|  |  |
| --- | --- |
| Институт (факультет) | Информационных технологий |
| Кафедра | Математического и программное обеспечение ЭВМ |

КУРСОВАЯ РАБОТА

|  |
| --- |
| по дисциплине Объектно-ориентированное программирование |

|  |  |
| --- | --- |
| на тему | “Объектно-ориентированное программирование на языке С++” |
|  | |

|  |
| --- |
| Выполнил студент группы 1ПИб-02-2оп-22 |
| *группа* |
| направления подготовки (специальности) |
| 09.03.04 Программная инженерия |
| *шифр, наименование* |
| Зернов Владислав Александрович |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| **Матевосян Ремик Артурович** |
| *фамилия, имя, отчество* |
| **Доцент** |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2024

Оглавление

[Введение 3](#_Toc105245094)

[Объектно-ориентированный анализ предметной области 5](#_Toc105245095)

[Проектирование классов 7](#_Toc105245096)

[Логическая структура программы 11](#_Toc105245097)

[Модульная структура программы 13](#_Toc105245098)

[Тестирование 15](#_Toc105245099)

[Заключение 19](#_Toc105245100)

[Список литературы 20](#_Toc105245101)

[Приложение 1 21](#_Toc105245102)

[Приложение 2 27](#_Toc105245103)

[Приложение 3 28](#_Toc105245104)

## Введение

Объектно-ориентированное программирование – программирование, ориентированное на данные, где операции неразрывно связаны. Вместе данные и операции над ними представляют собой классы.

Объектно-ориентированные программа – программа, которая представляет собой совокупность объектов, взаимодействующий посредством передачи сообщения.

Классы – это элемент ПО, описывающий абстрактный тип данных и его частичную или полную реализацию.

Объекты – экземпляры классов. Здесь речь идет о конкретном представлении объекта в виде абстракции с собственными свойствами и методами. Объекты, созданные на основе одного класса, называются экземплярами этого класса. Объекты этого класса имеют разные характеристики, свойства, но все равно будут считаться объектами одного класса.

Объектно-ориентированное программирование позволяет разложить проблему на составные части, каждая из которых становится самостоятельным объектом. В каждом из объектов содержится свой собственный код и данные, относящиеся к этому объекту.

Основными принципами организации в объектно-ориентированном программирование являются различные аспекты базового понимания предметной задачи, которые необходимы для оптимального управления соответствующей моделью:

* Абстракция. Скрыть детали и описать вещи простыми словами. Например, метод складывает два целых чисел, его внутренняя логика скрыта от внешнего мира: он просто берет на вход два числа и на выходе дает результат сложения. Существует множество способов достижения абстракции в объектно-ориентированном программировании, например: инкапсуляция и наследование.
* Инкапсуляция. Это метод достижения абстракции в объектно- ориентированном программировании. Инкапсуляция используется для того, чтобы ограничить доступ к членам класса и методам. Одним из способов достижения инкапсуляционного состояния в ООП является модификатор доступа
* Наследование. Для быстрой и безопасной организации родственных представлений: чтобы было достаточно на каждом иерархичном шаге учитывать лишь изменение, а не все остальные, учтенные на предыдущих шагах.
* Полиморфизм. Данная концепция позволяет объекту вести себя по-разному при разных обстоятельствах. При компиляции и выполнении, есть два типа полиморфизма: полиморфизмы в процессе компиляции и выполнения. В процессе компиляции полиморфизм в процессе компиляции достигаются за счет перегрузки метода.

Актуальность изучения концепции ООП заключается, прежде всего, в том, что ООП является востребованной парадигматической концепцией программирования при разработке программного обеспечения что обуславливает программисту необходимость понимания и применения ООП.

Цель данной курсовой работы является изучение методов, моделирование иерархий классов родственных типов и освоение объектно-ориентированного программирования на языке С++.

## Объектно-ориентированный анализ предметной области

Полученный вариант задания – воздушный транспорт. На основании этого задания и создаётся иерархия родственных классов.

Программа содержит в себе базовый класс «ElectricalMachine», наследуемый класс «Generator», от которого наследуются «DieselGenerator» и «GasGenerator», от которых затем наследуется «MobileGenerator». Эти классы моделируют структуру реальных объектов, таких как электрической машины, генератора и т.д. Также программа содержит класс «MyVector», являющийся динамическим вектором. Он служит для хранения вышеперечисленных объектов. Иерархия классов представлена на рис. 1.

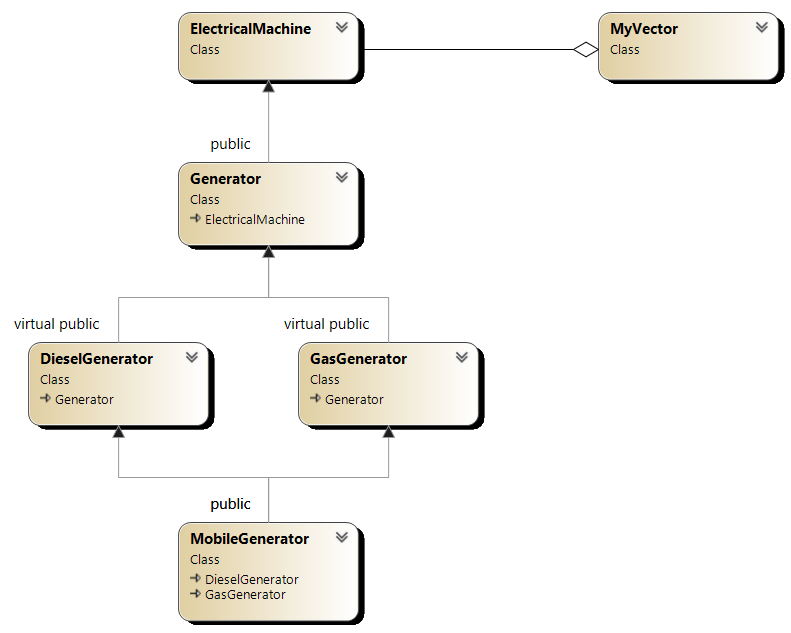


Рис. 1. Контекстная диаграмма классов

Краткое описание классов:

* Электрическая машина (ElectricalMachine) – электромеханический преобразователь, который преобразует механическую энергию в электрическую;
* Генератор (Generator) – источник электрической энергии;
* Дизельный генератор (DieselGenerator) – генератор, работающий на дизельном топливе;
* Газовый генератор (GasGenerator) – генератор, работающий на газу;
* Переносной генератор (MobileGenerator) – небольшой генератор, применяющийся в основном для бытовых целей.

## Проектирование классов

Детальная диаграмма классов представлена на рис. 2.

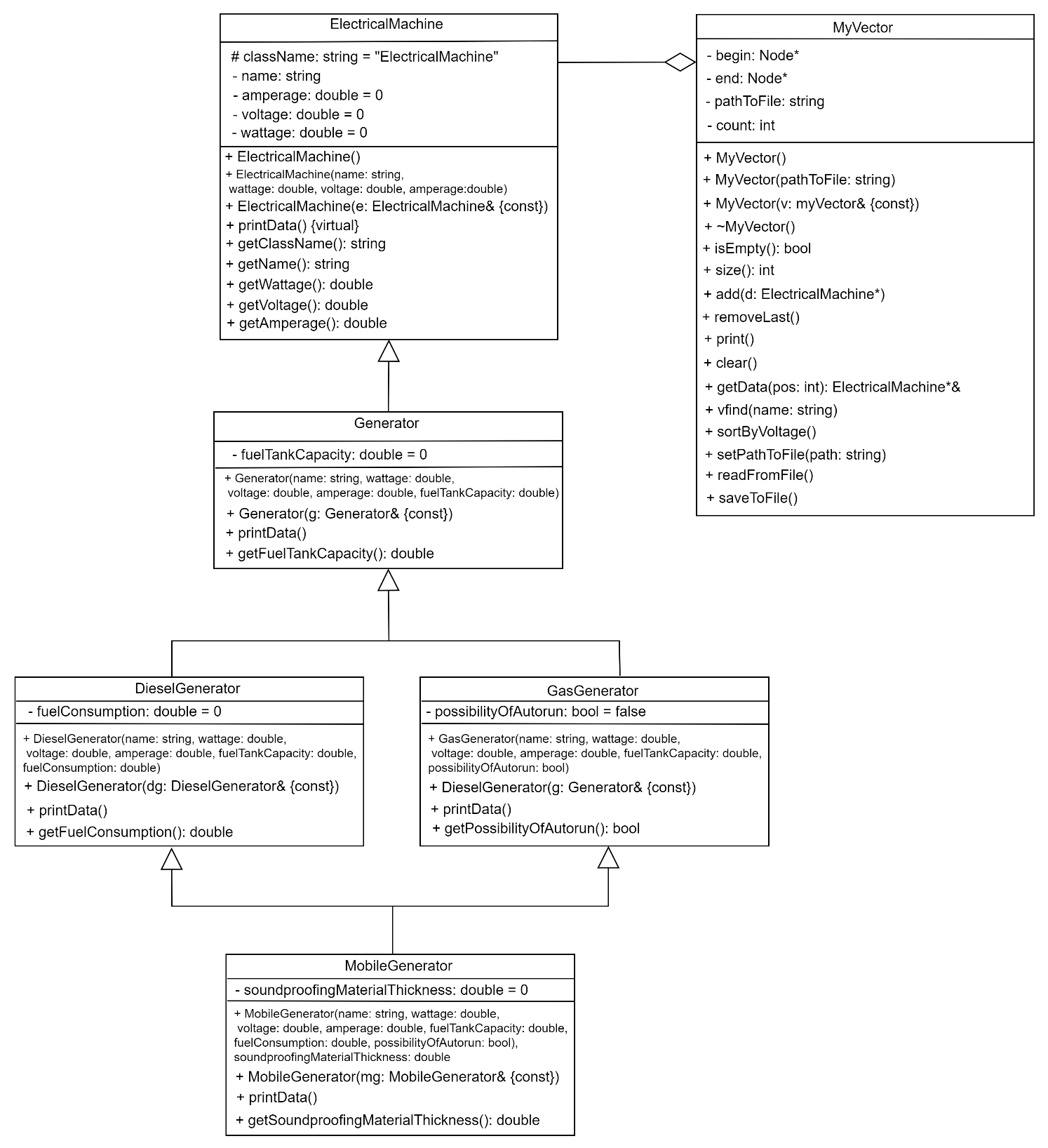


Рис. 2. Детальная диаграмма классов

Описание элементов классов:

* ElectricalMachine:
  + Закрытые поля name, amperage, voltage, wattage, защищенное поле className;
  + ElectricalMachine() – конструктор по умолчанию;
  + ElectricalMachine(string name, double wattage, double voltage, double amperage) – конструктор с параметрами;
  + ElectricalMachine(const ElectricalMachine& e) – копирующий конструктор;
  + virtual void printData() const – вывод содержимого класса на экран;
  + string getClassName() const – возвращает значение поля className, содержащее название класса;
  + string getName() const – возвращает значение поля name, содержащее имя объекта;
  + double getWattage() const – возвращает значение поля wattage, определяющего мощность;
  + double getVoltage() const – возвращает значение поля voltage, определяющего напряжение;
  + double getAmperage() const – возвращает значение поля amperage, определяющего силу тока.
* Generator:
  + Закрытое поле fuelTankCapacity.
  + Generator(string name, double wattage, double voltage, double amperage, double fuelTankCapacity) – конструктор с параметрами;
  + Generator(const Generator& g) – копирующий конструктор;
  + void printData() const – вывод содержимого класса на экран;
  + double getFuelTankCapacity() const - возвращает значение поля fuelTankCapacity, определяющего вместимость топливного бака.
* DieselGenerator:
  + Закрытое поле fuelConsumption;
  + DieselGenerator(string name, double wattage, double voltage, double amperage, double fuelTankCapacity, double fuelConsumption) – конструктор с параметрами
  + DieselGenerator(const DieselGenerator& dg) – копирующий конструктор;
  + void printData() const – вывод содержимого класса на экран;
  + double getFuelConsumption() const - возвращает значение поля fuelConsumption, определяющего расход топлива.
* GasGenerator:
  + Закрытое поле possibilityOfAutorun;
  + GasGenerator(std::string name, double wattage, double voltage, double amperage, double fuelTankCapacity, bool possibilityOfAutorun) – конструктор по умолчанию;
  + GasGenerator(const GasGenerator& gg) – копирующий конструктор;
  + void printPossibilityOfAutorunField() const – выводит на экран содержимое поле possibilityOfAutorun;
  + void printData() const – вывод содержимого класса на экран;
  + double getPossibilityOfAutorun() const - возвращает значение поля possibilityOfAutorun, определяющего поддерживает ли генератор возможность автозапуска.
* MobileGenerator:
  + Закрытое поле soundproofingMaterialThickness;
  + MobileGenerator(string name, double wattage, double voltage, double amperage, double fuelTankCapacity, double fuelConsumption, bool possibilityOfAutorun, double soundproofingMaterialThickness) – конструктор с параметрами;
  + MobileGenerator(MobileGenerator& mg) – копирующий конструктор;
  + void printData() const – вывод содержимого класса на экран;
  + double getSoundproofingMaterialThickness() const - возвращает значение поля soundproofingMaterialThickness, определяющего толщину шумоизолирующего материала.
* MyVector:
  + Закрытая структура Node, содержащая в себе указатель на ElectricalMachine data, указатели на Node next и prev.
  + Закрытые поля begin и end(указатели на Node), pathToFile и count;
  + MyVector() – конструктор по умолчанию;
  + MyVector(string pathToFile) – конструктор с одним параметром;
  + MyVector(const MyVector& v) – копирующий конструктор;
  + ~MyVector() - деструктор;
  + bool isEmpty() const – проверка вектора на пустоту;
  + int size() – возвращает размер вектора;
  + void add(ElectricalMachine\* d) – добавление элемента в вектор;
  + void removeLast() – удаление элемента из вектора;
  + void print() const – вывод содержимого вектора на экран;
  + void clear() – удаление всех элементов из вектора;
  + ElectricalMachine\*& getData(int pos) const – возвращает элемент на заданной позиции;
  + void vfind(string name) const – поиск элемента по имени;
  + void sortByVoltage() – сортировка по полю voltage;
  + void setPathToFile(string path) – метод для указания пути к файлу, в котором хранятся данные из вектора;
  + void readFromFile() – чтение данных из файла;
  + void saveToFile() – запись данных в файл;

## Логическая структура программы

В программе присутствует пользовательский интерфейс, представленный в консольном окне. Через интерфейс пользователь взаимодействует с вектором, вводя команды, выполняющие определенные действия, такие как:

* вывод элементов вектора на экран,
* добавление нового элемента в вектор,
* удаление элемента из вектора, сортировка элементов в векторе,
* поиск элемента в векторе,
* сохранение элементов вектора в файл
* чтение элементов из файла.

Вектор содержит в себе объекты базового класса ElectricalMachine, от него в свою очередь наследуются классы Generator, GasGenerator, DieselGenerator и MobileGenerator.

Данные об объектах вектора по умолчанию находятся в файле data.txt, находящегося в папке с проектом. При желании пользователь может выбрать другой файл для сохранения. Если на момент записи этот файл отсутствует, он создастся автоматически.

Все элементы взаимодействия пользователя с программой представлены в функции main() модуля Main.

Логическая структура программы представлена на рис. 4.

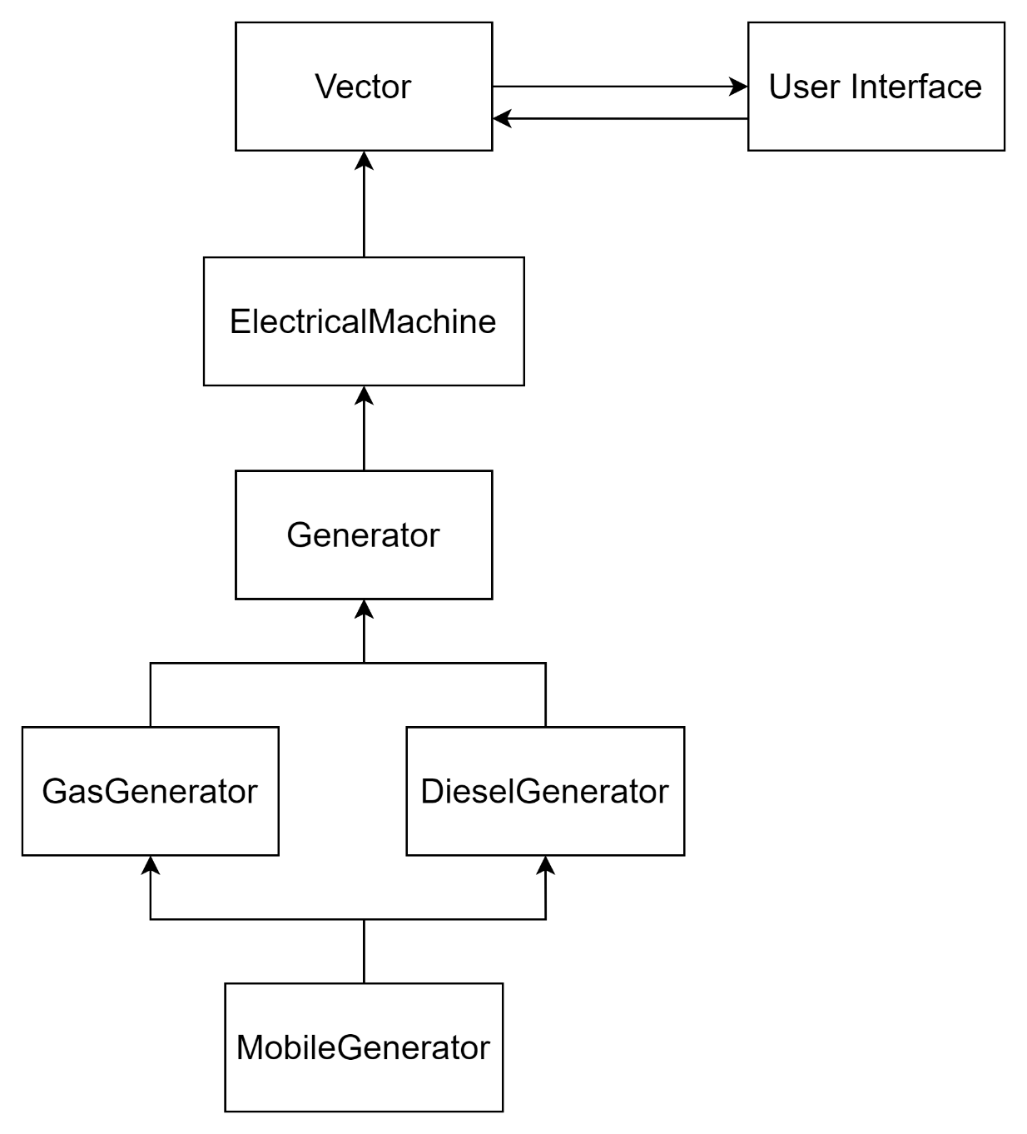


Рис. 4. Логическая структура программы

## Модульная структура программы

Модульная структура программы представляет собой древовидную структуру, в узлах которой размещаются программные модули, а направленные дуги показывают статическую подчиненность модулей. Если в тексте модуля имеется ссылка на другой модуль, то их на структурной схеме соединяет дуга, которая исходит из первого и входит во второй модуль.

Модульная структура представлена на рис. 4.

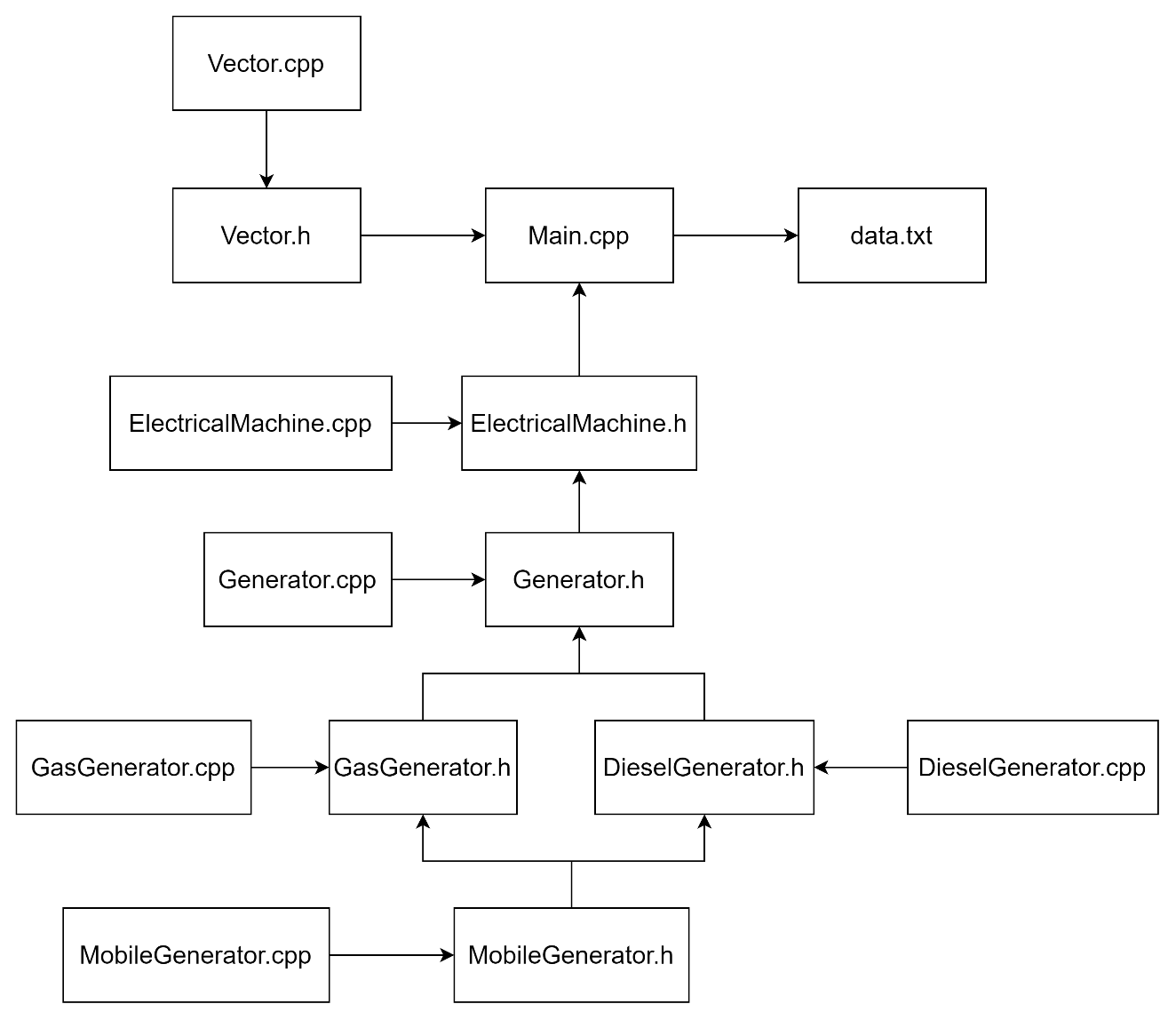


Рис. 4. Модульная структура программы

Программа состоит из нескольких модулей, каждый из которых содержит одноименные .h и .cpp файлы – ElectricalMachine, Generator, DieselGenerator, GasGenerator, MobileGenerator, Vector. Также программа содержит главный модуль – Main. Посредством этого модуля происходит взаимодействие пользователя с программой. Спецификация модулей представлена в Табл.1.

Таблица 1.

|  |  |  |
| --- | --- | --- |
| Модуль | Связанные файлы | Краткое описание |
| Main | Main.cpp | Содержит основную структуру программы, связывающую все в единое целое |
| Vector | MyVector.h, MyVector.cpp | Содержит определение и реализацию класса-контейнера по варианту (динамический вектор) |
| ElectricalMachine | ElectricalMachine.h, ElectricalMachine.cpp | Содержит определение и реализацию базового класса ElectricalMachine |
| Generator | Generator.h, Generator.cpp | Содержит определение и реализацию родительского класса Generator |
| DieselGenerator | DieselGenerator.h, DieselGenerator.cpp | Содержит определение и реализацию родительского класса DieselGenerator |
| GasGenerator | GasGenerator.h, GasGenerator.cpp | Содержит определение и реализацию родительского класса GasGenerator |
| MobileGenerator | MobileGenerator.h, MobileGenerator.cpp | Содержит определение и реализацию родительского класса MobileGenerator |

## Тестирование

Отчеты тестирования классов, внешних функций и требований технического задания приведены далее в табл. 2.1 - 2.6.

Таблица 2.1

Тестирование компонентных функций класса ElectricalMachine

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Функция | Входные данные | Результат | Ответственное лицо |
| 02.06.2022 | Конструктор по умолчанию | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | Конструктор с параметрами | string name, double wattage, double voltage, double amperage | Успех | Тестировщик |
| 02.06.2022 | Копирующий конструктор | const ElectricalMachine& e | Успех | Тестировщик |
| 02.06.2022 | printData | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getClassName | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getWattage | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getVoltage | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getAmperage | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getName | Отсутствуют | Успех | Тестировщик |

Таблица 2.2

Тестирование компонентных функций класса Generator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Функция | Входные данные | Результат | Ответственное лицо |
| 02.06.2022 | Конструктор с параметрами | string name, double wattage, double voltage, double amperage, double fuelTankCapacity | Успех | Тестировщик |
| 02.06.2022 | Копирующий конструктор | const Generator& g | Успех | Тестировщик |
| 02.06.2022 | printData | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getFuelTankCapacity | Отсутствуют | Успех | Тестировщик |

Таблица 2.3

Тестирование компонентных функций класса DieselGenerator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Функция | Входные данные | Результат | Ответственное лицо |
| 02.06.2022 | Конструктор с параметрами | string name, double wattage, double voltage, double amperage, double fuelTankCapacity, double fuelConsumption | Успех | Тестировщик |
| 02.06.2022 | Копирующий конструктор | const DieselGenerator& dg | Успех | Тестировщик |
| 02.06.2022 | printData | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getFuelConsumption | Отсутствуют | Успех | Тестировщик |

Таблица 2.4

Тестирование компонентных функций класса GasGenerator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Функция | Входные данные | Результат | Ответственное лицо |
| 02.06.2022 | Конструктор с параметрами | string name, double wattage, double voltage, double amperage, double fuelTankCapacity, bool possibilityOfAutorun | Успех | Тестировщик |
| 02.06.2022 | Копирующий конструктор | const GasGenerator& gg | Успех | Тестировщик |
| 02.06.2022 | printData | Отсутствуют | Успех | Тестировщик |
| 02.06.2022 | getPossibilityOfAutorun | Отсутствуют | Успех | Тестировщик |

Таблица 2.5

Тестирование компонентных функций класса MobileGenerator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Функция | Входные данные | Результат | Ответственное лицо |
| 03.06.2022 | Конструктор с параметрами | string name, double wattage, double voltage,  double amperage, double fuelTankCapacity, double fuelConsumption, bool possibilityOfAutorun, double soundproofingMaterialThickness | Успех | Тестировщик |
| 03.06.2022 | Копирующий конструктор | const MobileGenerator& mg | Успех | Тестировщик |
| 03.06.2022 | printData | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | get SoundproofingMaterialThickness | Отсутствуют | Успех | Тестировщик |

Таблица 2.6

Тестирование компонентных функций класса MyVector

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Функция | Входные данные | Результат | Ответственное лицо |
| 03.06.2022 | Конструктор по умолчанию | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | Конструктор с параметрами | string pathToFile | Успех | Тестировщик |
| 03.06.2022 | Копирующий конструктор | const MyVector& v | Успех | Тестировщик |
| 03.06.2022 | isEmpty | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | size | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | add | ElectricalMachine\* d | Успех | Тестировщик |
| 03.06.2022 | removeLast | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | print | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | clear | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | getData | int pos | Успех | Тестировщик |
| 03.06.2022 | vfind | string name | Успех | Тестировщик |
| 03.06.2022 | sortByVoltage | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | setPathToFile | string path | Успех | Тестировщик |
| 03.06.2022 | readFromFile | Отсутствуют | Успех | Тестировщик |
| 03.06.2022 | SaveToFile | Отсутствуют | Успех | Тестировщик |

## Заключение

Результатом выполнения курсовой работы стала объектно-ориентированная программа, предназначенная для работы с небольшой базой посвященной электрическим машинам. В программе предусмотрена обработка нескольких исключительных ситуаций, связанных с неподходящими значениями для переменных. В рамках данной работы проводилась работа с механизмами одиночного и множественного наследования, указателями. В ходе работы были получены навыки разработки и отладки объектно-ориентированной программы, а также освоены новые особенности разработки ООП программ.

# Список литературы

1. Ершов Е.В., Виноградова Л.Н. «Методика и организация самостоятельной работы студентов».
2. Фаулер, М. UML Основы. 3-е издание. Пер. с англ. / М. Фаулер – СПб.: «Символ-Плюс», 2004 – 192с
3. Страуструп, Б. Язык программирования С++. Специальное издание. Пер. с англ. / Бьерн Страуструп – М.: «Бином», 2019. – 1136 с.: ил. Скотт Мейерс — Эффективное использование STL.
4. Программирование [Электронный ресурс], статья «UML–диаграммы классов». URL: https://prog-cpp.ru/uml-classes/

## Приложение 1

минобрнауки россии

федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Информационных технологий |
| наименование института (факультета) |
| Математическое и программное обеспечение ЭВМ |
| наименование кафедры |
| Объектно-ориентированное программирование |

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ

д. т.н. \_\_\_\_\_\_\_\_\_ Ершов Е.В.

« » 20 г.

Объектно-ориентированное программирование на языке С++

Техническое задание на курсовую работу

Листов \_

Руководитель: Шаханов Никита Иванович

Исполнитель: студент гр. 1ИСб-00-1оп-20

Ракитский И. С.

2022 г.

Введение

Тема курсовой работы: Объектно-ориентированное программирование на языке С ++.

Цель: написать приложение, использующее механизмы объектно-ориентированного программирования, согласно разработанной иерархии родственных классов

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине "Объектно-ориентированное программирование", выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения:

Наименование темы разработки: «Объектно-ориентированное программирование на языке С++»

1. Назначение разработки

Разработка приложения предназначена для закрепления знаний по дисциплине «Объектно-ориентированное программирование».

1. Требования к программе
   1. Требования к функциональным характеристикам:

* Разработать иерархию родственных типов для моделирования и обработки данных предметной области электрические машины (A22);
* Для хранения объектов каждого производного класса использовать структуру данных динамический вектор (C2);
* Реализовать функции обработки данных (сортировка и поиск по выбранным полям и задаваемым диапазонам значений, другие функции, в том числе перегруженные);
* Реализовать файловый ввод/вывод и ввод данных с клавиатуры, вывод данных на дисплей;
* Предусмотреть обработку различных исключительных ситуаций;
  1. Требования к надёжности

В ходе разработки выявить все недостатки, при которых возникают проблемы в работе программы, устранить их, а также обеспечить защиту при попытках пользователя совершить то или иное действие, которое может нарушить работу приложения.

* 1. Условия эксплуатации

Условия эксплуатации ограничены условиями эксплуатации ПК.

3.4. Требования к составу и параметрам технических средств

Минимальные требования к составу и параметрам технических средств:

* оперативная память 512 Мб или больше;
* свободное место на жестком диске от 120 Мб;
* клавиатура;
* мышь;
* монитор.
  1. Требования к информационной и программной совместимости

Для выполнения всех действий, связанных с работой программы, подходят такие операционные системы как Windows 7,8 и 10. Текст программы реализуется на языке высоко уровня – C++ с помощью программы Visual Studio (версия программы желательно 2017г. и выше). Для нормальной работы программы требуются библиотеки, которые использует компилятор.

* 1. Требования к программной документации

Программная документация должна содержать расчётно-пояснительную записку с содержанием:

* расчетно-пояснительная записка;
* руководство пользователя;
* текст программы.

1. Стадии и этапы разработки

Стадии и этапы разработки с отметками о выполнении подробно приведены в Табл.П.1.1.

Табл.П.1.1

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки ПО | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Разработка технического задания | До 20.05.2022 | Оформленное техническое задание |  |
| Объектно-ориентированный анализ предметной области | До 22.05.2022 | Понимание предметной области |  |
| Проектирование классов | До 23.05.2022 | Создана иерархия классов и реализован алгоритм работы приложения |  |
| Создание логической структуры программы | До 24.05.2022 | Создана логическая структура программы |  |
| Создание модульной структуры программы | До 25.05.2022 | Создана модульная структура программы |  |
| Создание программного обеспечения | До 29.05.2022 | Программное обеспечение создано |  |
| Тестирование программного обеспечения | До 29.05.2022 | Программное обеспечение протестировано |  |
| Создание расчетно-пояснительной записки | До 30.05.2022 | Расчетно-пояснительная записка написана |  |
| Представление расчетно-пояснительной записки | До 1.06.2022 | Расчетно-пояснительная записка представлена |  |

5. Порядок контроля и приемки

Порядок контроля и приемки приведен в Табл.П.1.2.

Табл.П.1.2

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой работы | Сроки контроля | Результат выполнения | Отметка о приемке результата контрольного этапа |
| Получение и утверждение задания и написание технического задания | До 01.02.2022 | Задание получено |  |
| Сбор и анализ информации для написания курсовой работы | До 15.03.2022 | Информация собрана |  |
| Программирование приложения | До 29.05.2022 | Приложение написано |  |
| Написание расчетно-пояснительной записки | До 30.05.2022 | Расчетно-пояснительная записка написана |  |
| Сдача расчетно-пояснительной записки преподавателю | До 04.06.2022 | Расчетно-пояснительная записка сдана |  |
| Защита курсовой работы перед преподавателем | До 05.06.2022 | Расчетно-пояснительная записка защищена |  |

## Приложение 2

Руководство пользователя

1. Запустить приложение двойным кликом по coursework\_program.exe (рис. П2.1)

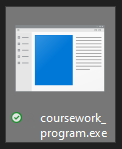


Рис. П2.1. Приложение для запуска

1. После запуска появится консольное окно, в котором нужно вписывать команды для взаимодействия с программой (список команд появляется при запуске программы (рис. П2.2)

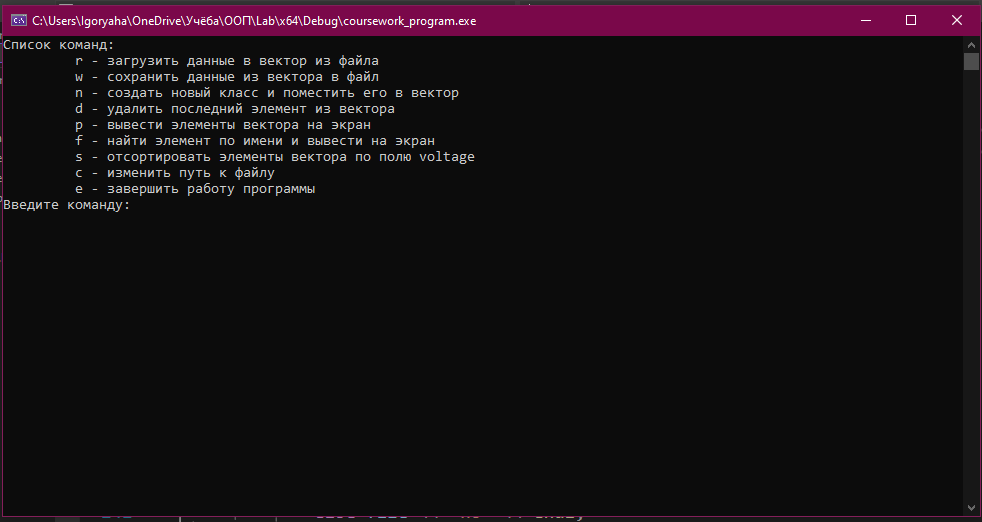


Рис. П2.2 Окно приложения

## Приложение 3

Текст программы

Файл Main.cpp:

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include "MyVector.h"

#include "ElectricalMachine.h"

#include "Generator.h"

#include "DieselGenerator.h"

#include "GasGenerator.h"

#include "MobileGenerator.h"

using namespace std;

int main() {

setlocale(0, "");

string filename = "data.txt";

MyVector vec(filename);

cout << "Список команд:\n\t r - загрузить данные в вектор из файла\n\t w - сохранить данные из вектора в файл\n\t n - создать новый класс и поместить его в вектор\n\t d - удалить последний элемент из вектора\n\t p - вывести элементы вектора на экран\n\t f - найти элемент по имени и вывести на экран\n\t s - отсортировать элементы вектора по полю voltage\n\t с - изменить путь к файлу\n\t e - завершить работу программы\n";

cout << "Введите команду: ";

char input;

cin >> input;

while (input != 'e') {

switch (input)

{

case 'r':

try {

vec.clear();

vec.readFromFile();

cout << "Данные успешно загружены" << endl;

}

catch (exception& ex) {

cout << ex.what() << endl;

}

break;

case 'w':

try {

vec.saveToFile();

cout << "Данные успешно сохранены в файл." << endl;

}

catch (exception& ex) {

cout << ex.what() << endl;

}

break;

case 'n':

int classSelectionSymbol;

cout << "Выберите добавляемый класс (1 - ElectricalMachine, 2 - Generator, 3 - DieselGenerator, 4 - GasGenerator, 5 - MobileGenerator): ";

cin >> classSelectionSymbol;

char name[30]; double wattage, voltage, amperage;

cout << "Введите имя объекта: "; cin >> name;

cout << "Введите мощность: "; cin >> wattage;

cout << "Введите напряжение: "; cin >> voltage;

cout << "Введите силу тока: "; cin >> amperage;

double fuelTankCapacity;

double fuelConsumption;

char possibilityStr[5];

bool possibilityOfAutorun;

double soundproofingMaterialThickness;

try {

switch (classSelectionSymbol)

{

case 1:

vec.add(new ElectricalMachine(name, wattage, voltage, amperage));

cout << "Класс успешно добавлен." << endl;

break;

case 2:

cout << "Введите вместимость топливного бака: ";

cin >> fuelTankCapacity;

vec.add(new Generator(name, wattage, voltage, amperage, fuelTankCapacity));

cout << "Класс успешно добавлен." << endl;

break;

case 3:

cout << "Введите вместимость топливного бака: ";

cin >> fuelTankCapacity;

cout << "Введите расход топлива: ";

cin >> fuelConsumption;

vec.add(new DieselGenerator(name, wattage, voltage, amperage, fuelTankCapacity, fuelConsumption));

cout << "Класс успешно добавлен." << endl;

break;

case 4:

cout << "Введите вместимость топливного бака: ";

cin >> fuelTankCapacity;

cout << "Возможность автозапуска (yes или no): ";

cin >> possibilityStr;

possibilityOfAutorun = strcmp(possibilityStr, "yes") == 0 ? true : false;

vec.add(new GasGenerator(name, wattage, voltage, amperage, fuelTankCapacity, possibilityOfAutorun));

cout << "Класс успешно добавлен." << endl;

break;

case 5:

cout << "Введите вместимость топливного бака: ";

cin >> fuelTankCapacity;

cout << "Введите расход топлива: ";

cin >> fuelConsumption;

cout << "Возможность автозапуска (yes или no): ";

cin >> possibilityStr;

possibilityOfAutorun = strcmp(possibilityStr, "yes") == 0 ? true : false;

cout << "Толщина звукоизоляции: ";

cin >> soundproofingMaterialThickness;

vec.add(new MobileGenerator(name, wattage, voltage, amperage, fuelTankCapacity, fuelConsumption, possibilityOfAutorun, soundproofingMaterialThickness));

cout << "Класс успешно добавлен." << endl;

break;

default:

cout << "Ошибка: неверное название класса." << endl;

break;

}

}

catch (exception& ex) {

cout << ex.what() << endl;

}

break;

case 'd':

vec.removeLast();

cout << "Последний элемент удален." << endl;

break;

case 'p':

cout << "Данные в векторе:" << endl;

vec.print();

break;

case 'f':

char nameForFind[30];

cout << "Введите имя искомого объекта: ";

cin >> nameForFind;

vec.vfind(nameForFind);

break;

case 's':

vec.sortByVoltage();

cout << "Элементы вектора успешно отсортированы." << endl;

break;

case 'c':

char path[256];

cout << "Введите новый путь к файлу: ";

cin >> path;

vec.setPathToFile(path);

cout << "Путь успешно изменен." << endl;

break;

default:

cout << "Введена неверная команда. Попробуйте еще раз." << endl;

break;

}

cout << endl << "Введите команду: ";

cin >> input;

}

return 0;

}

Файл ElectricalMachine.h:

#include <iostream>

#include <string>

#include <cstddef>

using namespace std;

#pragma once

class ElectricalMachine {

private:

string name; // название машины

double wattage = 0; // мощность

double voltage = 0; // напряжение

double amperage = 0; // сила тока

protected:

string className = "ElectricalMachine";

public:

ElectricalMachine();

ElectricalMachine(string name, double wattage, double voltage, double amperage);

ElectricalMachine(const ElectricalMachine& e);

virtual void printData() const; //вывод данных на экран

string getClassName() const;

string getName() const;

double getWattage() const;

double getVoltage() const;

double getAmperage() const;

};

Файл ElectricalMachine.cpp:

#include <iostream>

#include "ElectricalMachine.h"

using namespace std;

#pragma warning(disable:4996)

ElectricalMachine::ElectricalMachine() {

wattage = 0;

voltage = 0;

amperage = 0;

}

ElectricalMachine::ElectricalMachine(std::string name, double wattage, double voltage, double amperage) {

this->name = name;

this->wattage = wattage;

this->voltage = voltage;

this->amperage = amperage;

}

ElectricalMachine::ElectricalMachine(const ElectricalMachine& e) {

this->name = e.name;

this->wattage = e.wattage;

this->voltage = e.voltage;

this->amperage = e.amperage;

}

void ElectricalMachine::printData() const {

cout << "Machine name: " << name << endl;

cout << "Wattage: " << wattage << " kW" << endl;

cout << "Voltage: " << voltage << " V" << endl;

cout << "Amperage: " << amperage << " A" << endl;

}

string ElectricalMachine::getClassName() const {

return className;

}

string ElectricalMachine::getName() const {

return name;

}

double ElectricalMachine::getWattage() const

{

return wattage;

}

double ElectricalMachine::getVoltage() const {

return voltage;

}

double ElectricalMachine::getAmperage() const {

return amperage;

}

Файл Generator.h:

#include "ElectricalMachine.h"

#pragma once

class Generator : public ElectricalMachine {

private:

double fuelTankCapacity = 0;

public:

Generator(string name, double wattage, double voltage, double amperage, double fuelTankCapacity);

Generator(const Generator& g);

void printData() const;

double getFuelTankCapacity() const;

};

Файл Generator.cpp:

#include <iostream>

#include "ElectricalMachine.h"

#include "Generator.h"

Generator::Generator(std::string name, double wattage, double voltage, double amperage, double fuelTankCapacity)

: ElectricalMachine(name, wattage, voltage, amperage), fuelTankCapacity(fuelTankCapacity) {

className = "Generator";

}

Generator::Generator(const Generator& g) : ElectricalMachine(g), fuelTankCapacity(g.fuelTankCapacity) {

className = "Generator";

}

void Generator::printData() const {

ElectricalMachine::printData();

std::cout << "Generators's power: " << fuelTankCapacity << std::endl;

}

double Generator::getFuelTankCapacity() const {

return fuelTankCapacity;

}

Файл DieselGenerator.h:

#include "Generator.h"

#pragma once

class DieselGenerator : public virtual Generator {

private:

double fuelConsumption = 0;

public:

DieselGenerator(string name, double wattage, double voltage, double amperage, double fuelTankCapacity, double fuelConsumption);

DieselGenerator(const DieselGenerator& dg);

void printData() const;

double getFuelConsumption() const;

};

Файл DieselGenerator.cpp:

#include "DieselGenerator.h"

#include "Generator.h"

#include <iostream>

DieselGenerator::DieselGenerator(std::string name,

double wattage,

double voltage, double amperage,

double fuelTankCapacity,

double fuelConsumption)

: Generator(name, wattage, voltage, amperage, fuelTankCapacity), fuelConsumption(fuelConsumption) {

className = "DieselGenerator";

}

DieselGenerator::DieselGenerator(const DieselGenerator& dg) : Generator(dg), fuelConsumption(dg.fuelConsumption) {

className = "DieselGenerator";

}

void DieselGenerator::printData() const {

Generator::printData();

cout << "Fuel consumption: " << fuelConsumption << endl;

}

double DieselGenerator::getFuelConsumption() const {

return fuelConsumption;

}

Файл GasGenerator.h:

#include "Generator.h"

#pragma once

class GasGenerator : public virtual Generator {

private:

bool possibilityOfAutorun = false;

public:

GasGenerator(string name, double wattage, double voltage, double amperage, double fuelTankCapacity, bool possibilityOfAutorun);

GasGenerator(const GasGenerator& gg);

void printPossibilityOfAutorunField() const;

void printData() const;

bool getPossibilityOfAutorun() const;

};

Файл GasGenerator.cpp:

#include <iostream>

#include "GasGenerator.h"

GasGenerator::GasGenerator(std::string name,

   double wattage,

   double voltage,

   double amperage,

   double fuelTankCapacity,

   bool possibilityOfAutorun)

: Generator(name, wattage, voltage, amperage, fuelTankCapacity), possibilityOfAutorun(possibilityOfAutorun) {

className = "GasGenerator";

}

GasGenerator::GasGenerator(const GasGenerator& gg) : Generator(gg), possibilityOfAutorun(possibilityOfAutorun) {

className = "GasGenerator";

}

void GasGenerator::printPossibilityOfAutorunField() const {

std::cout << "Possibility of autorun: ";

possibilityOfAutorun == true

? std::cout << "yes" << std::endl

: std::cout << "no" << std::endl;

}

void GasGenerator::printData() const {

Generator::printData();

std::cout << "Possibility of autorun: ";

possibilityOfAutorun ? std::cout << "yes" << std::endl : std::cout << "no" << std::endl;

}

bool GasGenerator::getPossibilityOfAutorun() const {

return possibilityOfAutorun;

}

Файл MobileGenerator.h:

#include "DieselGenerator.h"

#include "GasGenerator.h"

#pragma once

class MobileGenerator : public DieselGenerator, public GasGenerator {

private:

double soundproofingMaterialThickness = 0;

public:

MobileGenerator(string name,

double wattage,

double voltage,

double amperage,

double fuelTankCapacity,

double fuelConsumption,

bool possibilityOfAutorun,

double soundproofingMaterialThickness);

MobileGenerator(MobileGenerator& mg);

void printData() const;

double getSoundproofingMaterialThickness() const;

};

Файл MobileGenerator.cpp:

#include "MobileGenerator.h"

#include <iostream>

MobileGenerator::MobileGenerator(string name, double wattage, double voltage, double amperage, double fuelTankCapacity,

double fuelConsumption, bool possibilityOfAutorun, double soundproofingMaterialThickness)

: DieselGenerator(name, wattage, voltage, amperage, fuelTankCapacity, fuelConsumption),

GasGenerator(name, wattage, voltage, amperage, fuelTankCapacity, possibilityOfAutorun),

Generator(name, wattage, voltage, amperage, fuelTankCapacity),

soundproofingMaterialThickness(soundproofingMaterialThickness) {

className = "MobileGenerator";

}

MobileGenerator::MobileGenerator(MobileGenerator& mg) : DieselGenerator(mg), GasGenerator(mg), Generator(mg),

soundproofingMaterialThickness(mg.soundproofingMaterialThickness) {

className = "MobileGenerator";

}

void MobileGenerator::printData() const

{

DieselGenerator::printData();

GasGenerator::printPossibilityOfAutorunField();

std::cout << "Soundproofing material tickness: " << soundproofingMaterialThickness << std::endl;

}

double MobileGenerator::getSoundproofingMaterialThickness() const {

return soundproofingMaterialThickness;

}

Файл MyVector.h:

#include "ElectricalMachine.h"

#include "Generator.h"

#include "DieselGenerator.h"

#include "GasGenerator.h"

#include "MobileGenerator.h"

#include <cstddef>

#pragma once

class MyVector {

private:

struct Node { //структура, в которой хранятся все данные об элементе вектора

ElectricalMachine\* data; // данные

Node\* next = NULL, \* prev = NULL; // указатель на следующий и предыдущий элемент вектора

};

Node\* begin, \* end;

string pathToFile;

int count;

public:

MyVector();

MyVector(string pathToFile);

MyVector(const MyVector& v);

~MyVector();

bool isEmpty() const; // проверка вектора на пустоту

int size(); // возвращает размер вектора

void add(ElectricalMachine\* d);

void removeLast(); // удаление

void print() const; // вывод на экран элемента на указанной позиции

void clear();

ElectricalMachine\*& getData(int pos) const; // возвращает элемент на заданной позиции

void vfind(string name) const;

void sortByVoltage();

void setPathToFile(string path);

void readFromFile();

void saveToFile();

};

Файл MyVector.cpp:

#include <iostream>

#include <fstream>

#include "MyVector.h"

using namespace std;

MyVector::MyVector() {

begin = NULL; end = NULL;

count = 0;

}

MyVector::MyVector(const MyVector& v) {

begin = end = NULL;

count = 0;

Node\* tmp = v.begin;

while (tmp != NULL) {

add(tmp->data);

tmp = tmp->next;

}

}

MyVector::MyVector(string pathToFile) : MyVector::MyVector() {

setPathToFile(pathToFile);

}

MyVector::~MyVector() {

if (!isEmpty()) {

Node\* tmp = NULL;

while (tmp != NULL) {

tmp = end;

end = end->prev;

delete(tmp);

}

}

}

bool MyVector::isEmpty() const {

return begin == NULL;

}

int MyVector::size() {

return count;

}

void MyVector::add(ElectricalMachine\* d) {

if (isEmpty()) {

begin = new Node;

begin->data = d;

end = begin;

}

else {

end->next = new Node;

end->next->prev = end;

end = end->next;

end->data = d;

end->next = NULL;

}

count++;

}

void MyVector::removeLast() {

if (!isEmpty()) {

Node\* tmp = end;

end = end->prev;

delete(tmp);

count--;

if (count == 0) {

begin = NULL;

}

}

else {

throw exception("Vector is empty");

}

}

void MyVector::print() const {

if (isEmpty()) {

cout << "Вектор пуст." << endl;

return;

}

Node\* tmp = begin;

int currentPos = 0;

while (currentPos < count) {

cout << "\tЭлемент " << currentPos << ":" << endl;

tmp->data->printData();

tmp = tmp->next;

currentPos++;

}

}

void MyVector::clear() {

while (!isEmpty()) removeLast();

}

ElectricalMachine\*& MyVector::getData(int pos) const{

if (isEmpty())

throw exception("Vector is empty");

if (pos > count)

throw exception("Position is bigger than count of elemens");

if (pos < 0)

throw exception("Invalid position");

Node\* tmp = begin;

int currentPos = 0;

while (tmp != NULL && currentPos < pos) {

tmp = tmp->next;

currentPos++;

}

return tmp->data;

}

void MyVector::sortByVoltage() {

if (isEmpty())

return;

for (int i = 0; i < count; i++) {

for (int j = count - 1; j > i; j--) {

ElectricalMachine\*& tmp1 = getData(j);

ElectricalMachine\*& tmp2 = getData(j - 1);

if (tmp1->getVoltage() < tmp2->getVoltage()) {

swap(tmp1, tmp2);

}

}

}

}

void MyVector::vfind(string name) const {

if (isEmpty())

throw exception("Vector is empty");

Node\* tmp = begin;

while (tmp != NULL) {

if (name == tmp->data->getName()) tmp->data->printData();

tmp = tmp->next;

}

}

void MyVector::setPathToFile(string path) {

pathToFile = path;

}

void MyVector::readFromFile() {

ifstream file(pathToFile);

if (!file.is\_open()) throw exception("Error opening file");

while (!file.eof()) {

string className;

string name;

double wattage;

double voltage;

double amperage;

file >> className;

if (className == "ElectricalMachine") {

file >> name >> wattage >> voltage >> amperage;

add(new ElectricalMachine(name, wattage, voltage, amperage));

}

else if (className == "Generator") {

double fuelTankCapacity;

file >> name >> wattage >> voltage >> amperage >> fuelTankCapacity;

add(new Generator(name, wattage, voltage, amperage, fuelTankCapacity));

}

else if (className == "DieselGenerator") {

double fuelTankCapacity;

double fuelConsumption;

file >> name >> wattage >> voltage >> amperage >> fuelTankCapacity >> fuelConsumption;

add(new DieselGenerator(name, wattage, voltage, amperage, fuelTankCapacity, fuelConsumption));

}

else if (className == "GasGenerator") {

double fuelTankCapacity;

bool possibilityOfAutorun;

string possibilityOfAutorunString;

file >> name >> wattage >> voltage >> amperage >> fuelTankCapacity >> possibilityOfAutorunString;

possibilityOfAutorun = possibilityOfAutorunString.\_Equal("yes")

? true

: false;

add(new GasGenerator(name, wattage, voltage, amperage, fuelTankCapacity, possibilityOfAutorun));

}

else if (className == "MobileGenerator") {

double fuelTankCapacity;

bool possibilityOfAutorun;

double fuelConsumption;

string possibilityOfAutorunString;

double soundproofingMaterialThickness;

file >> name >> wattage >> voltage >> amperage >> fuelTankCapacity >> fuelConsumption >> soundproofingMaterialThickness >> possibilityOfAutorunString;

possibilityOfAutorun = possibilityOfAutorunString.\_Equal("yes")

? true

: false;

add(new MobileGenerator(name,

wattage,

voltage,

amperage,

fuelTankCapacity,

fuelConsumption,

possibilityOfAutorun,

soundproofingMaterialThickness)

);

}

else if (className == "") continue;

else throw exception("Error reading from file: invalid class specified");

}

file.close();

}

void MyVector::saveToFile() {

ofstream file(pathToFile, std::ofstream::out | std::ofstream::trunc);

Node\* tmp = begin;

while (tmp != NULL) {

string className = tmp->data->getClassName();

file << className << " ";

file << tmp->data->getName() << " ";

file << tmp->data->getWattage() << " ";

file << tmp->data->getVoltage() << " ";

file << tmp->data->getAmperage();

if (className == "Generator") {

Generator\* g = dynamic\_cast<Generator\*>(tmp->data);

file << " " << g->getFuelTankCapacity() << endl;

}

else if (className == "DieselGenerator")

{

DieselGenerator\* dg = dynamic\_cast<DieselGenerator\*>(tmp->data);

file << " " << dg->getFuelTankCapacity() << " ";

file << dg->getFuelConsumption() << endl;

}

else if (className == "GasGenerator") {

GasGenerator\* gg = dynamic\_cast<GasGenerator\*>(tmp->data);

file << " " << gg->getFuelTankCapacity() << " ";

if (gg->getPossibilityOfAutorun()) file << "yes" << endl;

else file << "no" << endl;

}

else if (className == "MobileGenerator") {

MobileGenerator\* mg = dynamic\_cast<MobileGenerator\*>(tmp->data);

file << " " << mg->getFuelTankCapacity() << " ";

file << mg->getFuelConsumption() << " ";

file << mg->getSoundproofingMaterialThickness() << " ";

if (mg->getPossibilityOfAutorun()) file << "yes" << endl;

else file << "no" << endl;

}

else file << endl;

tmp = tmp->next;

}

}