Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет математики и информационных технологий

Кафедра программного обеспечения вычислительной техники и

автоматизированных систем

**КУРСОВАЯ РАБОТА**

по дисциплине «Объектно-ориентированное программирование»

**Разработка объектно-ориентированных программ с использованием**

**современных технологий программирования**

Пояснительная записка

ОГУ 09.03.04.3021.110 ПЗ

|  |
| --- |
| Руководитель  канд. техн. наук, доцент  Н.А. Тишина  « » 2021 г.  Студент группы 20ПИнж(б)РПиС  E.А. Нечаев  « » 2021 г. |

Оренбург 2021

Утверждаю

заведующий кафедрой программного

обеспечения вычислительной техники

и автоматизированных систем

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Н.А. Соловьев

подпись

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы**

студенту Нечаеву Евгению Андреевичу

(фамилия имя отчество)

по направлению подготовки 09.03.04 Программная инженерия

код, наименование

по дисциплине «Объектно-ориентированное программирование»

наименование дисциплины, модуля

1 Тема проекта (работы) Разработка объектно-ориентированных программ с использованием современных технологий программирования

2 Срок сдачи студентом проекта (работы) «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

3 Цель и задачи проекта (работы) Изучить технологию объектно-ориентированного подхода к разработке программ в Visual Studio .Net, освоить использование элементов графического интерфейса Windows Forms для управления работой приложения

4 Исходные данные к проекту (работе) Предметная область: Демонстрационные модели физических явлений. Текст задания: Рассматривается замкнутая электрическая цепь, по которой течет постоянный ток. В цепь включены элементы нескольких видов: электрическая лампочка, резистор, источник

постоянного тока (ЭДС), выключатель (замыкающий ключ). В цепи так же находиться три узла. Основная функция программной системы – расчет для заданной электрической схемы основных ее характеристик, т.е. величины тока в каждой ее точке и направление тока в каждой ветке электрической схемы

5 Перечень вопросов, подлежащих разработке разработка и реализация диаграммы классов. Создать GUI-приложение с элементами ввода и отображения полей класса. Для этого использовать различные элементы управления: текстовые поля, списки, независимые кнопки, а также панели и менеджеры компоновки

Дата выдачи и получения задания

Руководитель «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г. \_\_\_\_\_\_\_\_\_\_\_\_ Н. А. Тишина

Студент «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г. \_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Нечаев

**Аннотация**

Курсовая работа посвящена вопросу разработки объектно-ориентированной программы для компьютеров, которая производит расчет сложных электрических цепей, на основе метода узловых потенциалов.

Подробно даётся характеристика принципа работы программы, а также описание метода узловых потенциалов.

Используя схематичное и словесное описание, автор излагает принципы взаимосвязи классов приложения.

Большое место в работе уделено описанию диаграммы классов.

В работе приводятся описание интерфейса программного средства.

Работа представляет интерес с точки зрения использования данной программы студентами, преподавателями, а также лаборантами.

Пояснительная записка к курсовой работе содержит 32 листа, 14 рисунков, 6 таблиц, 1 приложение.

**Содержание**

Введение 5

1 Исследование предметной области 6

1.1 Выбор метода расчета характеристик электрической цепи и описание метода 6

1.2 Диаграмма классов программного средства 8

1.3 Текстовые спецификации интерфейса классов системы 11

1.4 Диаграмма объектов программного средства 13

2 Разработка программного средства 15

3 Руководство оператора 15

3.1 Условия выполнения программы 16

3.2 Выполнение программы 16

Заключение 21

Список использованных источников 22

Приложение А (обязательное) Код программы 23

# Введение

# Расчет сложных электрических цепей без использования средств автоматизаций трудоемкий процесс. Необходимо производить большое количество вычислений, и зарисовок.

# Разработанное приложение призвано облегчить вычисления характеристик электрических схем, и упростить работу с их построением.

Цель работы: изучить технологию объектно-ориентированного подхода к разработке программ в Visual Studio .Net, освоить использование элементов графического интерфейса Windows Forms для управления работой приложения.

Для достижения цели работы необходимо решить следующие инженерные задачи:

* разработать диаграмму классов программного средства;
* разработать программу, осуществляющую построение и расчет основных характеристик электрических цепей
* ииспользовать для создания ПС объектно-ориентированный язык программирования С#, а также поддерживающие его инструментальные средства.

**1 Исследование предметной области**

**1.1 Выбор метода расчета характеристик электрической цепи и описание метода**

Для решения поставленной задачи был выбран метод узловых потенциалов, так как благодаря этому методу можно быстро рассчитать основные характеристики электрической цепи. Данный метод подходит для расчета как небольших электрических схем, с двумя или тремя ветками, так и для расчета цепей, где количество веток очень велико. Количество узлов в ветке так же может варьироваться, начиная с двух узлов в цепи, заканчивая любым их количеством. В программе реализован расчет схем, в которых количество узлов не превышает трех. Далее приведено описание метода узловых потенциалов, в основу которого легли законы Кирхгофа [1].

Данный метод начинается с выбора направления тока в каждой ветки электрической цепи. Это направление выбирается произвольно, так как в результате расчетов мы получим истинное направление тока в ветвях. Далее необходимо принять потенциал одного из узлов за ноль (данный узел будет называться базовым узлом), а потенциалы остальных узлов будем определять относительно базового. Сила тока в каждой ветке рассчитывается по формуле:

(1)

где – потенциал узла из которого выходит ток в ветке, В;

– потенциал узла в который входит ток, В;

– сумма напряжений ЭДС источников в ветке, В. Если ЭДС источники смотрят в разные стороны, то один из них нужно брать с минусом;

– общее сопротивление ветки, ОМ.

можно заменить на общую проводимость ветки , которая вычисляется по формуле:

(2)

Тогда формула примет следующий вид:

(3)

В работе будет использоваться второй вариант формулы.

Затем необходимо составить систему уравнений следующего вида:

(4)

где – общая проводимость всех веток, которые подключены к узлу 1 с потенциалом ;

– общая проводимость всех веток, которые подключены к узлам 1 и узлу 2;

– общая проводимость всех веток, подключенных к узлам 2 и 3;

– сумма выражений всех веток подключенных к узлу 1, где – сумма напряжений ЭДС источников ветки, причем если ЭДС источник направлен к узлу, то берем его со знаком плюс, иначе со знаком минус;

– общая проводимость ветки.

Остальные и вычисляются по аналогии, но для узлов 2 и 3 c потенциалами и соответственно.

Так как было оговорено, что потенциал 3-ого узла мы примем за ноль, то система уравнений примет следующий вид:

(5)

Затем необходимо создать матрицу, которая следует из системы уравнений, и найти ее определитель. Она имеет вид:

(6)

Будем считать, что определитель данной матрицы равен . Чтобы найти потенциалы узлов и нужно вычислить определитель следующих матриц:

(7)

(8)

Потенциал узла 1 будет равен:

(9)

Потенциал узла 2:

(10)

После нахождения потенциалов, можно вычислить силу тока в каждой ветке по формуле (2), причем если сила тока по модулю получилась отрицательна, то мы меняем направление тока в цепи [2].

**1.2** **Диаграмма классов программного средства**

Изучив поставленную задачу можно составить следующую структуру классов.

Абстрактный класс Elements является родителем всех классов, которые представляют элементы электрической цепи. От класса Elements наследуется четыре класса: Resister, Key, Lamp, CCS. Класс Resistor представляет из себя резистор, хранящий в себе сопротивление резистора и его проводимость. Класс Key представляет ключ в электрической цепи, и хранит в себе одну булевскую переменную, которая показывает, замкнут ли ключ (опущен) или разомкнут (поднят). На основе этого элемента будет учитываться, включена ли ветка в цепь или нет. Класс Lamp так же, как и класс Key хранит в себе всего одну булевскую переменную, указывающую есть ток в ветке или нет. Класс CCS представляет из себя ЭДС источник, который хранит в себе два узла: узел на который указывает ЭДС источник и узел от которого он повернут и реализует интерфейс IDirection, который задает направление тока. Так же в нем храниться информация о его выдаваемом напряжении.

Класс Node представляет из себя узел в электрической цепи. Он хранит в себе общую проводимость узла и его потенциал.

Класс Branche представляет из себя отдельную ветку в электрической цепи. Данный класс, так же, как и класс CCS, реализует интерфейс IDirection. Класс хранит в себе информацию о двух узлах, к которым подключена ветка, общую проводимость ветки, силе тока в данной ветке и ее направлении, сумме напряжений всех ЭДС источников в ветке, а также объект класса-коллекции CollectionElemnts.

Так же разработаны два класса коллекции CollectionElements и CollectionBramches, которые реализуют интерфейс IEnumerable. Данный интерфейс позволяет проходить по каждому элементу внутри класса-коллекции при помощи оператора foreach. Класс CollectionElements содержит в себе список элементов электрической цепи. Класс CollectionBranches содержит в себе список веток.

Класс ElectricalСircuit представляет электрическую цепь. Класс хранит в себе три узла и объект класса-коллекции CollectionBranches.

Описание интерфейсов предоставлено в таблице 1. Описание и связи вышеперечисленных классов представлены в таблице 2. Диаграмма классов представлена на рисунке 1.

Таблица 1 - Интерфейсы программного средства

|  |  |  |
| --- | --- | --- |
| Имя | Функциональное назначение | Поля и методы |
| IDirection | Позволяет задать направление тока в ветке и выбрать узлы для направления ЭДС | Node IncumingNode – входящий узел. Узел, в который входит ток.  Node OutcumingNode – выходящий узел. Узел, из которого выходит ток. |
| IEnumerable | Позволяет проходить по каждому элементу внутри класса-коллекции внутри оператора foreach | func GetEnumerator – метод, позволяющий получить нумератор элемента. |

Таблица 2 – Классы программного средства

|  |  |  |  |
| --- | --- | --- | --- |
| Имя класса | Модификаторы | Функциональное назначение | Связь с другими классами и интерфейсами |
| 1 | 2 | 3 | 4 |
| Elements | abstract | Представляет общие характеристики для элементов | - |
| Resister | - | Представляет резистор внутри электрической цепи | Наследуется от Elements и содержится в классе-коллекции CollectionElements |
| Key | - | Представляет ключ внутри электрической цепи | Наследуется от Elements и содержится в классе-коллекции CollectionElements |
| CCS | - | Представляет ЭДС источник внутри электрической цепи | Наследуется от Elements и содержится в классе-коллекции CollectionElements |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Lamp | - | Представляет лампу внутри электрической цепи | Наследуется от Elements и содержится в классе-коллекции CollectionElements |
| Node | - | Представляет узел в электрической цепи | Содержится в классе ElectricalCircuit |
| ElectricalCircuit | - | Представляет электрическую цепь | Содержит в себе объект класса-коллекции CollectionBranches |
| CollectionBranches | - | Представляет класс-коллекцию, который хранит в себе список всех веток Branches | Содержиться в классе ElectricalCircuit |
| Branch | - | Представляет ветку электрической цепи | Содержится внутри класса-коллекции CollectionBranches и содержит в себе коллекцию CollectionElements |
| CollectionElemets | - | Представляет класс-коллекцию, который хранит в себе список элементов Elements | Содержится внутри класса Branch |

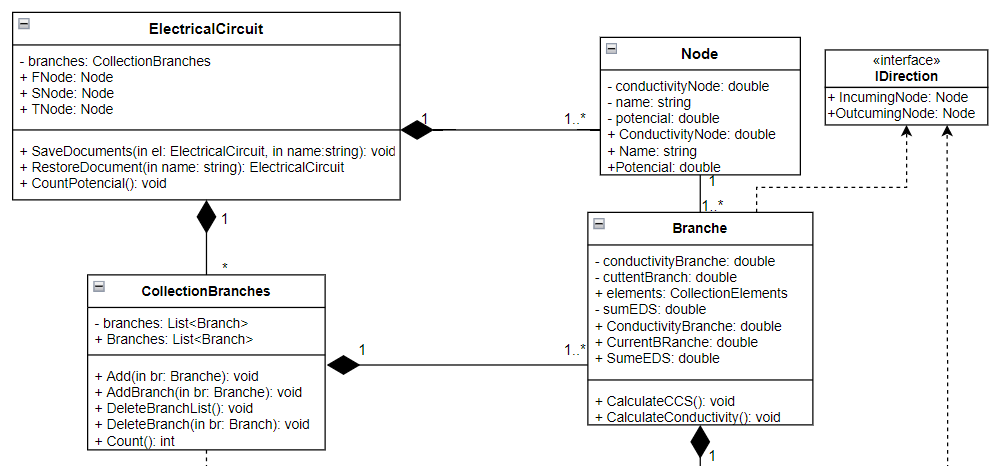


Рисунок 1 – Диаграмма классов программного средства

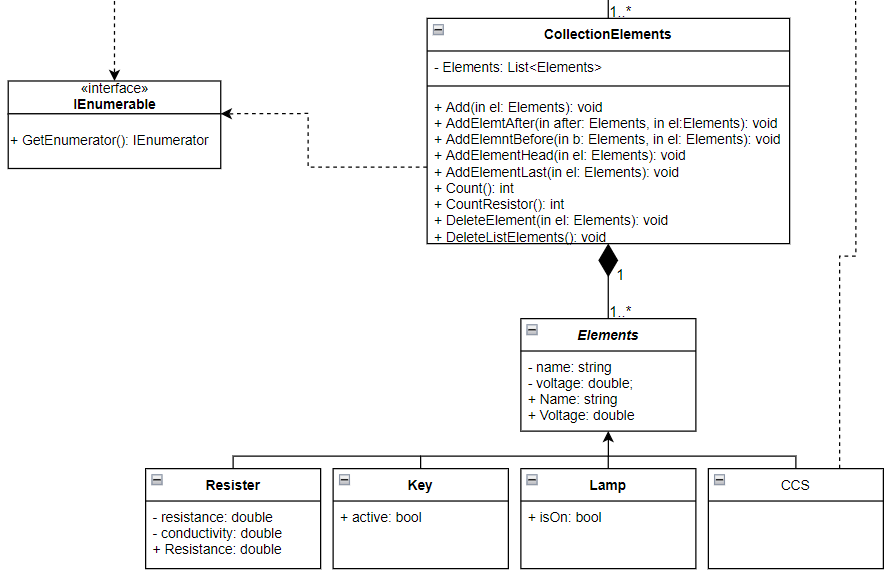


Рисунок 1, лист 1

**1.3 Текстовые спецификации интерфейса классов системы**

Спецификация методов класса ElectricalCircuit предоставлена в таблице 3.

Таблица 3 – Спецификация методов класса ElectricalCircuit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Модификаторы | Функциональное назначение | Входные параметры | Возвращаемое значение |
| RestoreDocuments | - | Загружает цепь из выбранного файла | nameFile: String – названия файла для загрузки | ElectricalCircuit – возвращает электрическую цепь, загруженную из файла |
| SaveDocuments | - | Сохраняет цепь в файл | electricalCircuit: ElectricalCircuit, nameFile: string – получает электрическую цепь и название файла, в которой нужно ее сохранить. | void |
| CountPotencial | - | Вычисляет потенциал узлов | - | - |

Спецификация методов класса CollectionBranches предоставлена в таблице 4.

Таблица 4 - Спецификация методов класса CollectionBRanches

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Модификаторы | Функциональное назначение | Входные параметры | Возвращаемое значение |
| AddBranche | - | Добавляет новую ветку в коллекцию | br: Branche – ветка | void |
| DeleteListBranche | - | Удаляет всю коллекцию веток | - | void |
| DeleteBranche | - | Удаляет заданную ветку из коллекции | br: Branche – заданная ветка | void |
| Count | - | Считает кол-во веток в коллекции | - | int |

Спецификация методов класса Branche предоставлена в таблице 5.

Таблица 5 – Спецификация методов класса Branche

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Модификаторы | Функциональное назначение | Входные параметры | Возвращаемое значение |
| CalculateCCS | - | Вычисляет общее напряжение выдаваемое ЭДС источниками в ветке | br: Branche – ветка для расчета | void |
| CalculateConductivity | - | Вычисляет общую проводимость ветки | br: Branche – ветка для расчета | void |

Спецификация методов класса CollectionElements предоставлена в таблице 6.

Таблица 6 – Спецификация методов класса CollectionElements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя метода | Модификаторы | Функциональное назначение | Входные параметры | Возвращаемое значение |
| 1 | 2 | 3 | 4 | 5 |
| AddElementAfter | - | Добавить элемент до заданного в коллекции | after: Elements  el: Elements – заданный и новый элемент | void |
| DeleteListElements | - | Удаляет всю коллекцию элементов | - | void |
| DeleteElement | - | Удаляет заданный элемент из коллекции | el: Elements – заданный элемент для удаления | void |

Продолжение таблицы 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| CountResistor | - | Подсчитать количество резисторов в коллекции | - | int |
| AddElementBefore | - | Добавляет элемент после заданного в коллекции | before: Elements  el: Elements – заданный и новый элемент | void |
| AddElementHead | - | Добавить элемент в начало коллекции | el: Elements – элемент для добавления | void |
| AddElementLast | - | Добавить элемент в конец коллекции | el: Elements – элемент для добавления | void |
| Count | - | Подсчитать общее количество элементов в коллекции | - | int |

**1.4 Диаграмма объектов программного средства**

Для простаты диаграммы (см. рисунок 2), допустим, что программе на расчет дана следующая электрическая цепь, которая не включена. (см. рисунок 3).

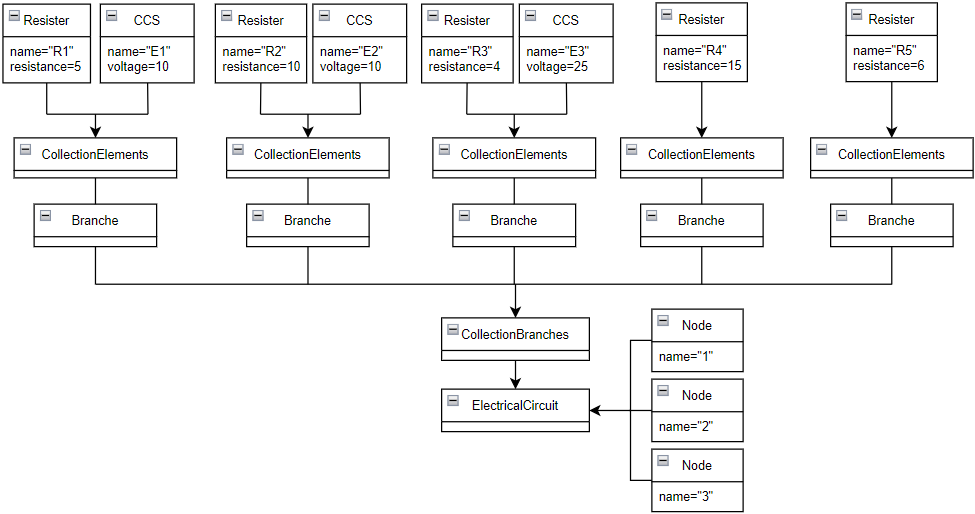
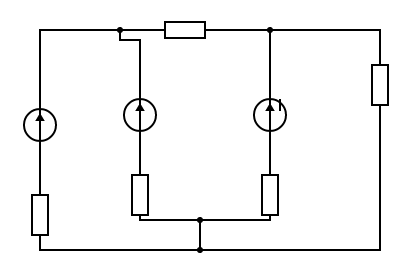


Рисунок 2 – Диаграмма объектов программного средства



3

2

1

R5

E3

E2

E1

R4

R3

R2

R1

Рисунок 3 – Пример электрической цепи

**2 Разработка программного средства**

Для выполнения поставленной задачи использовался язык программирования C#. C# обладает многими особенностями современных языков, которые делают его привлекательным для разработчиков. Эти особенности:

- Компактность, лаконичность и удобочитаемость;

- Объектно-ориентированное и функциональное программирование;

- Компилятор обеспечивает безопасность [3].

Используемая среда разработки – Visual Studio

В процессе разработки программного средства использовались следующие пространство имен:

* System. В данном пространстве имен описаны все основные типы, используемые в программировании на C#, такие как: int, string, double, Object и другие.
* System.Form. Позволяет работать с Windows Forms и его компонентами. Благодаря этому пространству имен возможно создание прикладных приложений. Весь визуальный интерфейс программы реализован с помощью данного пространства имен.
* System.IO. Позволяет организовать работу с файлами. Сами файлы использовались для сохранения и загрузки созданных электрических цепей.
* System.Xml.Serialization. Благодаря данному пространству имен возможно сохранения состояния объекта, записи этого состояния в файл и его хранение в файле с расширением xml, другими словами – сериализовать объект. Данное пространство имен так же позволяет де-сериализовать объект и загрузить его в программу из файла.
* System.Collections.Generic. Данное пространство имен позволяет использовать обобщенные коллекции, такие как: List<T>, Stack<T> и многие другие.

**3 Руководство оператора**

**3.1 Условия выполнения программы**

Для установки и правильной работы программы необходимо:

- Версия Windows не ниже 7;

- Не менее 1 Гб свободного места;

- Оперативной памяти не менее 2 Гб.

**3.2 Выполнение программы**

Запустите приложение. На экране появиться главное меню (см. рисунок 4). На данном этапе, вам доступно только две кнопок: Добавить ветку; Загрузить цепь.

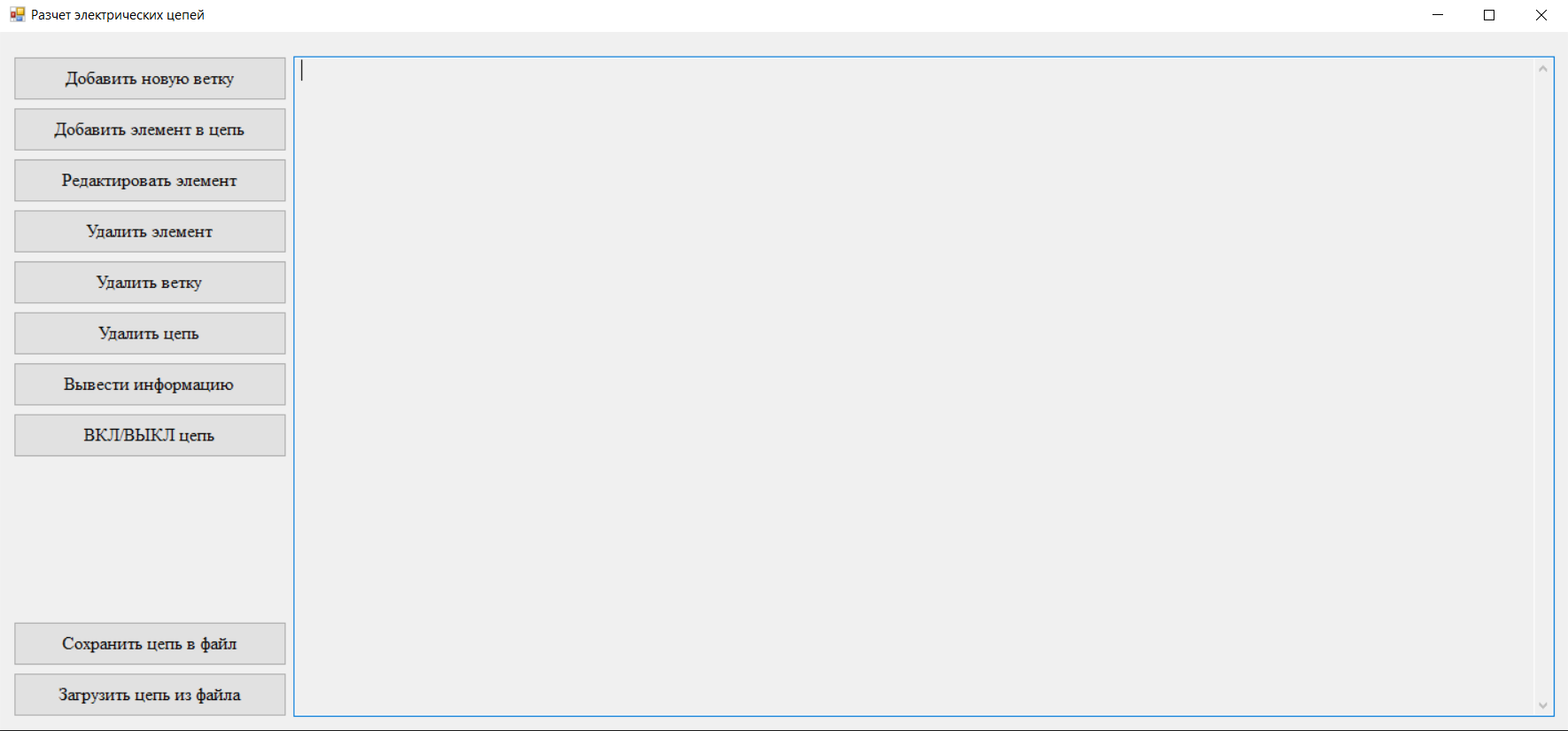


Рисунок 4 – Главное меню приложения

Далее необходимо нажать на кнопку “Добавление ветки”, после чего откроется новое окно (см. рисунок 5), в которой необходимо ввести узлы подключения ветки. В программе уже заложены три узла, создавать их отдельно не предусматривается. Введите в доступные поля номера узлов (1, 2 или 3), после чего нажмите кнопку “Указать”. Если все сделано правильно, на экране появиться сообщение об успешном подключении ветки (см. рисунок 6).

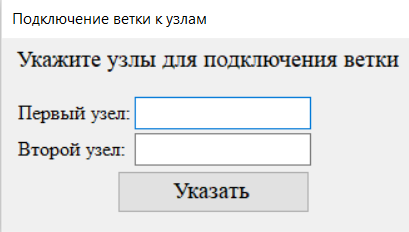


Рисунок 5 – Новое окно подключения ветки к узлам

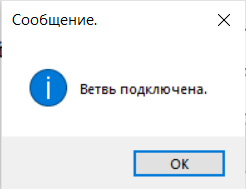


Рисунок 6 – Сообщение об удачном подключении ветки

Далее, откроется новое окно, в котором необходимо добавить в созданную ветку элементы (см. рисунок 7), такие как: резистор, ЭДС источник, лампу, ключ. Для добавления элемента, необходимо выбрать в поле “Выберите тип элемента” выбрать тип элемента из раскрывающегося списка. Затем необходимо заполнить все раскрывшиеся поля и нажать кнопку “Добавить элемент”. После удачного добавления элемента, заполненные поля очистятся. Если необходимо добавить еще один элемент, то повторяем описанные выше действия. Не обязательно, чтобы в ветке присутствовали все из перечисленных элементов. Необходимо, чтобы в ветке был как минимум один резистор. Вы не сможете закрыть данное окно, пока не будет добавлен резистор. После успешного добавления элементов, окно можно закрыть нажатием на кнопку “Закончить”.

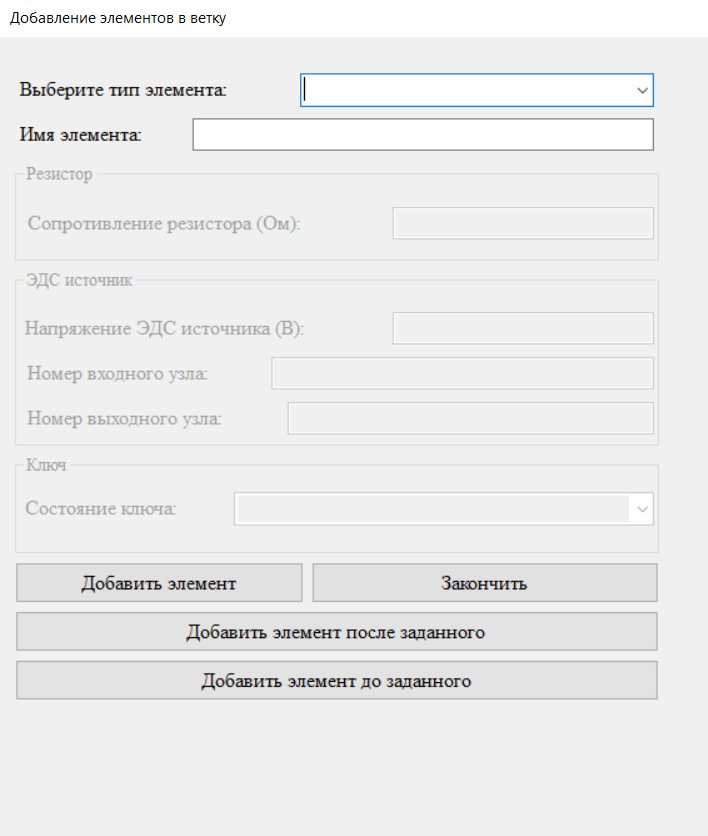


Рисунок 7 – Добавление элемента в ветку.

После успешного подключения ветки и добавления в нее элементов, вам стал доступен остальной функционал программы. Рассмотрим функционал кнопки добавления элемента в цепь.

При нажатии на кнопку “Добавить элемент в цепь” откроется новое окно (см. рисунок 8), в котором необходимо в доступном поле “Введите имя элемента” ввести имя элемента, который находится внутри ветки, в которую вы хотите добавить элемент. После ввода элемента, и нажатия кнопки “Выбрать”, выведется информация о данной ветке и станет доступно поле “Тип элемента”. Выбрав тип элемента и заполнив нужные поля, вы можете добавить элемент в цепь. Чтобы выбрать другую ветку для добавления элемента, необходимо нажать кнопку “Закончить” и ввести в поле “Имя элемента” имя элемента, который находиться в другой ветке. Для закрытия окна необходимо нажать на красный крестик вверху окна.

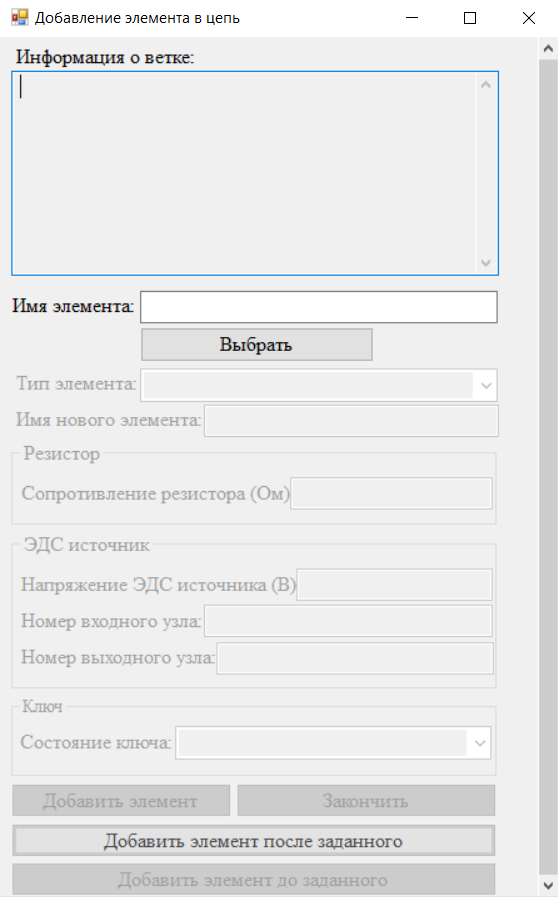


Рисунок 8 – Добавление нового элемента в ветку

Далее рассмотрим работу редактирования элемента. При нажатии на кнопку “Редактировать элемент”, откроется новое окно (см. рисунок 9). Для редактирования элемента необходимо вписать его имя в поле “Имя элемента для редактирования” и нажать кнопку “Выбрать элемент”. Если элемент был найден в цепи, то можно продолжить редактирование, иначе выйдет уведомление об ошибке (см. рисунок 10). Чтобы внести изменения в имя элемента, необходимо нажать кнопку “Изменить имя элемента”. Для внесения других изменений нужно нажать кнопку “Внести изменения”. Чтобы выбрать другой элемент для редактирования необходимо нажать на кнопку “Выбрать другой элемент”. Чтобы закончить редактирование элементов и выхода из окна редактирования, нажмите на кнопку “Закончить”.

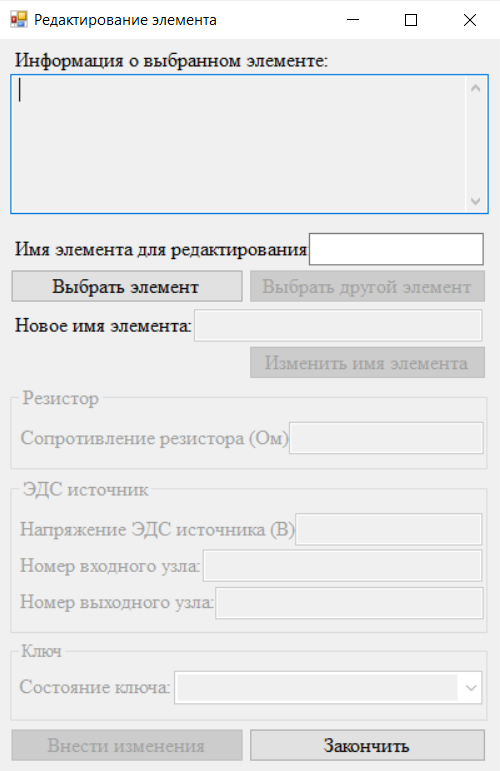


Рисунок 9 – Окно редактирования элемента

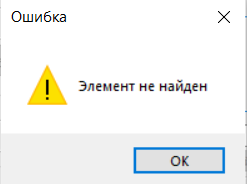


Рисунок 10 – Уведомление об ошибке

Для сохранения электрической цепи, необходимо нажать на кнопку “Сохранить цепь в файл”. При нажатии на данную кнопку откроется окно (см. рисунок 11), в которое необходимо вписать название файла, в который необходимо сохранить цепь. Программа не позволит сохранить цепь в файле под уже использованным именем. Имя файла должно всегда заканчиваться на .xml. Для загрузки электрической цепи из файла необходимо в главном меню приложения нажать на кнопку “Загрузить цепь из файла”. После нажатия откроется новое окно (см. рисунок 12). Для загрузки цепи из файла необходимо вписать имя этого файла и в конце имени добавить .xml.

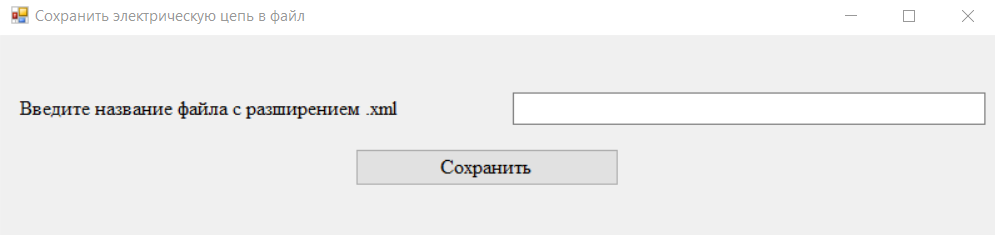


Рисунок 11 – Окно сохранения цепи в файл

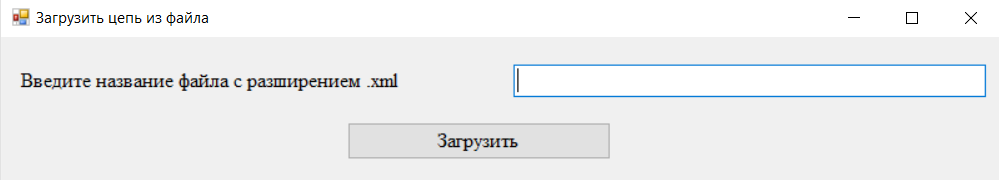


Рисунок 12 – Окно загрузки цепи из файла

Для того, чтобы вывести информацию о созданной цепи, необходимо нажать кнопку “Вывести информацию”, после чего на главный экран отобразиться информация о цепи. Пример отображения информации представлен на рисунке 13.

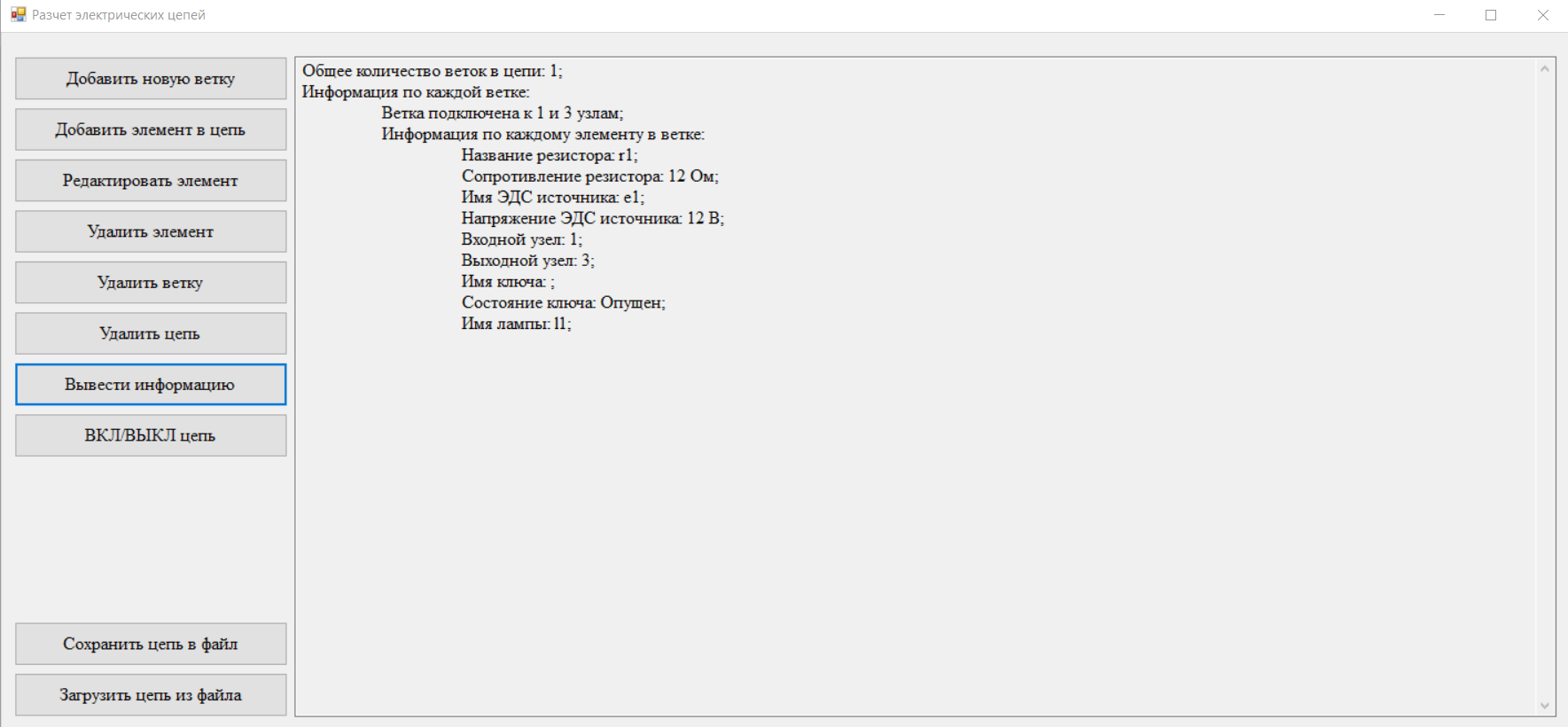


Рисунок 13 – Пример отображения информации о созданной цепи

Для расчета силы тока и потенциалов узлов, необходимо нажать кнопку “ВКЛ/ВЫКЛ цепь”. После ее нажатия на главном меню отобразиться вся необходимая информация (см. рисунок 14).

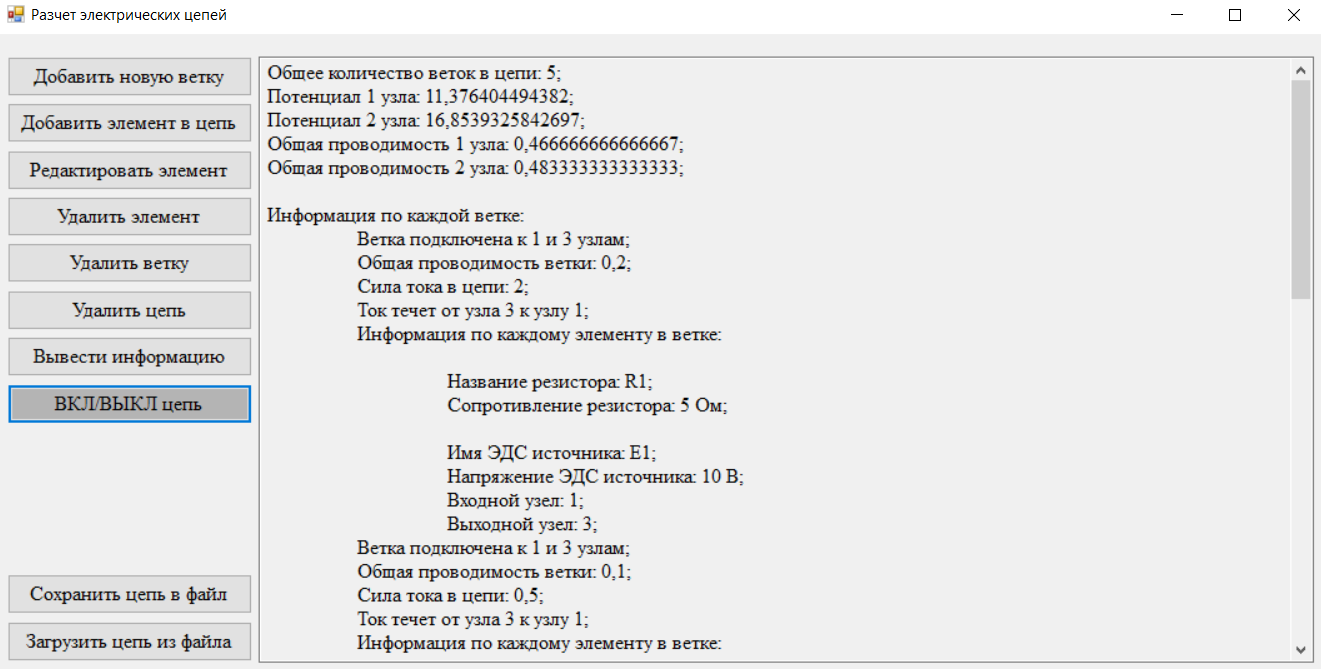


Рисунок 14 – Пример отображения информации о созданной цепи

**Заключение**

Рассмотрим результаты выполнения задания, целью которого являлось изучить технологию объектно-ориентированного подхода к разработке программ в Visual Studio .Net, освоить использование элементов графического интерфейса Windows Forms для управления работой приложения.

Была разработана диаграмма классов для данной предметной области.

Было разработано программное средство для расчета электрических цепей.

Тестирование программного средства показало, что программа успешно выполняет поставленную задачу.

Таким образом, поставленные инженерные задачи выполнены, цель курсовой работы достигнута.

# Список использованных источников

1 Матвеев А. Н. Электричество и магнетизм: учебное пособие. — М.: Высшая школа, 1983. – 463 с.

2  Нейман Л. Р., Демирчян К. С. Теоретические основы электротехники: в 2-х т. Учебник для вузов. Том I. — 3-е изд., перераб. и доп. — Л.: Энергоиздат. Ленингр. отд-ние, 1981. — 536 с.

3 Эндрю Троелсен, Филипп Джепикс. Язык программирования С# 7 и платформы .Net и .Net Core, 8-е изд.: Книга. Пер. с англ. – СПб.: OOO “Диалектика”. 2018 – 1328 с - 867 с.

**Приложение А**

***(обязательное)***

**Программный код**

Файл Branche.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml.Serialization;

namespace OOP\_KR

{

[Serializable]

public class Branche : IDirection

{

private double currentBranch=0;

private double conductivityBranch=0;//??????????

private Node incumingNode=null;

private Node outcumingNode=null;

public CollectionElements elements = new CollectionElements();

public Branche()

{}

public Node IncumingNode

{

get => incumingNode;

set => incumingNode = value;

}

public Node OutcummingNode

{

get => outcumingNode;

set => outcumingNode = value;

}

public double CurrentBranch

{

get => currentBranch;

set => currentBranch = value;

}

public double ConductivityBranch { get => conductivityBranch; set => conductivityBranch = value; }

public void CalculateСonductivity(Branche br)

{

foreach (Elements el in elements)

{

if (el is Resister)

{

Resister r = (Resister)el;

conductivityBranch += r.Conductibvity;

}

}

}

public void CalculatecurrentBranch()

{

currentBranch = outcumingNode.Potencial - incumingNode.Potencial;

foreach (Elements el in elements)

{

if (el is CCS)

{

CCS c = (CCS)el;

currentBranch = (currentBranch + c.Voltage) \* conductivityBranch;

return;

}

}

currentBranch = currentBranch \* conductivityBranch;

}

}

}

Файл ССS.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

[Serializable]

public class CCS : Elements, IDirection

{

private Node incumingNode=null;

private Node outcumingNode=null;

public Node IncumingNode

{

get => incumingNode;

set => incumingNode = value;

}

public Node OutcummingNode

{

get => outcumingNode;

set => outcumingNode = value;

}

public CCS(string name, double voltage):base(name)

{

Voltage = voltage;

}

public CCS() { }

public override string ToString()

{

string s = $"Имя ЭДС источника: {this.Name};\r\nНапряжение ЭДС источника: {this.Voltage};\r\nВходной узел: {this.incumingNode.Name};\r\nВыходной узел: {this.OutcummingNode.Name};";

return s;

}

}

}

Файл CollectionBRanches.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

[Serializable]

public class CollectionBranches:IEnumerable

{

private List<Branche> branches = new List<Branche>();

public List<Branche> Branches { get => branches; set => branches = value; }

public CollectionBranches() { }

public void Add(Object obj)

{

Branche br = (Branche)obj;

branches.Add(br);

}

public void AddBranche(Branche br)

{

Branches.Add(br);

}

public void DeleteListBranch()

{

for(int i = branches.Count - 1; i >= 0; i--)

{

DeleteBranche(branches[i]);

}

if (branches.Count == 0)

{

branches.Clear();

}

}

public void DeleteBranche(Branche br)

{

br.elements.DeleteListElements();

branches.Remove(br);

}

public int Count()

{

int i = 0;

foreach (Branche br in Branches)

{

i++;

}

return i;

}

public IEnumerator GetEnumerator()

{

foreach (Branche br in Branches)

{

yield return br;

}

}

}

}

Файл CollectionElements.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

[Serializable]

public class CollectionElements:IEnumerable

{

private List<Elements> Elements = new List<Elements>();

public CollectionElements() { }

public void AddElementAfter(Elements after, Elements el)

{

if (Elements.Count == 0)

{

return;

}

if (Elements[Elements.Count-1]==after)

{

Elements.Add(el);

return;

}

int i = 0;

foreach(Elements a in Elements)

{

i++;

if (a.Name == after.Name)

{

Elements.Insert(i + 1, el);

return;

}

}

return;

}

public void DeleteListElements()

{

Elements.Clear();

}

public void DeleteElement(Elements el)

{

Elements.Remove(el);

}

public int CountResistor()

{

int i = 0;

foreach(Elements el in Elements)

{

if (el is Resister)

{

i++;

}

}

return i;

}

public void AddElementBefore(Elements before, Elements el)

{

if (Elements.Count == 0)

{

return;

}

if (Elements[0] == before)

{

Elements.Insert(0, el);

return;

}

int i = 0;

foreach(Elements a in Elements)

{

i++;

if (a.Name == before.Name)

{

Elements.Insert(i, el);

return;

}

}

return;

}

public void AddElementLast(Elements el)

{

Elements.Add(el);

return;

}

public void AddElementHead(Elements el)

{

Elements.Insert(0, el);

return;

}

public int Count()

{

int count = 0;

foreach (Elements el in Elements)

{

count++;

}

return count;

}

public void Add(Object obj)

{

Elements el = (Elements)obj;

Elements.Add(el);

}

public IEnumerator GetEnumerator()

{

foreach(Elements el in Elements)

{

yield return el;

}

}

}

}

Файл ElectricalCircuit.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Xml.Serialization;

namespace OOP\_KR

{

[Serializable, XmlInclude(typeof(Node)), XmlInclude(typeof(CollectionBranches)), XmlInclude(typeof(CollectionElements)), XmlInclude(typeof(Branche)), XmlInclude(typeof(Resister)), XmlInclude(typeof(Lamp)), XmlInclude(typeof(Key)), XmlInclude(typeof(CCS))]

public class ElectricalСircuit

{

public CollectionBranches branches = new CollectionBranches();

private Node fNode=new Node("1");

private Node sNode= new Node("2");

private Node tNode=new Node("3");

public Node FNode { get=>fNode; set=>fNode=value; }

public Node SNode { get=>sNode; set=>sNode=value; }

public Node TNode { get=>tNode; set=>tNode=value; }

public ElectricalСircuit()

{

}

public ElectricalСircuit(CollectionBranches br) { }

public static ElectricalСircuit RestoreDocuments(string nameFile)

{

FileStream reader = new FileStream(nameFile, FileMode.OpenOrCreate, FileAccess.Read);

XmlSerializer deserializer = new XmlSerializer(typeof(ElectricalСircuit));

ElectricalСircuit documents;

try

{

documents = (ElectricalСircuit)deserializer.Deserialize(reader);

}

catch (Exception ex)

{

documents = new ElectricalСircuit(new CollectionBranches());

}

reader.Close();

return documents;

}

public static void SaveDocuments(ElectricalСircuit electricalСircuit, string nameFile)

{

StreamWriter writer = new StreamWriter(nameFile);

XmlSerializer serializer = new XmlSerializer(typeof(ElectricalСircuit));

serializer.Serialize(writer, electricalСircuit);

writer.Close();

}

public void СountPotencial()

{

}

public void DeleteCircuit()

{

branches.DeleteListBranch();

}

}

}

Файл Elements.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml.Serialization;

namespace OOP\_KR

{

[Serializable]

abstract public class Elements

{

private string name;

private double voltage=0;

public double Voltage

{

set

{

voltage = value;

}

get

{

return voltage;

}

}

public string Name

{

set

{

name = value;

}

get

{

return name;

}

}

public Elements()

{

}

public Elements(string name)

{

Name = name;

}

}

}

Файл IDirection.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

public interface IDirection

{

Node IncumingNode { get; set; }

Node OutcummingNode { get; set; }

}

}

Файл Key.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

[Serializable]

public class Key : Elements

{

public bool active = true;

public Key() { }

public Key(string name) : base(name)

{ }

public override string ToString()

{

string s = $"Имя ключа: {this.Name};\r\nСостояние ключа: ";

if (active)

{

s += "Опущен;";

return s;

}

s += "Поднят;";

return s;

}

}

}

Файл Lamp.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

[Serializable]

public class Lamp : Elements

{

public bool isOn = false;

public Lamp() { }

public Lamp(string name) : base(name) { }

}

}

Файл Node.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

[Serializable]

public class Node

{

private string name="";

private double conductivityNode=0;

private double potencial=0;

private double sumEg = 0;

public double ConductivityNode { get => conductivityNode; set => conductivityNode = value; }

public double Potencial { get => potencial; set => potencial = value; }

public string Name { get => name; set => name = value; }

public double SumeEg { get => sumEg; set => sumEg = value; }

public void CalculateEg(ElectricalСircuit elcir)

{

foreach(Branche br in elcir.branches)

{

if (br.IncumingNode.Name == name || br.OutcummingNode.Name == name)

{

bool key = true;

foreach(Elements el in br.elements)

{

if (el is Key)

{

Key k = (Key)el;

if (k.active)

{

key = true;

}

else

key = false;

break;

}

}

if (key)

{

foreach (Elements el in br.elements)

{

if (el is CCS)

{

CCS c = (CCS)el;

if (c.IncumingNode.Name == name)

{

sumEg += c.Voltage \* br.ConductivityBranch;

}

else

{

sumEg -= c.Voltage \* br.ConductivityBranch;

}

}

}

}

}

}

}

public void CalculateConductivityNode(ElectricalСircuit elcir)

{

bool key = true;

foreach (Branche br in elcir.branches)

{

if (br.IncumingNode.Name == name || br.OutcummingNode.Name == name)

{

foreach(Elements el in br.elements)

{

if (el is Key)

{

Key k = (Key)el;

if (k.active)

{

key = true;

}

else

{

key = false;

}

break;

}

}

if (key)

conductivityNode += br.ConductivityBranch;

else

{

continue;

}

}

}

}

public Node() { }

public Node(string name)

{

this.name = name;

}

}

}

Файл Resister.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace OOP\_KR

{

[Serializable]

public class Resister : Elements

{

private double resistance=0;

private double conductivity=0;

public double Resistance

{

set

{

resistance = value;

conductivity = 1 / resistance;

}

get

{

return resistance;

}

}

public double Conductibvity

{

get => conductivity;

}

public Resister() { }

public Resister(string name, double res) : base(name)

{

Resistance = res;

conductivity = 1 / resistance;

}

public override string ToString()

{

string s = $"Имя резистора: {this.Name};\r\nСопротивление резистора {this.resistance} Ом;";

return s;

}

}

}