет рассчитана на пользователей, обслуживающих клиентов в страховых фирмах.

Далее приведем краткое описание разделов пояснительной записки.

Первый раздел носит название “Анализ предметной области и формулировка требований к программе”. В нем вы сможете ознакомиться с постановкой задачи, которая включает в себя: исследование предметной области поставленной задачи, определение ее организационно-экономической сущности. Также в этом разделе вы сможете узнать о том, как данная задача решается в настоящее время. Все входные и выходные данные тоже будут описаны в первом разделе. В подразделе “Инструменты разработки” будет рассмотрена среда, в которой создается данный проект.

В разделе “Проектирование задачи” будут рассмотрены основные аспекты разработки программного продукта. Здесь можно будет узнать о вариантах использования программного продукта и тестах.

“Построение программы” – это третий раздел пояснительной записки, в котором описываются все классы, их атрибуты и методы, которые будут использованы при реализации данного приложения. В этом разделе будут четко описаны функции пользователя и их структура.

Четвертый раздел – “Тестирование”. В нем будет описано полное и функциональное тестирование данной программы, т.е. будет оттестирован каждый пункт меню, каждая операция, которая выполняется приложением. Будут смоделированы все возможные действия пользователя при работе с программой, начиная от запуска до выхода.

В разделе “Применение” будет описано назначение, область применения, среда функционирования программного продукта.

“Заключение” будет содержать краткую формулировку задачи, результаты проделанной работы, описание использованных методов и средств, описание степени автоматизации процессов на различных этапах разработки.

В “Списке используемых источников” будет приведен список используемых при разработке источников.

В приложениях к пояснительной записке будет приведен листинг программы с необходимыми комментариями, а также диаграммы, связанные с разработкой ПП.

**1. Анализ предметной области и формулировка требований к программе**

**1. 1 Исследование предметной области:**

**1. 1. 1 Организационно-экономическая сущность задачи:**

**Наименование задачи:** Консольное приложение «Страховая компания»;

**Цель разработки:** Создание приложения для упрощения работы сотрудников страховых компаний.

**Назначение:** Данный программный продукт разрабатывается для пользователей, обслуживающих клиентов в страховых фирмах.

**Периодичность использования:** любое время;

**Источники и способы получения данных:** ручной ввод данных;

**1. 1. 2 Функциональные требования**

Описание перечня функций и задач, которые должен выполнять будущий ПП:

**Гость:**

1. Выбор раздела «Филиалы»;
2. Выбор раздела «Виды страхования»;
3. Выбор раздела «Договоры»;
4. Выход из программы;
5. Показ существующих филиалов;
6. Добавление филиалов;
7. Удаление филиалов;
8. Выход в главное меню;
9. Показ существующих видов страхования;
10. Добавление вида страхования;
11. Удаление вида страхования;
12. Показ существующих договоров;
13. Добавление договора;
14. Удаление договора;

**1. 1. 3 Описание процессов с входной, выходной и условно-постоянной информацией**

Таблица 1 — описание процессов с входной, выходной и условно-постоянной информацией

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Категория пользователей | Наименование процесса | Краткое описание алгоритма выполнения процесса | Входная информация | Выходная информация | Условно-постоянная информация |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | Гость | Выбор раздела «Филиалы» | При нажатии кнопки «1» открывается раздел «Филиалы» | Ручной ввод информации | Меню для работы | Отсутствует |
| 2 | Гость | Выбор раздела «Виды страхования» | При нажатии кнопки «2» открывается раздел «Виды страхования» | Ручной ввод информации | Меню для работы | Отсутствует |
| 3 | Гость | Выбор раздела «Договоры» | При нажатии кнопки «3» открывается раздел «Договоры» | Ручной ввод информации | Меню для работы | Отсутствует |
| 4 | Гость | Выход из программы | При нажатии кнопки «4» осуществляется выход из программы | Ручной ввод информации | Отсутствует | Отсутствует |
| 5 | Гость | Просмотр существующих филиалов | В меню филиалов при нажатии кнопки «1» будет осуществляться просмотр филиалов | Ручной ввод информации | Показ существующих филиалов | Данные файла с информацией о филиалах |
| 6 | Гость | Добавление филиалов | В меню филиалов при нажатии кнопки «1» будет осуществляться добавление филиала | Ручной ввод информации | Добавление информации о филиале в файл | Отсутствует |
| 7 | Гость | Удаление филиалов | В меню филиалов при нажатии кнопки «1» будет осуществляться удаление филиала | Ручной ввод информации | Удаление информации о филиале из файла | Отсутствует |

Продолжение: Таблица 1 — описание процессов с входной, выходной и условно-постоянной информацией

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8 | Гость | Выход в меню | В любом под-меню при нажатии кнопки «4» будет осуществляться выход в главное меню | Ручной ввод информации | Показ главного меню | Отсутствует |
| 9 | Гость | Просмотр существующих видов страхования | В меню видов страхования при нажатии кнопки «1» будет осуществляться просмотр видов страхования | Ручной ввод информации | Показ существующих видов страхования | Данные файла с информацией о видах страхования |
| 10 | Гость | Добавление видов страхования | В меню видов страхования при нажатии кнопки «2» будет осуществляться добавление видов страхования | Ручной ввод информации | Добавление информации о виде страхования в файл | Отсутствует |
| 11 | Гость | Удаление видов страхования | В меню видов страхования при нажатии кнопки «3» будет осуществляться удаление видов страхования | Ручной ввод информации | Удаление информации о виде страхования | Отсутствует |
| 12 | Гость | Просмотр существующих договоров | В меню договор при нажатии кнопки «1» будет осуществляться просмотр договоров | Ручной ввод информации | Показ существующих договоров | Данные файла с информацией о договорах |
| 13 | Гость | Добавление договоров | В меню договор при нажатии кнопки «2» будет осуществляться добавление договоров | Ручной ввод информации | Добавление информации о договоре в файл | Отсутствует |
| 14 | Гость | Удаление договоров | В меню договор при нажатии кнопки «3» будет осуществляться удаление договоров | Ручной ввод информации | Удаление информации о договоре | Отсутствует |

**1. 2 Инструменты разработки**

Для разработки проекта будет выбрана среда разработки Visual Studio 2022, которая является наиболее актуальной средой для создания программных продуктов данного типа.

Разработка будет производиться на языке программирования C#.

Иные инструменты, используемые при разработке и написании сопутствующей документации:

- WEB-ресурс DRAW.IO – будет использоваться для создания графической части и разработки UML-диаграмм;

- Microsoft Office Word – для написания документации к программному продукту;

Разработка проекта будет происходить на компьютере со следующими параметрами:

- процессор intel core i5-3230M 2.6 GHz

- объем оперативной памяти 4 GB;

- объем места на жестком диске 1 TB;

- видеокарта GT 720M;

- ОС Windows 8.

**1. 2. 1 Выбор модели, метода и подхода разработки программы.**

Таблица 2 — Выбор модели жизненного цикла на основе характеристик команды разработчиков:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Являются ли требования к проекту легко определимыми и реализуемыми? | Да | Да | Да | Нет | Нет | Нет |
| 2. | Могут ли требования быть сформулированы в начале ЖЦ? | Да | Да | Да | Да | Нет | Нет |
| 3. | Часто ли будут изменяться требования на протяжении ЖЦ? | Нет | Нет | Нет | Нет | Да | Да |
| 4. | Нужно ли демонстрировать требования с целью их определения? | Нет | Нет | Да | Нет | Да | Да |
| 5. | Требуется ли проверка концепции программного средства или системы? | Нет | Нет | Да | Нет | Да | Да |
| 6. | Будут ли требования изменяться или уточняться с ростом сложности системы (программного средства) в ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Нужно ли реализовать основные требования на ранних этапах разработки? | Нет | Нет | Да | Да | Да | Да |
| Всего: | | 5 | 5 | 4 | 6 | 2 | 2 |

Таблица 3 — Выбор модели жизненного цикла на основе характеристик команды разработчиков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Являются ли проблемы предметной области проекта новыми для большинства разработчиков? | Нет | Нет | Нет | Нет | Да | Да |
| 2. | Являются ли инструментальные средства, используемые в проекте, новыми для большинства разработчиков? | Да | Да | Нет | Нет | Нет | Да |
| 3. | Изменяются ли роли участников проекта на протяжении ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Является ли структура процесса разработки более значимой для разработчиков, чем гибкость? | Да | Да | Нет | Да | Нет | Нет |
| 5. | Важна ли легкость распределения человеческих ресурсов проекта? | Да | Да | Да | Да | Нет | Нет |
| 6. | Приемлет ли команда разработчиков оценки, проверки, стадии разработки? | Да | Да | Нет | Да | Да | Да |
| Всего: | | 4 | 4 | 5 | 4 | 3 | 2 |

Таблица 4 — Выбор модели жизненного цикла на основе характеристик коллектива пользователей

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Будет ли присутствие пользователей ограничено в ЖЦ разработки? | Да | Да | Нет | Да | Нет | Да |
| 2. | Будут ли пользователи оценивать текущее состояние программного продукта (системы) в процессе разработки? | Нет | Нет | Нет | Да | Да | Да |
| 3. | Будут ли пользователи вовлечены во все фазы ЖЦ разработки? | Нет | Нет | Да | Нет | Да | Нет |
| 4. | Будет ли заказчик отслеживать ход выполнения проекта? | Нет | Нет | Нет | Нет | Да | Да |
| Всего: | | 1 | 1 | 1 | 2 | 3 | 3 |

Таблица 5 — Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Разрабатывается ли в проекте продукт нового для организации направления? | Нет | Нет | Нет | Да | Да | Да |
| 2. | Будет ли проект являться расширением существующей системы? | Да | Да | Да | Да | Нет | Нет |
| 3. | Будет ли проект крупно- или среднемасштабным? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Ожидается ли длительная  эксплуатация продукта? | Да | Да | Нет | Да | Нет | Да |
| 5. | Необходим ли высокий уровень надежности продукта проекта? | Нет | Да | Нет | Да | Нет | Да |
| 6. | Предполагается ли эволюция продукта проекта в течение ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Велика ли вероятность изменения системы (продукта) на этапе сопровождения? | Нет | Нет | Нет | Да | Да | Да |
| 8. | Является ли график сжатым? | Нет | Нет | Да | Да | Да | Да |
| 9. | Предполагается ли повторное использование компонентов? | Нет | Нет | Да | Да | Да | Да |
| 10. | Являются ли достаточными  ресурсы (время, деньги, инструменты, персонал)? | Нет | Нет | Нет | Нет | Да | Да |
| Всего: | | 5 | 4 | 7 | 4 | 5 | 4 |

**Общие итоги:**

Каскадная: 15

V-образная: 14

RAD: 17

Инкрементная: 16

Быстрого прототипирования: 13

Эволюционная: 11

Благодаря процедуре выбора модели ЖЦ ПС и систем Института SQI базируется на применении четырех таблиц вопросов, соответствующих предложенной данным институтом классификации проектов и выделенным на них ответам, соответствующие моему проекту, мы выбрали модель с наибольшим количеством отмеченных ответов. Это оказалась RAD-модель.

**2. Проектирование задачи**

**2. 1 Диаграмма вариантов использования**

Диаграмма вариантов использования представлена в приложении Б

**2. 2 Диаграмма деятельности**

Диаграмма деятельности представлена в приложении А

**2. 3 Описание тестов**

Таблица 6 — Проведение тестов программы.

|  |  |  |
| --- | --- | --- |
| № Теста | Тест | Ожидаемый результат |
| 1 | 2 | 3 |
| 1 | Проверка вкладки меню “Филиалы” | При нажатии на клавишу в главном меню “1” на клавиатуре будет открыто меню ““Филиалы” |
| 2 | Проверка вкладки меню “Виды страхования” | При нажатии на клавишу в главном меню “2” на клавиатуре будет открыто меню “Виды страхования” |
| 3 | Проверка вкладки меню “Договоры” | При нажатии на клавишу в главном меню “3” на клавиатуре будет открыто меню ““Договоры” |
| 4 | Проверка вкладки меню “Выход из программы” | При нажатии на клавишу в главном меню ‘4’, программа должна завершить работу. |
| 5 | Проверка вкладки меню “Показать филиалы” | При нажатии на клавиатуре клавиши ‘1’ в меню «филиалы»,, должна быть показана информация о всех филиалах |
| 6 | Проверка вкладки меню “Добавить филиалы” | При нажатии на клавиатуре клавиши ‘2’ в меню «филиалы»,, должно быть показано меню для добавления филиала |
| 7 | Проверка вкладки меню “Удалить филиал” | При нажатии на клавиатуре клавиши ‘3’ в меню «филиалы», должно быть показано меню для удаления филиала |
| 8 | Проверка выхода в главное меню | При нажатии на клавиатуре клавиши ‘4’ в любом меню , должен быть осуществлён выход в главное меню |
| 9 | Проверка вкладки меню “Показать виды” | При нажатии на клавиатуре клавиши ‘1’ в меню «Виды страхования»,, должна быть показана информация о всех видах страхования |
| 10 | Проверка вкладки меню “Добавить вид” | При нажатии на клавиатуре клавиши ‘2’ в меню «Виды страхования»,, должна быть показано меню для добавление вида страхования |

Продолжение: Таблица 6 — Проведение тестов программы.

|  |  |  |
| --- | --- | --- |
| 11 | Проверка вкладки меню “Удалить вид” | При нажатии на клавиатуре клавиши ‘3’ в меню «Виды страхования»,, должна быть показано меню для удаления вида страхования |
| 12 | Проверка вкладки меню “Показать контракты” | При нажатии на клавиатуре клавиши ‘3’ в меню «Договоры»,, должна быть показана информация о всех договорах |
| 13 | Проверка вкладки меню “Добавить контракты” | При нажатии на клавиатуре клавиши ‘3’ в меню «Договоры»,, должно быть показано меню для добавления договора |
| 14 | Проверка вкладки меню “Удалить контракты” | При нажатии на клавиатуре клавиши ‘3’ в меню «Договоры»,, должно быть показано меню для удаления договора |

**3. Построение программы**

**3. 1 Реализация проекта**

**3. 1. 1 Выбор классов**

Для реализации программного продукта были определены следующие классы:

1. Класс «Branch» — класс, описывающий контракты;

2. Класс «InsuranceType» — класс, описывающий виды договоров;

3. Класс «Contract» — класс, описывающий контракты;

4. Класс «Program» — основной класс программы.

**3. 1. 2 Определение атрибутов классов**

В классе «Branch» определены следующие атрибуты:

1. «Id» — Код филиала;
2. «Name» — Название филиала;
3. «Adress» — Адресс;
4. «PhoneNumber» — номер телефона;

В классе «InsuranceType» определены следующие атрибуты:

1. «Id» — Код вида страхования
2. «Name» — Название;
3. «BaseCost» — Ставка;

В классе «Contract» определены следующие атрибуты:

1. «Id» — Код контракта;
2. «Date» — Дата заключения ;
3. «InsuranceSum» — Страховая сумма;
4. «Rate» — Тарифная ставка.
5. «Branch» — филиал.
6. «InsuranceType» — Вид страхования;

**3. 1. 3 Определение методов классов**

Описание разработанных процедур находится в таблице 7.

Таблица 7 — Методы классов

|  |  |  |
| --- | --- | --- |
| Имя метода | В каком классе находится | Назначение |
| 1 Branch | Branch | Конструктор класса филиалы |
| 2 Show() | Branch, InsuranceType, Contract | Метод отвечающий за вывод информации |
| 3 ToString() | Branch, InsuranceType, Contract | Метод, отвечающий за добавление информации о кдассах текстовый файл |
| 4 ToClass | Program | Вывод списка автомобилей |
| 5 Menu | Program | Основное меню программы |
| 6 BranchMenu() | Program | Метод отвечающий за меню «Филиалы» |
| 7 ShowBranches() | Program | Метод показа информации о филиалов |
| 8 AddBranch() | Program | Метод добавления филиала |
| 9 DeleteBranch() | Program | Метод удаления информации о филиала |
| 10 TypeMenu() | Program | Метод отвечающий за меню «Виды страхования» |
| 11 ShowTypes | Program | Метод показа информации о видах страхования |
| 12 AddType | Program | Метод для добавления информации о видах страхования |
| 13 DeleteType | Program | Метод для удаления вида страхования |
| 14 ContractMenu() | Program | Метод отвечающий за меню «Контракты» |
| 15 ContractsShow() | Program | Метод показа информации о филиалов |
| 16 ContractAdd | Program | Метод для добавления информации о контрактах |
| 17 ContractDelete | Program | Метод для удаления информации о контрактах |

**3. 1. 4 Диаграмма классов**

Диаграмма вариантов использования представлена в приложении Г

**4. Тестирование**

Таблица - 8 отчёт о результатах тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № Теста | Тест | Ожидаемый результат | Результат |
| 1 | 2 | 3 | 4 |
| 1 | Проверка вкладки меню “Филиалы” | При нажатии на клавишу в главном меню “1” на клавиатуре будет открыто меню ““Филиалы” | Выполнено |
| 2 | Проверка вкладки меню “Виды страхования” | При нажатии на клавишу в главном меню “2” на клавиатуре будет открыто меню “Виды страхования” | Выполнено |
| 3 | Проверка вкладки меню “Договоры” | При нажатии на клавишу в главном меню “3” на клавиатуре будет открыто меню ““Договоры” | Выполнено |
| 4 | Проверка вкладки меню “Выход из программы” | При нажатии на клавишу в главном меню ‘4’, программа должна завершить работу. | Выполнено |
| 5 | Проверка вкладки меню “Показать филиалы” | При нажатии на клавиатуре клавиши ‘1’ в меню «филиалы»,, должна быть показана информация о всех филиалах | Выполнено |
| 6 | Проверка вкладки меню “Добавить филиалы” | При нажатии на клавиатуре клавиши ‘2’ в меню «филиалы»,, должно быть показано меню для добавления филиала | Выполнено |
| 7 | Проверка вкладки меню “Удалить филиал” | При нажатии на клавиатуре клавиши ‘3’ в меню «филиалы», должно быть показано меню для удаления филиала | Выполнено |
| 8 | Проверка выхода в главное меню | При нажатии на клавиатуре клавиши ‘4’ в любом меню , должен быть осуществлён выход в главное меню | Выполнено |
| 9 | Проверка вкладки меню “Показать виды” | При нажатии на клавиатуре клавиши ‘1’ в меню «Виды страхования»,, должна быть показана информация о всех видах страхования | Выполнено |
| 10 | Проверка вкладки меню “Добавить вид” | При нажатии на клавиатуре клавиши ‘2’ в меню «Виды страхования»,, должна быть показано меню для добавление вида страхования | Выполнено |
| 11 | Проверка вкладки меню “Удалить вид” | При нажатии на клавиатуре клавиши ‘3’ в меню «Виды страхования»,, должна быть показано меню для удаления вида страхования | Выполнено |
| 12 | Проверка вкладки меню “Показать контракты” | При нажатии на клавиатуре клавиши ‘3’ в меню «Договоры»,, должна быть показана информация о всех договорах | Выполнено |

Продолжение: Таблица - 8 отчёт о результатах тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| 13 | Проверка вкладки меню “Добавить контракты” | При нажатии на клавиатуре клавиши ‘3’ в меню «Договоры»,, должно быть показано меню для добавления договора | Выполнено |
| 14 | Проверка вкладки меню “Удалить контракты” | При нажатии на клавиатуре клавиши ‘3’ в меню «Договоры»,, должно быть показано меню для удаления договора | Выполнено |

При разработке данной программы многие возникающие ошибки и недоработки были исправлены на этапе реализации проекта. После завершения этапа написания программы было проведено тщательное функциональное тестирование. Функциональное тестирование должно гарантировать работу всех элементов программы в автономном режиме.

Вывод: все тесты были успешно выполнены, все обнаруженные ошибки исправлены. Весь изначально составленный функционал успешно реализован.

**5 Руководство пользователя**

**5. 1 Назначение и условия применения программы**

Разработанная программа является консольным приложением «Страховая компания».

Создаваемая программа будет рассчитана на пользователей, обслуживающих клиентов в страховых фирмах.

Минимальные системные требования:

- процессор Pentium 300 Mhz;

- объём ОЗУ 32 мб;

- видеоподсистема: 1024\*768 точек с глубиной цвета 16 бит;

- операционная система Windows XP sp2.

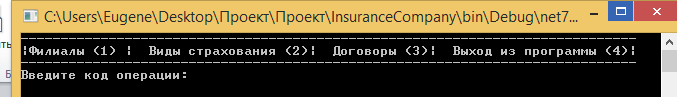
**5. 2 Выполнение программы**

Данное приложение можно запустить с помощью открытия .exe-файла.

**5. 2. 1 Инструкции по работе с программой**

При запуске приложения открывается главное меню (рисунок 1)

Рисунок 1 — Главное меню



При переходе в меню филиалы м пользователь сможет взаимодействовать с филиалами: просмотр информации о филиалов, добавление и удаление(рисунок 2)

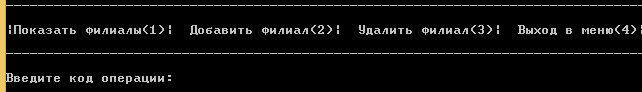


Рисунок 2 - Меню "Филиалы"

При нажатии кнопки «1» в меню филиалов будет показана информация о филиалах (рисунок 3)

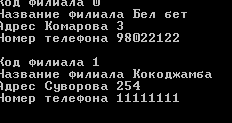


Рисунок 3 - Информация о филиалах

При нажатии кнопки «2» в меню филиалов будет показано меню для добавления (рисунок 4)

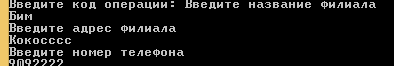


Рисунок 4 - Меню добавления филиала

При нажатии кнопки «3» в меню филиалов будет показана информация о филиалах (рисунок 5)

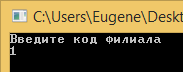


Рисунок 5 - Меню удаления филиала

При нажатии кнопки «2» в главном меню будет показано меню видов страхования (рисунок 6)

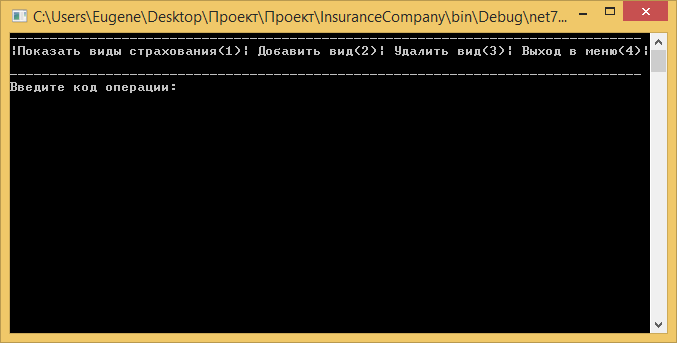


Рисунок 6 - Меню Видов страхования

При нажатии кнопки «2» в меню видов страхования будет показана информация о всех видах страхования (рисунок 7)

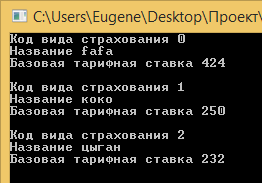


Рисунок 7 - Информация о видах страхования

При нажатии кнопки «3» в меню видов страхования будет показана информация о всех видах страхования (рисунок 8)

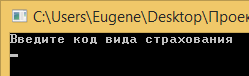


Рисунок 8 - Меню удаления видов страхования

При нажатии кнопки «3» в главном будет показано меню для взаимодействия с договорами (рисунок 9)

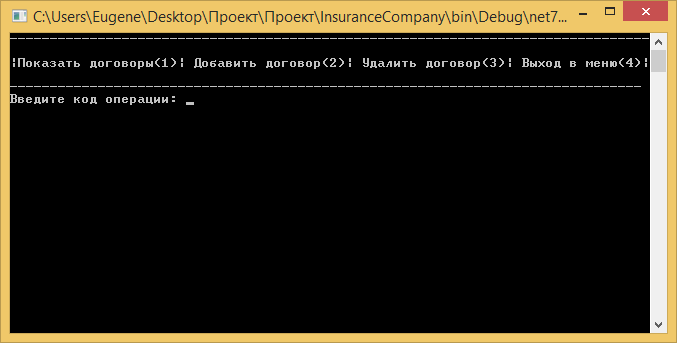


Рисунок 9 - Меню договоры

При нажатии кнопки «1» в меню договоров будет показана информация о всех договорах (рисунок 10)

При нажатии кнопки «2» в меню договоров будет показано меню для добавления договоров (рисунок 11)

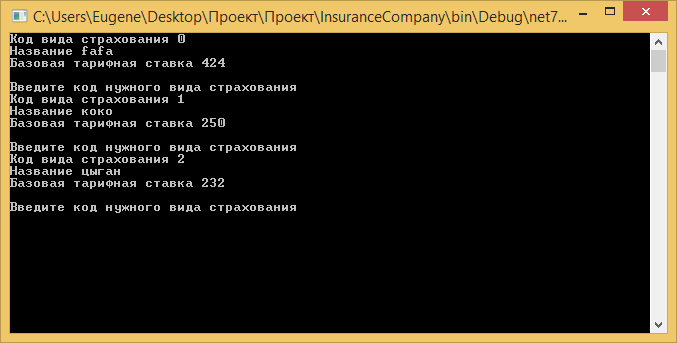


Рисунок 11 - Меню для добавления договоров

При нажатии кнопки «3» в меню договоров будет показано меню для удаления договоров (рисунок 12)

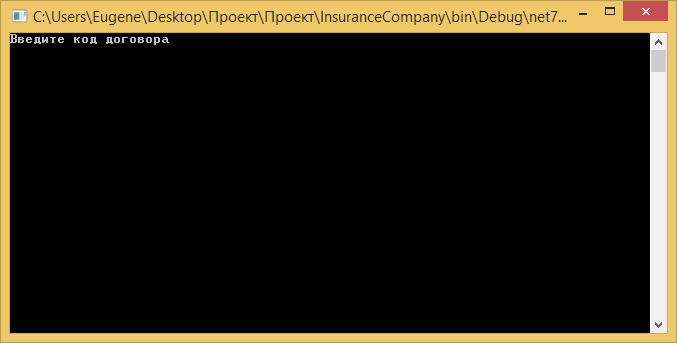


Рисунок 12 - Меню для удаления договоров

**5. 3 Завершение работы с программой**

Для завершения работы с программой нужно закрыть вкладку консоли или нажать на кнопку «4» в главном меню приложения.

**Заключение**

Целью данного проекта была разработка консольного приложения «Страховая компания». В поставленной задаче был реализован простой и понятный пользовательский интерфейс, в ходе тестирования все исключительные ситуации были обработаны. Проект работает без сбоев и ошибок. В процессе разработки программного продукта я улучшил свои навыки использования C#, закрепил умение создания собственных функций, научился обрабатывать все исключительные ситуации. Все поставленные требования были выполнены и проект реализован успешно

**Список использованных источников**

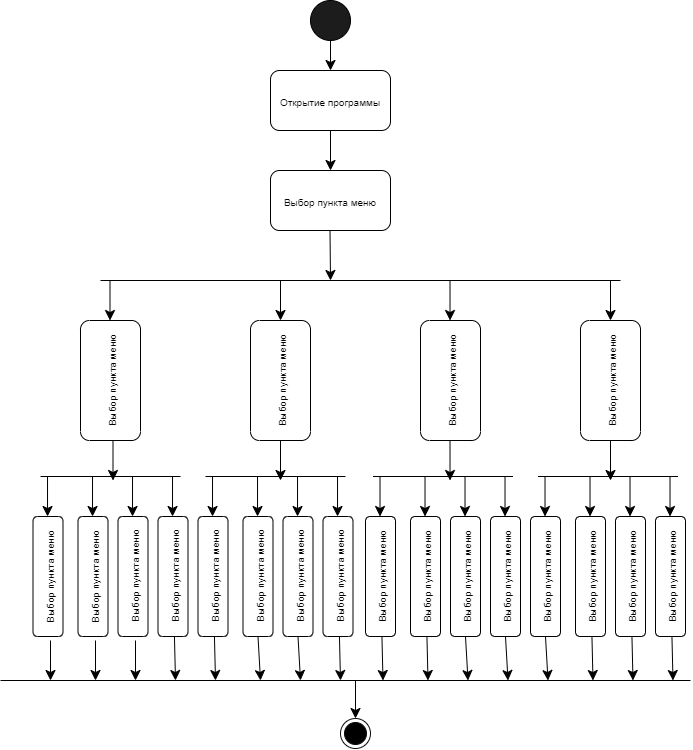
1. Руководство по C# [Электронный ресурс]. – Режим доступа: https://metanit.com/

2. Документация по С# [Электронный ресурс]. – Режим доступа:

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

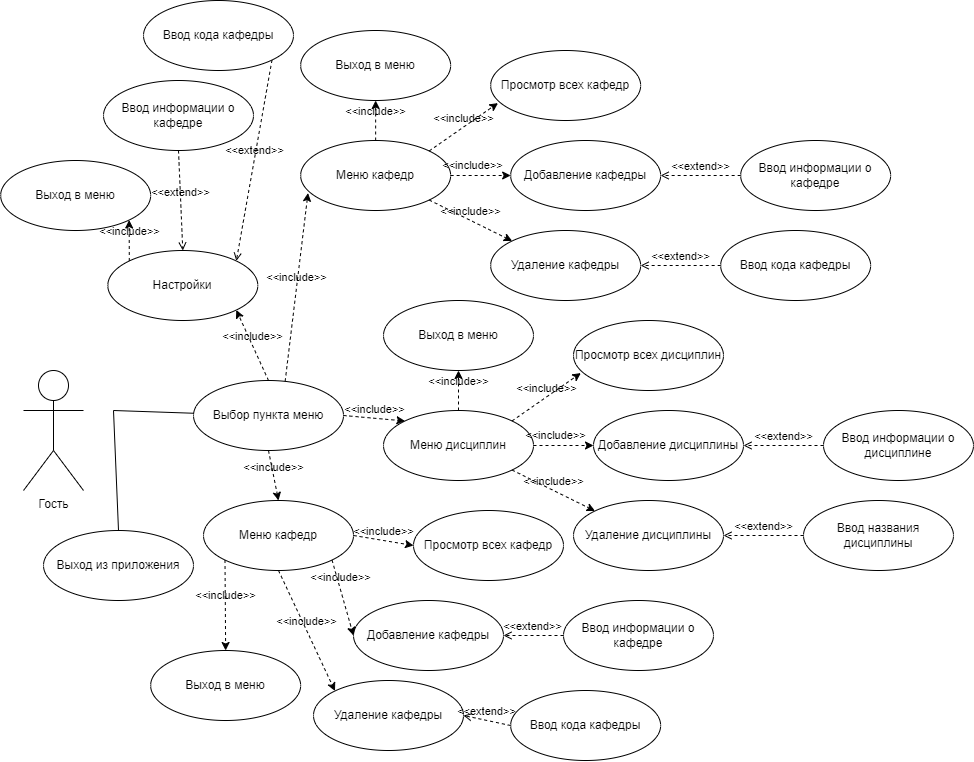
**Приложение А**

**Диаграмма деятельности**

****

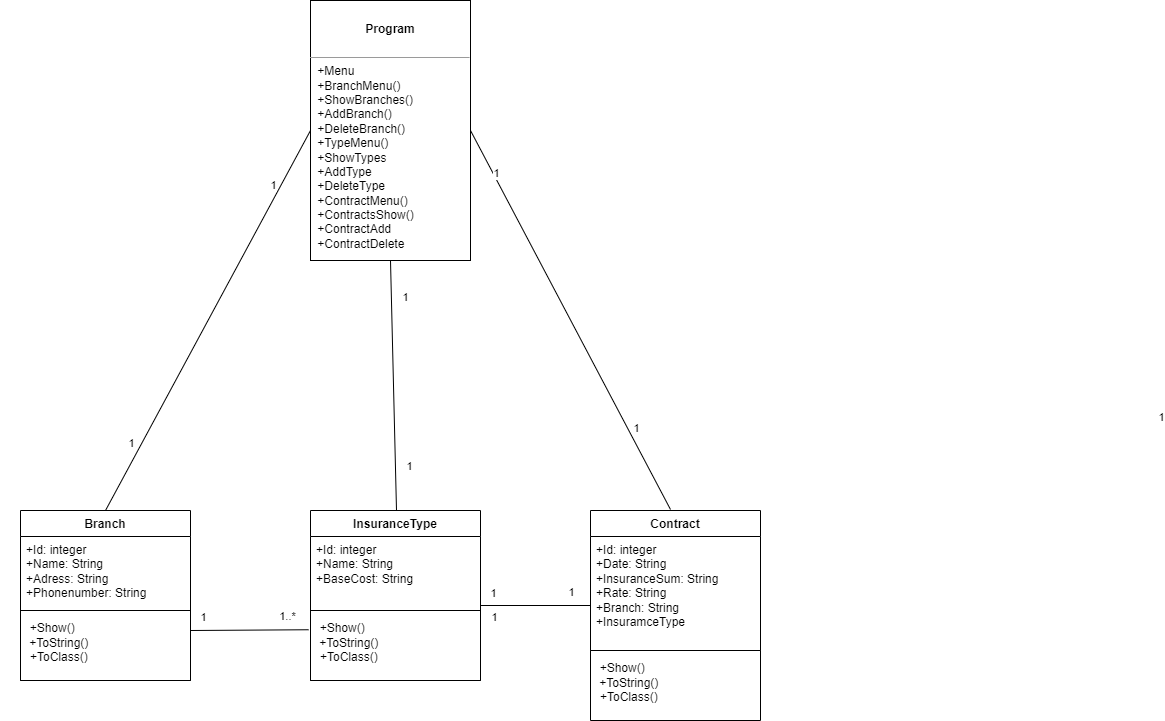
**Приложение Б**

**Диаграмма вариантов использования**

****

**Приложение В**

**Диаграмма классов**

****

**Приложение Г**

**Листинг программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Classes

{

public class Branch

{

public int Id { get; set; }

public string Name { get; set; }

public string Adress { get; set; }

public string PhoneNumber { get; set; }

public Branch(int id, string name, string adress, string phoneNumber)

{

Id = id;

Name = name;

Adress = adress;

PhoneNumber = phoneNumber;

}

public void Show()

{

Console.WriteLine($"Код филиала {Id}\n" +

$"Название филиала {Name}\n" +

$"Адрес {Adress}\n" +

$"Номер телефона {PhoneNumber} \n");

}

public string ToString()

{

return $"{Id},{Name},{Adress},{PhoneNumber}";

}

public static Branch ToClass(string line)

{

string[] mas = line.Split(',');

Branch branch = new Branch(int.Parse(mas[0]), mas[1], mas[2], mas[3]);

return branch;

}

}

}

namespace Classes

{

public class InsuranceType

{

public int Id { get; set; }

public string Name { get; set; }

public decimal BaseCost { get; set; }

public InsuranceType(int id, string name, decimal baseCost)

{

Id = id;

Name = name;

BaseCost = baseCost;

}

public void Show()

{

Console.WriteLine($"Код вида страхования {Id}\n" +

$"Название {Name}\n" +

$"Базовая тарифная ставка {BaseCost}\n");

}

public string ToString()

{

return $"{Id},{Name},{BaseCost}";

}

public static InsuranceType ToClass(string line)

{

string[] mas = line.Split(',');

InsuranceType type = new InsuranceType(int.Parse(mas[0]), mas[1], decimal.Parse(mas[2]));

return type;

}

}

}using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Classes

{

public class Contract

{

public int Id { get; set; }

public DateTime Date { get; set; }

public decimal InsuranceSum { get; set; }

public decimal Rate { get; set; }

public int Branch { get; set; }

public int InsuranceType { get; set; }

public Contract(int id, DateTime date, decimal insuranceSum, decimal rate, int branch, int insuranceType)

{

Id = id;

Date = date;

InsuranceSum = insuranceSum;

Rate = rate;

Branch = branch;

InsuranceType = insuranceType;

}

public void Show()

{

Console.WriteLine($"Номер договора {Id}\n" +

$"Дата заключения {Date}\n" +

$"Страховая сумма {InsuranceSum}\n" +

$"Тарифная ставка {Rate} \n" +

$"Код филиала {Branch} \n" +

$"Код вида страхования {InsuranceType} \n");

}

public string ToString()

{

return $"{Id},{Date},{InsuranceSum},{Rate},{Branch},{InsuranceType}";

}

public static Contract ToClass(string line)

{

string[] mas = line.Split(",");

Contract contract = new Contract(int.Parse(mas[0]), DateTime.Parse(mas[1]), decimal.Parse(mas[2]), decimal.Parse(mas[3]), int.Parse(mas[4]), int.Parse(mas[5]));

return contract;

}

}

}

using Classes;

using System;

using System.IO.Pipes;

using System.Net.Http.Headers;

Menu();

static void Menu()

{

Console.Clear();

Console.WriteLine("-----------------------------------------------------------------------------\n" +

"|Филиалы (1) | Виды страхования (2)| Договоры (3)| Выход из программы (4)|\n" +

"-----------------------------------------------------------------------------");

Console.Write("Введите код операции: ");

char Code = Console.ReadKey(true).KeyChar;

switch (Code)

{

case '1':

BranchMenu();

break;

case '2':

TypeMenu();

break;

case '3':

ContractMenu();

break;

case '4':

Environment.Exit(0);

break;

}

}

static void BranchMenu()

{

ICollection<Branch> \_branches = new List<Branch>();

int branch\_id = -1;

using (StreamReader reader = new StreamReader("branches.txt"))

{

while (!reader.EndOfStream)

{

\_branches.Add(Branch.ToClass(reader.ReadLine()));

}

}

if (\_branches.Count > 0) { branch\_id = \_branches.Last().Id; }

Console.Clear();

Console.WriteLine("--------------------------------------------------------------------------------\n" +

"|Показать филиалы(1)| Добавить филиал(2)| Удалить филиал(3)| Выход в меню(4)|\n" +

"--------------------------------------------------------------------------------");

Console.Write("Введите код операции: ");

char Code = Console.ReadKey(true).KeyChar;

switch (Code)

{

case '1':

ShowBranches(\_branches);

break;

case '2':

AddBranch(\_branches, branch\_id);

break;

case '3':

DeleteBranch(\_branches);

break;

case '4':

Menu();

break;

}

}

static void ShowBranches(ICollection<Branch> \_branches)

{

Console.Clear();

if (\_branches.Count == 0)

{

Console.WriteLine("Филиалов нет");

}

else

{

foreach (Branch branch in \_branches)

{

branch.Show();

}

}

Console.ReadKey();

BranchMenu();

}

static void AddBranch(ICollection<Branch> \_branches, int branch\_id)

{

Console.WriteLine("Введите название филиала");

string Name = Console.ReadLine();

Console.WriteLine("Введите адрес филиала");

string Adress = Console.ReadLine();

Console.WriteLine("Введите номер телефона");

string PhoneNumber = Console.ReadLine();

branch\_id++;

Branch branch = new Branch(branch\_id, Name, Adress, PhoneNumber);

\_branches.Add(branch);

using (StreamWriter writer = new StreamWriter("branches.txt", false))

{

foreach (Branch \_branch in \_branches)

{

writer.WriteLine(\_branch.ToString());

}

}

BranchMenu();

}

static void DeleteBranch(ICollection<Branch> \_branches)

{

Console.Clear();

Console.WriteLine("Введите код филиала");

int id = int.Parse(Console.ReadLine());

var temp = \_branches.Where(d => d.Id == id).First();

if (temp is not null)

{

\_branches.Remove(temp);

using (StreamWriter writer = new StreamWriter("branches.txt", false))

{

foreach (Branch \_branch in \_branches)

{

writer.WriteLine(\_branch.ToString());

}

}

}

else

{

Console.WriteLine("Филиала с таким кодом не существует");

Console.ReadKey();

}

BranchMenu();

}

static void TypeMenu()

{

ICollection<InsuranceType> \_types = new List<InsuranceType>();

int types\_id = -1;

using (StreamReader reader = new StreamReader("types.txt"))

{

while (!reader.EndOfStream)

{

\_types.Add(InsuranceType.ToClass(reader.ReadLine()));

}

}

if (\_types.Count > 0)

{

types\_id = \_types.Last().Id;

}

Console.Clear();

Console.WriteLine("-------------------------------------------------------------------------------\n" +

"|Показать виды страхования(1)| Добавить вид(2)| Удалить вид(3)| Выход в меню(4)|\n" +

"-------------------------------------------------------------------------------");

Console.Write("Введите код операции: ");

char Code = Console.ReadKey(true).KeyChar;

switch (Code)

{

case '1':

ShowTypes(\_types);

break;

case '2':

AddType(\_types, types\_id);

break;

case '3':

DeleteType(\_types);

break;

case '4':

Menu();

break;

}

}

static void ShowTypes(ICollection<InsuranceType> types)

{

Console.Clear();

if (types.Count == 0)

{

Console.WriteLine("Видов страхования нет");

}

else

{

foreach (InsuranceType type in types)

{

type.Show();

}

}

Console.ReadKey();

TypeMenu();

}

static void AddType(ICollection<InsuranceType> types, int type\_id)

{

Console.WriteLine("Введите название");

string Name = Console.ReadLine();

Console.WriteLine("Введите базовую ставку");

decimal BaseCost = decimal.Parse(Console.ReadLine());

type\_id++;

InsuranceType type = new InsuranceType(type\_id, Name, BaseCost);

types.Add(type);

using (StreamWriter writer = new StreamWriter("types.txt", false))

{

foreach (InsuranceType \_type in types)

{

writer.WriteLine(\_type.ToString());

}

}

TypeMenu();

}

static void DeleteType(ICollection<InsuranceType> types)

{

Console.Clear();

Console.WriteLine("Введите код вида страхования");

int id = int.Parse(Console.ReadLine());

var temp = types.Where(d => d.Id == id).First();

if (temp is not null)

{

types.Remove(temp);

using (StreamWriter writer = new StreamWriter("types.txt", false))

{

foreach (InsuranceType \_type in types)

{

writer.WriteLine(\_type.ToString());

}

}

}

else

{

Console.WriteLine("Вида страхования с таким кодом не существует");

Console.ReadKey();

}

TypeMenu();

}

static void ContractMenu()

{

ICollection<Contract> \_contracts = new List<Contract>();

int contract\_id = -1;

using (StreamReader reader = new StreamReader("contracts.txt"))

{

while (!reader.EndOfStream)

{

\_contracts.Add(Contract.ToClass(reader.ReadLine()));

}

}

if (\_contracts.Count > 0) { contract\_id = \_contracts.Last().Id; }

Console.Clear();

Console.WriteLine("--------------------------------------------------------------------------------\n" +

"|Показать договоры(1)| Добавить договор(2)| Удалить договор(3)| Выход в меню(4)|\n" +

"-------------------------------------------------------------------------------");

Console.Write("Введите код операции: ");

char Code = Console.ReadKey(true).KeyChar;

switch (Code)

{

case '1':

ContractsShow(\_contracts);

break;

case '2':

ContractAdd(\_contracts,contract\_id);

break;

case '3':

ContractDelete(\_contracts);

break;

case '4':

Menu();

break;

}

}

static void ContractsShow(ICollection<Contract> contracts)

{

Console.Clear();

if (contracts.Count== 0)

{

Console.WriteLine("Договоров нет");

}

else

{

foreach (Contract contract in contracts)

{

contract.Show();

}

}

ContractMenu();

}

static void ContractAdd(ICollection<Contract> contracts, int contract\_id)

{

ICollection<InsuranceType> \_types = new List<InsuranceType>();

using (StreamReader reader = new StreamReader("types.txt"))

{

while (!reader.EndOfStream)

{

\_types.Add(InsuranceType.ToClass(reader.ReadLine()));

}

}

ICollection<Branch> \_branches = new List<Branch>();

using (StreamReader reader = new StreamReader("branches.txt"))

{

while (!reader.EndOfStream)

{

\_branches.Add(Branch.ToClass(reader.ReadLine()));

}

}

Console.Clear();

if (\_types.Count == 0)

{

Console.WriteLine("Видов страхования нет");

Console.ReadKey();

ContractMenu();

}

else

{

foreach (InsuranceType type in \_types)

{

type.Show();

Console.WriteLine("Введите код нужного вида страхования");

}

int t\_code = int.Parse(Console.ReadLine());

int t\_temp = \_types.Where(d => d.Id == t\_code).First().Id;

if (t\_temp != null)

{

Console.Clear();

if (\_branches.Count == 0)

{

Console.WriteLine("Филиалов нет");

Console.ReadKey();

ContractMenu();

} else

{

foreach (Branch branch in \_branches)

{

branch.Show();

}

int b\_code = int.Parse(Console.ReadLine());

int b\_temp = \_branches.Where(d => d.Id == b\_code).First().Id;

if (b\_temp != null)

{

Console.Clear();

Console.WriteLine("Введите дату заключения");

DateTime Date = DateTime.Parse(Console.ReadLine());

Console.WriteLine("Введите страховую сумму");

decimal InsuranceSum = decimal.Parse(Console.ReadLine());

Console.WriteLine("Введите тарифную ставку");

decimal Rate = decimal.Parse(Console.ReadLine());

contract\_id++;

Contract contract = new Contract(contract\_id, Date, InsuranceSum, Rate, b\_code, t\_code);

contracts.Add(contract);

using (StreamWriter writer = new StreamWriter("contracts.txt", false))

{

foreach (Contract \_contract in contracts)

{

writer.WriteLine(\_contract.ToString());

}

}

}

}

}

}

ContractMenu();

}

static void ContractDelete(ICollection<Contract> contracts)

{

Console.Clear();

Console.WriteLine("Введите код договора");

int id = int.Parse(Console.ReadLine());

var temp = contracts.Where(d => d.Id == id).First();

if (temp != null)

{

contracts.Remove(temp);

using (StreamWriter writer = new StreamWriter("contracts.txt", false))

{

foreach (Contract contract in contracts)

{

writer.WriteLine(contract.ToString());

}

}

}

ContractMenu();

}