МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт информационных технологий

Кафедра математического и программного обеспечения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

на тему «Объектно-ориентированное программирование на языке С ++»

Выполнил студент группы

  1ПИб-01-31оп

*группа*

направления подготовки

  09.03.04 Программная инженерия

*шифр, наименование*

  Чистяков Антон Андреевич

*фамилия, имя, отчество*

Руководитель

  Ершов Евгений Валентинович

*фамилия, имя, отчество*

  директор института ИИТ

*должность*

Дата представления работы

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Заключение о допуске к защите

Оценка \_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*количество баллов*

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Череповец, 2020

Содержание

[Введение 3](#_Toc30370647)

[1. Объектно-ориентированный анализ предметной области. 5](#_Toc30370648)

[2. Проектирование классов. 6](#_Toc30370649)

[3. Логическая структура программы. 9](#_Toc30370650)

[4. Модульная структура программы. 11](#_Toc30370651)

[5. Тестирование программы. 13](#_Toc30370652)

[Заключение 19](#_Toc30370653)

[Приложение 1. Техническое задание 21](#_Toc30370654)

[Приложение 2. Текст программы 25](#_Toc30370655)

[Приложение 3. Руководство пользователя 45](#_Toc30370656)

Введение

Объектно-ориентированное программирование (ООП) позволяет разложить проблему на составные части, каждая из которых становится самостоятельным объектом. Каждый из объектов содержит свой собственный код и данные, которые относятся к этому объекту.

Любая программа, написанная на языке ООП, отражает в своих данных состояние физических предметов либо абстрактных понятий – объектов программирования, для работы, с которыми она предназначена.

Все данные об объекте программирования и его связях с другими объектами можно объединить в одну структурированную переменную. В первом приближении ее можно назвать объектом.

С объектом связывается набор действий, иначе называемых методами. С точки зрения языка программирования набор действий или методов – это функции, получающие в качестве обязательного параметра указатель на объект и выполняющие определенные действия с данными объекта программирования. Технология ООП запрещает работать с объектом иначе, чем через методы, таким образом, внутренняя структура объекта скрыта от внешнего пользователя.

Описание множества однотипных объектов называется классом.

Объект – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

Класс – это описание множества объектов программирования (объектов) и выполняемых над ними действий.

Класс можно сравнить с чертежом, согласно которому создаются объекты. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области решаемой задачи.

Основными понятиями ООП являются

* инкапсуляция;
* наследование;
* полиморфизм.

Инкапсуляция данных (от «капсула») – это механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В ООП код и данные могут быть объединены вместе (в так называемый «черный ящик») при создании объекта [1].

Внутри объекта коды и данные могут быть закрытыми или открытыми.

Закрытые коды или данные доступны только для других частей того же самого объекта и, соответственно, недоступны для тех частей программы, которые существуют вне объекта.

Открытые коды и данные, напротив, доступны для всех частей программы, в том числе и для других частей того же самого объекта.

Наследование. Новый, или производный класс может быть определен на основе уже имеющегося, или базового класса.

При этом новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, а методы базового класса могут быть вызваны для объекта производного класса, причем они будут выполняться над данными включенного в него объекта базового класса.

Если объект наследует свои свойства от одного родителя, то говорят об одиночном наследовании. Если объект наследует данные и методы от нескольких базовых классов, то говорят о множественном наследовании.

Полиморфизм – это свойство, которое позволяет один и тот же идентификатор (одно и то же имя) использовать для решения двух и более схожих, но технически разных задач [1].

Целью полиморфизма, применительно к ООП, является использование одного имени для задания действий, общих для ряда классов объектов. Такой полиморфизм основывается на возможности включения в данные объекта также и информации о методах их обработки (в виде указателей на функции).

Полиморфная функция – это семейство функций с одним и тем же именем, но выполняющие различные действия в зависимости от условий вызова.

1. Объектно-ориентированный анализ предметной области.

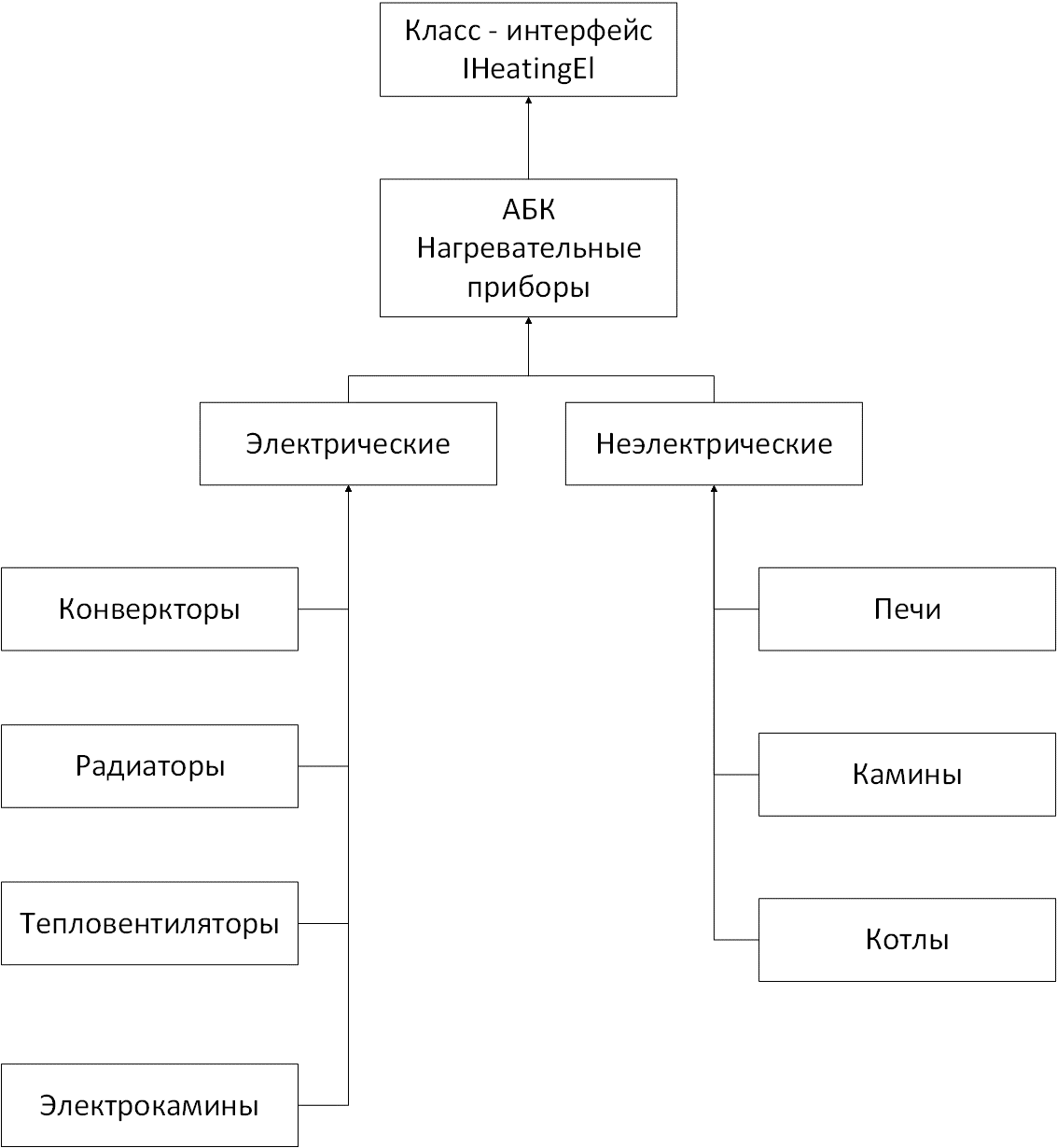
Предметной областью для разработки иерархии родственных классов являются нагревательные приборы. В общем случае можно различить два типа нагревательных приборов: электрические и неэлектрические, от которых уже можно вывести другие типы. В ходе анализа предметной области и декомпозиции объекта исследования была составлена иерархия родственных классов (рис. 1).

Рис. 1. Контекстная диаграмма классов.

1. Проектирование классов.

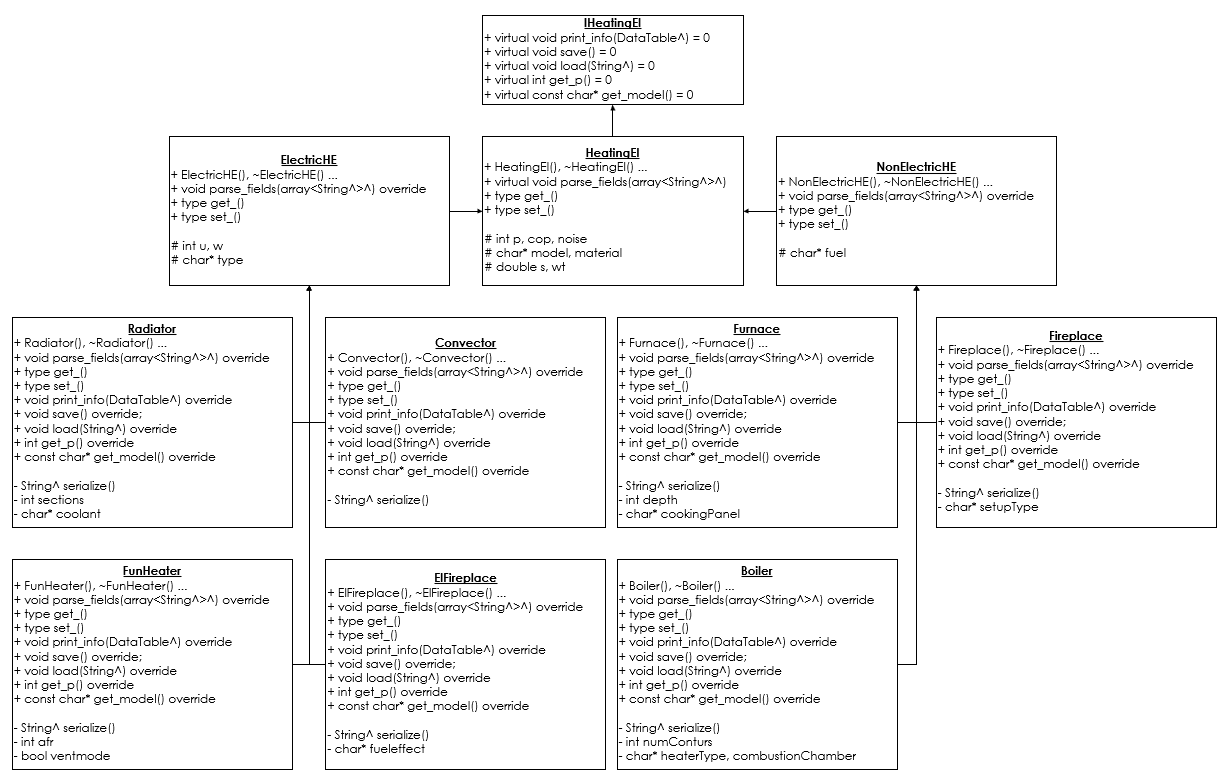
Исходя из составленной иерархии родственных классов была спроектирована детальная диаграмма классов, демонстрирующая использованные типы данных и реализованные методы (рис. 2).

Рис. 2. Детальная диаграмма классов.

Также в детальную диаграмму классов включены два класса, реализующих контейнер для хранения указателей на класс интерфейс (рис. 3).



Рис. 3. Диаграмма классов контейнера.

Класс интерфейс IHeatingEl:

* print\_info(DataTable^) – чисто виртуальный метод, служит для реализации вывода информации об объекте в таблицу;
* save() – чисто виртуальный метод, служит для сохранения информации об объекте в файл;
* load(String^) – чисто виртуальный метод, служит для загрузки информации из файла в объект;
* get\_p – чисто виртуальный метод, служит для получения значения тепловой мощности объекта;
* get\_model() – чисто виртуальный метод, служит для получения значения модели объекта.

Абстрактный базовый класс HeatingEl:

* конструкторы, дестркутор;
* parse\_fileds(array<String^>^) – виртуальный метод, служит для преобразования строковых значений, в значения типов полей;
* поля: тепловая мощность, уровень шума, КПД, модель, материал корпуса, рекомендуемая площадь помещения, вес.

Базовый класс ElectricHE:

* конструкторы, дестркутор;
* сеттеры, геттеры;
* parse\_fileds(array<String^>^) – виртуальный метод, служит для преобразования строковых значений, в значения типов полей;
* поля: рабочее напряжение, потребляемая мощность, тип нагревателя.

Потомки ElectricHE – Convector, Radiator, FunHeater, ElFireplace:

* конструкторы, дестркутор;
* сеттеры, геттеры;
* parse\_fileds(array<String^>^) – виртуальный метод, служит для преобразования строковых значений, в значения типов полей;
* переопределенные методы load, save, print\_info, get\_p, get\_model;
* метод serialize() – метод, позволяющий представить информацию об объекте виде строки;
* уникальные поля для каждого типа.

Базовый класс NonElectricHe:

* конструкторы, дестркутор;
* сеттеры, геттеры;
* parse\_fileds(array<String^>^) – виртуальный метод, служит для преобразования строковых значений, в значения типов полей;
* поле тип топлива.

Потомки NonElectricHE – Furnace, Fireplace, Boiler:

* конструкторы, дестркутор;
* сеттеры, геттеры;
* parse\_fileds(array<String^>^) – виртуальный метод, служит для преобразования строковых значений, в значения типов полей;
* переопределенные методы load, save, print\_info, get\_p, get\_model;
* метод serialize() – метод, позволяющий представить информацию об объекте виде строки;
* уникальные поля для каждого типа.

1. Логическая структура программы.

Логическая структура программы представлена на рис. 4.

Рис. 4. Логическая структура программы.

Описание работы:

1. Загрузка программы, запуск и инициализация компонентов, применение стартовой конфигурации: устанавливаются первоначальные значения глобальных переменных, контейнеров для хранения объектов и контейнера-шаблона для хранения указателей на класс интерфейс.
2. Ожидание действий пользователя: т.к. программа предполагает графический интерфейс, под ожиданием действий подразумевается навигация по программе, нажатие кнопок, открытие списков.
3. Выполнение обработчика события: после нажатие пользователем на какой-либо интерактивный элемент интерфейса, происходит запуск обработчика события, привязанного к этому элементу.
4. Состояние контейнеров: входными данными для обработчика события является текущее состояние контейнеров.
5. Обработка контейнеров: полиморфная обработка объектов контейнеров.
6. После выполнения обработчика события пользователю будет выведен результат.
7. Модульная структура программы.

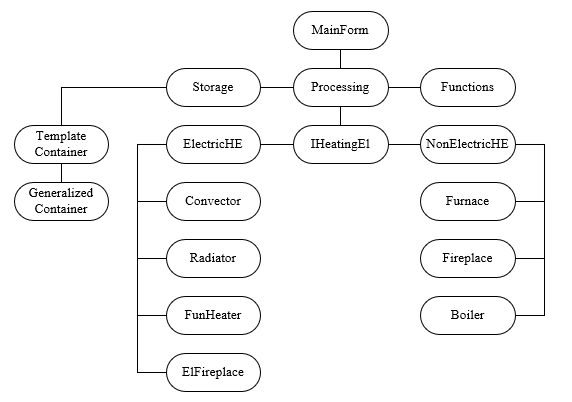
Модульная структура программы представлена на рис. 5.

Рис. 5. Модульная структура программы.

Спецификации модулей.

1. MainForm. Включает в себя модуль Processing. В данном модуле представлены компоненты GUI и функции обработчиков событий формы.
2. Processing. Включает в себя два модуля Storage и Functions. В данном модуле представлены функции полиморфной обработки родственных объектов.
3. Storage. Включает в себя модуль TemplateContainer. В данном модуле представлена реализация контейнеров для хранения объектов производных классов.
4. TemplateContainer. Включает в себя модуль GeneralizedContainer. В данном модуле представлена реализация контейнера-шаблона для хранения указателей на класс интерфейс.
5. GeneralizedContainer. В данном модуле представлена реализация обобщённого контейнерного класса.
6. Functions. В данном модуле представлены вспомогательные внешние функции, такие как конвертация строк.
7. IHeatingEl. Включает в себя два модуля ElectricHE и NonElectricHE. В данном модуле реализован класс интерфейс с чисто виртуальными методами.
8. ElectricHE и NonElectricHE. Включают в себя модули Convector, Radiator, FunHeater, ElFireplace, Furnace, Fireplace, Boiler. В данных модулях реализованы абстрактные базовые классы (второй уровень иерархии).
9. Convector, Radiator, FunHeater, ElFireplace, Furnace, Fireplace, Boiler – модули реализующие нижний уровень иерархии родственных классов – классы, от которых будут создаваться объекты.
10. Тестирование программы.
11. Тестирование классов.

Тестирование класса Convector (табл. 1).

Таблица 1. Тестирование класса Convector.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 21.12.2019 | Создание объекта класса | Нет | Успех |
| 21.12.2019 | print\_info() | Storage::convectors | Успех |
| 21.12.2019 | save() | Storage::convectors | Ошибка. Исправлена функция serialize(). |
| 21.12.2019 | load() | convectors.txt | Успех |
| 21.12.2019 | get(), set() | this | Успех |

Тестирование класса Radiator (табл. 2).

Таблица 2. Тестирование класса Radiator.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 22.12.2019 | Создание объекта класса | Нет | Успех |
| 22.12.2019 | print\_info() | Storage::radiators | Успех |
| 22.12.2019 | save() | Storage::radiators | Успех |
| 22.12.2019 | load() | radiators.txt | Успех |
| 22.12.2019 | get(), set() | this | Успех |

Тестирование класса FunHeater (табл. 3).

Таблица 3. Тестирование класса FunHeater.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 23.12.2019 | Создание объекта класса | Нет | Успех |
| 23.12.2019 | print\_info() | Storage::funheater | Успех |
| 23.12.2019 | save() | Storage::funheater | Успех |
| 23.12.2019 | load() | funheater.txt | Ошибка. Исправлена функция parse\_fileds(). |
| 23.12.2019 | get(), set() | this | Успех |

Тестирование класса ElFireplace (табл. 4).

Таблица 4. Тестирование класса ElFireplace.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 24.12.2019 | Создание объекта класса | Нет | Ошибка. Исправлено выделение памяти в конструкторе. |
| 24.12.2019 | print\_info() | Storage::elfires | Успех |
| 24.12.2019 | save() | Storage::elfires | Успех |
| 24.12.2019 | load() | elfires.txt | Успех |
| 24.12.2019 | get(), set() | this | Успех |

Тестирование класса Furnace (табл. 5).

Таблица 5. Тестирование класса Furnace.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 25.12.2019 | Создание объекта класса | Нет | Успех |
| 25.12.2019 | print\_info() | Storage::furnaces | Успех |
| 25.12.2019 | save() | Storage::furnaces | Успех |
| 25.12.2019 | load() | furnaces.txt | Успех |
| 25.12.2019 | get(), set() | this | Успех |

Тестирование класса Fireplace (табл. 6).

Таблица 6. Тестирование класса Fireplace.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 25.12.2019 | Создание объекта класса | Нет | Успех |
| 25.12.2019 | print\_info() | Storage::fireplaces | Ошибка. Исправлено представление таблицв. |
| 25.12.2019 | save() | Storage::fireplaces | Успех |
| 25.12.2019 | load() | fireplaces.txt | Ошибка. Исправлен некорректный вызов parse\_fileds(). |
| 25.12.2019 | get(), set() | this | Успех |

Тестирование класса Boiler (табл. 7).

Таблица 7. Тестирование класса Boiler.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 26.12.2019 | Создание объекта класса | Нет | Успех |
| 26.12.2019 | print\_info() | Storage::boilers | Успех |
| 26.12.2019 | save() | Storage::boilers | Успех |
| 26.12.2019 | load() | boilers.txt | Успех |
| 26.12.2019 | get(), set() | this | Успех |

Тестирование класса GeneralizedContainer (табл. 8).

Таблица 8. Тестирование класса GeneralizedContainer.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 26.12.2019 | Создание объекта класса | Нет | Успех |
| 26.12.2019 | Методы push\_front(), push\_back() | IHeatingEl\* | Успех |
| 26.12.2019 | Методы pop\_front(), pop\_back() | Нет | Ошибка. Исправлена утечка памяти использованием make\_share(). |
| 26.12.2019 | Методы front(), back() | Нет | Успех |

Тестирование класса TemplateContainer (табл. 9).

Таблица 9. Тестирование класса TemplateContainer.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 27.12.2019 | Создание объекта класса | Нет | Успех |
| 27.12.2019 | Методы push\_front(), push\_back() | Шаблонный тип Т | Успех |
| 27.12.2019 | Методы pop\_front(), pop\_back() | Нет | Успех |
| 27.12.2019 | Методы front(), back() | Нет | Ошибка. Замена инструкции static\_cast<> на reinterpret\_cast<>. |

1. Тестирование внешних функций.

Тестирование внешних функций представлено в табл. 10.

Таблица 10. Тестирование внешних функций.

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Тестируемая функция | Входные данные | Результат работы |
| 28.12.2019 | CharToSysString() | char\* | Успех |
| 28.12.2019 | SysStringToChar() | String^ | Успех |
| 28.12.2019 | PrintInfoObjects() | T, DataTable^ | Успех |
| 28.12.2019 | UploadObjects() | T | Ошибка. Заменен режим записи в файл. |

Продолжение таблицы 10.

|  |  |  |  |
| --- | --- | --- | --- |
| 28.12.2019 | GetAllModels() | Т | Успех |
| 28.12.2019 | GetSummP() | Т | Успех |
| 28.12.2019 | DownloadObjects() | Файлы с данными | Ошибка. Использован управляемый класс StreamReader. |

1. Тестирование требований ТЗ

Требования к функциональным характеристикам:

1. Была разработана и протестирована иерархия родственных классов, реализованы и протестированы функции обработки данных.
2. Реализован и протестирован файловый ввод/вывод.
3. Обработчики исключительных ситуаций отрабатывают при вводе ошибочных данных, неполном заполнении, ошибках сохранения или загрузки данных.

Требования к надежности:

1. При возникновении исключительной ситуации программа выведет пользователю информационное окно с содержанием ошибки, при этом не вызывая критических ошибок и аварийного завершения программы. Состояние загруженные в память объектов будет сохранено.

Заключение

В результате выполнения данной курсовой работы были получены:

Знания:

* основы технологии объектно-ориентированного программирования в
* объеме программы;
* особенности построения объектно-ориентированных систем;
* основные инструментальные средства для программирования систем;
* возможности объектно-ориентированного языка;
* процесс проектирования и создания компьютерной программы;
* стандартные библиотеки для построения объектно-ориентированных
* программ;
* средства реализации принципов ООП.

Умения:

* разрабатывать программный код с использованием принципов ООП;
* проектировать и разрабатывать локальные приложения;
* использовать инструментальные средства для создания систем;
* использовать стандартные библиотеки при программировании системы.

Владения:

* навыками программирования с использованием ООП;
* навыками работы в инструментальной среде разработки программного продукта;
* навыками построения объектно-ориентированной модели.

Список литературы

1. Пол А. «Объектно-ориентированное программирование на C++»:СПб, изд. «Невский диалект», 2001.

2. Павловская Т. А. «С/С++. Программирование на языке высокого уровня»: СПб, изд. «ПИТЕР», 2003.

3. Герберт Шилдт. «Полный справочник по С++»:СПб, изд. «Вильямс», 2007.

4. Указания к лабораторным работам по дисциплине «Структуры и алгоритмы обработки данных». Ч. 1: Учеб.-метод. пособие. Череповец: ЧГУ, 2002. 21 с.

5. Р. Лафоре "Объектно-Ориентированное Программирование в С++.": 4-е издание.; – М.:Питер, – 2004. – 902 с.

Приложение 1. Техническое задание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное

образовательное учреждение высшего образования

**«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт информационных технологий

Кафедра математического и программного обеспечения ЭВМ

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,

д. т.н., профессор Ершов Е.В.

« » 2020 г.

Конструирование модели лексического анализа Техническое задание на курсовую работу

Листов

Руководитель: директор института ИИТ, Ершов Е.В. Исполнитель: студент гр.

1ПИб-01-31оп

Чистяков Антон Андреевич

2020 г.

1. Введение

Наименование разрабатываемого программного продукта: «Объектно-ориентированное программирование на языке С++». Предметная область – нагревательные приборы.

С помощью механизмов объектно-ориентированного программирования, возможно, наиболее наглядно представлять различные типы данных. Каждый объект, описывающий информацию некоторой предметной области, обладает рядом свойств, для работы с которыми существуют специальные функции-методы. Вместе свойства и методы образуют класс.

2. Основания для разработки

Основанием для разработки данного программного продукта является задание на курсовую работу «Объектно-ориентированное программирование на языке С ++» выданное преподавателем кафедры ПИ ЧГУ в рамках учебного плана.

3. Назначение программного продукта

Основной задачей данного программного продукта является изучение языка программирования C++ в объектно-ориентированном контексте. Целью данной разработки является:

* создание объектов классов и обмен данными между ними;
* запись данных в файл;
* вывод данных и результатов на экран;
* создание классов и демонстрация механизмов работы с ними;
* реализация механизмов наследования;
* полиморфная обработка родственных объектов;
* обработка исключительных ситуаций.

4. Требования к программе

4.1. Требование к функциональным характеристикам

* Разработать иерархию родственных типов, корневой класс которой - абстрактный базовый класс (интерфейс), для моделирования и обработки данных предметной области «Нагревательные приборы»;
* реализовать файловый вывод, вывод данных на дисплей;
* исключительные ситуаций обрабатываются при сохранении объектов классов в файл, ввода данных с клавиатуры и загрузки объектов из файлов.

4.2. Требования к входным данным

* Файл не должен быть пустым.

4.3. Требования к надежности

В программе должна быть реализована обработка различных исключительных ситуаций и корректное ее завершение после возникновения ошибок.

4.4. Условия эксплуатации

Данный программный продукт предназначен для работы на одном персональном компьютере одним пользователем.

4.5. Требования к составу и параметрам технических средств

Для работы программы требуется IBM-совместимый персональный компьютер, ОС Windows 98/2000/XP/Vista/7, процессор Pentium и выше, минимум 64 Мб ОП и минимум 32 Мб свободного места на жестком диске.

4.6. Требования к информационной и программной совместимости

Разработка программы производится с использованием средств языка С++ в среде разработки Borland C++ Builder 6.0. Может эксплуатироваться в операционных системах Windows 98/2000/XP/Vista/7.

5. Требования к программной документации

Разработка данной программы предполагает написание следующей документации:

* расчетно-пояснительной записки;
* технического задания (Приложение 1);
* текст программы (Приложение 2);
* руководство пользователя (Приложение 3).

6. Стадии и этапы разработки

Выполнение работы будет осуществляться по следующим этапам:

|  |  |  |
| --- | --- | --- |
| Этап | Плановая дата | Фактическая дата |
| Объектно-ориентированный анализ предметной области |  |  |
| Разработка и реализация алгоритмов |  |  |
| Разработка интерфейса программы |  |  |
| Отладка и тестирование |  |  |
| Составление программной документации |  |  |

7. Порядок контроля и приемки

Контроль и прием работы будет происходить в следующем порядке:

* cдача технического задания и иерархии родственных типов;
* обработка исключительных ситуаций и тестирование программы;
* предоставление программы в электронном виде;
* сдача расчетно-пояснительной записки.

Приложение 2. Текст программы

MainForm.h

private: System::Void Electric\_R\_CheckedChanged(System::Object^ sender, System::EventArgs^ e)

{

if (Electric\_R->Checked) {

Electric\_GB->Visible = true;

}

else Electric\_GB->Visible = false;

}

private: System::Void NonElectric\_R\_CheckedChanged(System::Object^ sender, System::EventArgs^ e)

{

if (NonElectric\_R->Checked) {

NonElectirc\_GB->Visible = true;

}

else NonElectirc\_GB->Visible = false;

}

private: System::Void Add\_B\_Click(System::Object^ sender, System::EventArgs^ e)

{

ObjectsDGV->Visible = false;

Width = 990;

MainPropetry\_GB->Visible = true;

SpecialProperty\_GB->Visible = true;

PanelsClear();

if (Electric\_R->Checked) {

SP\_Electric\_P->Visible = true;

if (Convector\_R->Checked) {

SP\_Convector\_P->Visible = true;

}

else if (Radiator\_R->Checked) {

SP\_Radiator\_P->Visible = true;

}

else if (FunHeater\_R->Checked) {

SP\_FunHeater\_P->Visible = true;

}

else if (ElectricFirePlace\_R->Checked) {

SP\_ElectricFireplace\_P->Visible = true;

}

}

else if (NonElectric\_R->Checked) {

SP\_NonElectric\_P->Visible = true;

if (Furnace\_R->Checked) {

SP\_Furnace\_P->Visible = true;

}

else if (Fireplace\_R->Checked) {

SP\_Fireplace\_P->Visible = true;

}

else if (Boiler\_R->Checked) {

SP\_Boiler\_P->Visible = true;

}

}

}

private: Void AddVisibleFalse()

{

MainPropetry\_GB->Visible = false;

SpecialProperty\_GB->Visible = false;

PanelsClear();

}

private: Void PanelsClear()

{

ClearUserText(MainPropetry\_GB);

for each (Control ^ c in SpecialProperty\_GB->Controls){

if (c->GetType() == SP\_NonElectric\_P->GetType())

{

ClearUserText(c);

c->Visible = false;

}

}

}

private: Void ClearUserText(Control ^group)

{

for each (Control ^ c in group->Controls) {

if (c->GetType() != label1->GetType()) {

c->Text = "";

}

}

}

private: System::Void OpenList\_B\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

AddVisibleFalse();

ObjectsDGV->Visible = true;

Width = 1200;

int i = 0;

double d = 0;

data = gcnew DataTable();

data->Columns->Add("Модель");

data->Columns->Add("Материал");

data->Columns->Add("Тепловая мощность (вт)", i.GetType());

data->Columns->Add("КПД (%)", i.GetType());

data->Columns->Add("Рек. площадь (кв.м)", d.GetType());

data->Columns->Add("Уровень шума (дб)", i.GetType());

data->Columns->Add("Вес (кг)", d.GetType());

if (Electric\_R->Checked)

{

data->Columns->Add("Рабочее напряжение (В)", i.GetType());

data->Columns->Add("Потребляемая мощность (Вт)", i.GetType());

data->Columns->Add("Тип нагревательного элемента");

if (Convector\_R->Checked)

{

//

}

else if (Radiator\_R->Checked)

{

data->Columns->Add("Количество секций", i.GetType());

data->Columns->Add("Теплоноситель");

PrintInfoObjects(&Storage::radiators, data);

}

else if (FunHeater\_R->Checked)

{

data->Columns->Add("Расход воздуха", i.GetType());

data->Columns->Add("Режим вентиляции");

PrintInfoObjects(&Storage::funheaters, data);

}

else if (ElectricFirePlace\_R->Checked)

{

data->Columns->Add("Эффект топлива");

PrintInfoObjects(&Storage::electricfireplaces, data);

}

}

else if (NonElectric\_R->Checked)

{

data->Columns->Add("Топливо");

if (Furnace\_R->Checked)

{

data->Columns->Add("Глубина топки", d.GetType());

data->Columns->Add("Варочная панель");

PrintInfoObjects(&Storage::furnaces, data);

}

else if (Fireplace\_R->Checked)

{

data->Columns->Add("Тип установки");

PrintInfoObjects(&Storage::fireplaces, data);

}

else if (Boiler\_R->Checked)

{

data->Columns->Add("Количество контуров", i.GetType());

data->Columns->Add("Тип горелки");

data->Columns->Add("Камера сгорания");

PrintInfoObjects(&Storage::boilers, data);

}

}

ObjectsDGV->DataSource = nullptr;

ObjectsDGV->Rows->Clear();

ObjectsDGV->DataSource = data;

}

catch (const char\* ex)

{

MessageBox::Show(CharToSysString(ex), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (Exception^ ex)

{

MessageBox::Show(ex->Message, "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (...)

{

MessageBox::Show("Неизвестная ошибка.", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

private: System::Void MainForm\_Load(System::Object^ sender, System::EventArgs^ e)

{

Width = 295;

}

private: System::Void AddToMemory\_Btn\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

if (CheckFields(MainPropetry\_GB)) return;

if (Electric\_R->Checked)

{

if (CheckFields(SP\_Electric\_P)) return;

if (Convector\_R->Checked)

{

auto ptr = make\_shared<Convector>();

ptr->load(Model\_TB->Text + "|" + Material\_CB->Text + "|" + P\_TB->Text + "|" + COP\_TB->Text + "|" +

S\_TB->Text + "|" + NoiseLvl\_TB->Text + "|" + WT\_TB->Text + "|" +

U\_TB->Text + "|" + W\_TB->Text + "|" + HeatingEl\_CB->Text);

Storage::convectors.push\_back(\*ptr);

}

else if (Radiator\_R->Checked)

{

if (CheckFields(SP\_Radiator\_P)) return;

auto ptr = make\_shared<Radiator>();

ptr->load(Model\_TB->Text + "|" + Material\_CB->Text + "|" + P\_TB->Text + "|" + COP\_TB->Text + "|" +

S\_TB->Text + "|" + NoiseLvl\_TB->Text + "|" + WT\_TB->Text + "|" +

U\_TB->Text + "|" + W\_TB->Text + "|" + HeatingEl\_CB->Text + "|" +

Sections\_TB->Text + "|" + Coolant\_TB->Text);

Storage::radiators.push\_back(\*ptr);

}

else if (FunHeater\_R->Checked)

{

if (CheckFields(SP\_FunHeater\_P)) return;

auto ptr = make\_shared<FunHeater>();

ptr->load(Model\_TB->Text + "|" + Material\_CB->Text + "|" + P\_TB->Text + "|" + COP\_TB->Text + "|" +

S\_TB->Text + "|" + NoiseLvl\_TB->Text + "|" + WT\_TB->Text + "|" +

U\_TB->Text + "|" + W\_TB->Text + "|" + HeatingEl\_CB->Text + "|" +

AFR\_TB->Text + "|" + Ventmode\_CB->Text);

Storage::funheaters.push\_back(\*ptr);

}

else if (ElectricFirePlace\_R->Checked)

{

if (CheckFields(SP\_ElectricFireplace\_P)) return;

auto ptr = make\_shared<ElectricFireplace>();

ptr->load(Model\_TB->Text + "|" + Material\_CB->Text + "|" + P\_TB->Text + "|" + COP\_TB->Text + "|" +

S\_TB->Text + "|" + NoiseLvl\_TB->Text + "|" + WT\_TB->Text + "|" +

U\_TB->Text + "|" + W\_TB->Text + "|" + HeatingEl\_CB->Text + "|" +

FuelEffect\_CB->Text);

Storage::electricfireplaces.push\_back(\*ptr);

}

}

else if (NonElectric\_R->Checked)

{

if (CheckFields(SP\_NonElectric\_P)) return;

if (Furnace\_R->Checked)

{

if (CheckFields(SP\_Furnace\_P)) return;

auto ptr = make\_shared<Furnace>();

ptr->load(Model\_TB->Text + "|" + Material\_CB->Text + "|" + P\_TB->Text + "|" + COP\_TB->Text + "|" +

S\_TB->Text + "|" + NoiseLvl\_TB->Text + "|" + WT\_TB->Text + "|" +

FuelType\_CB->Text + "|" +

FurnaceDepth\_TB->Text + "|" + CookingPanel\_CB->Text);

Storage::furnaces.push\_back(\*ptr);

}

else if (Fireplace\_R->Checked)

{

if (CheckFields(SP\_Fireplace\_P)) return;

auto ptr = make\_shared<Fireplace>();

ptr->load(Model\_TB->Text + "|" + Material\_CB->Text + "|" + P\_TB->Text + "|" + COP\_TB->Text + "|" +

S\_TB->Text + "|" + NoiseLvl\_TB->Text + "|" + WT\_TB->Text + "|" +

FuelType\_CB->Text + "|" +

SetupMethod\_GB->Text);

Storage::fireplaces.push\_back(\*ptr);

}

else if (Boiler\_R->Checked)

{

if (CheckFields(SP\_Boiler\_P)) return;

auto ptr = make\_shared<Boiler>();

ptr->load(Model\_TB->Text + "|" + Material\_CB->Text + "|" + P\_TB->Text + "|" + COP\_TB->Text + "|" +

S\_TB->Text + "|" + NoiseLvl\_TB->Text + "|" + WT\_TB->Text + "|" +

FuelType\_CB->Text + "|" +

CircuitNum\_GB->Text + "|" + BurnerType\_GB->Text + "|" + CombustorType\_GB->Text);

Storage::boilers.push\_back(\*ptr);

}

}

}

catch (const char\* ex)

{

MessageBox::Show(CharToSysString(ex), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (Exception^ ex)

{

MessageBox::Show(ex->Message, "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (...)

{

MessageBox::Show("Неизвестная ошибка.", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

private: bool CheckFields(Control^ group)

{

for each (Control ^ c in group->Controls)

{

c->Name;

if (c->GetType() != label1->GetType())

{

if (c->Text == "")

{

MessageBox::Show("Заполните все поля!", "Внимание", MessageBoxButtons::OK, MessageBoxIcon::Warning);

return true;

}

}

}

return false;

}

private: System::Void ЗагрузитьОбъектыToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

DownloadObjects();

}

catch (const char\* ex)

{

MessageBox::Show(CharToSysString(ex), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (Exception^ ex)

{

MessageBox::Show(ex->Message, "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (...)

{

MessageBox::Show("Неизвестная ошибка.", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

private: System::Void ВыгрузитьОбъектыToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

UploadObjects(&Storage::convectors);

UploadObjects(&Storage::radiators);

UploadObjects(&Storage::funheaters);

UploadObjects(&Storage::electricfireplaces);

UploadObjects(&Storage::furnaces);

UploadObjects(&Storage::fireplaces);

UploadObjects(&Storage::boilers);

}

catch (const char\* ex)

{

MessageBox::Show(CharToSysString(ex), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (Exception^ ex)

{

MessageBox::Show(ex->Message, "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (...)

{

MessageBox::Show("Неизвестная ошибка.", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

private: System::Void ВывестиВсеМоделиToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

String^ str = "";

str += GetAllModels(&Storage::convectors);

str += GetAllModels(&Storage::radiators);

str += GetAllModels(&Storage::funheaters);

str += GetAllModels(&Storage::electricfireplaces);

str += GetAllModels(&Storage::furnaces);

str += GetAllModels(&Storage::fireplaces);

str += GetAllModels(&Storage::boilers);

MessageBox::Show(str, "Вывод", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

catch (const char\* ex)

{

MessageBox::Show(CharToSysString(ex), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (Exception^ ex)

{

MessageBox::Show(ex->Message, "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (...)

{

MessageBox::Show("Неизвестная ошибка.", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

private: System::Void ПодсчитатьСуммарнуюМощностьToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

try

{

int sum = 0;

sum += GetSummP(&Storage::convectors);

sum += GetSummP(&Storage::radiators);

sum += GetSummP(&Storage::funheaters);

sum += GetSummP(&Storage::electricfireplaces);

sum += GetSummP(&Storage::furnaces);

sum += GetSummP(&Storage::fireplaces);

sum += GetSummP(&Storage::boilers);

MessageBox::Show("Суммарная мощность всех нагрвательных приборов - " + sum.ToString() + " Вт.", "Вывод", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

catch (const char\* ex)

{

MessageBox::Show(CharToSysString(ex), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (Exception^ ex)

{

MessageBox::Show(ex->Message, "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (...)

{

MessageBox::Show("Неизвестная ошибка.", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

IHeatingEl.h

#pragma once

// Подключаем файл с внешними функциями

#include "Functions.h"

class IHeatingEl

{

public:

virtual int get\_p() = 0;

virtual const char\* get\_model() = 0;

virtual void print\_info(DataTable^ data) = 0;

virtual void save() = 0;

virtual void load(String^ serialized\_object) = 0;

};

HeatingEl.h

#pragma once

#include "IHeatingEl.h"

class HeatingEl : public IHeatingEl

{

public:

// SET

void set\_p(int value);

void set\_cop(int value);

void set\_noise(int value);

void set\_model(char\*);

void set\_material(char\*);

void set\_s(double);

void set\_wt(double);

// GET

int get\_cop();

int get\_noise();

const char\* get\_material();

double get\_s();

double get\_wt();

virtual void parse\_fields(cli::array<String^>^ fields);

HeatingEl();

HeatingEl(const HeatingEl&);

HeatingEl(int p, int cop, int noise, char\* model, char\* material, double s, double wt);

~HeatingEl();

protected:

int p; // Тепловая мощность

int cop; // КПД

int noise; // Уровень шума

char\* model; // Модель

char\* material; // Материал

double s; // Рекомендуемая площадь помещения

double wt; // Вес

};

HeatingEl.cpp

#include "HeatingEl.h"

HeatingEl::HeatingEl()

{

s = wt = p = cop = noise = 0;

model = nullptr;

material = nullptr;

}

HeatingEl::HeatingEl(const HeatingEl& ref)

{

s = ref.s; wt = ref.wt; p = ref.p; cop = ref.cop; noise = ref.noise;

model = nullptr; set\_model(ref.model);

material = nullptr; set\_material(ref.material);

}

HeatingEl::HeatingEl(int p, int cop, int noise, char\* model, char\* material, double s, double wt)

{

this->p = p; this->cop = cop; this->noise = noise; this->s = s; this->wt = wt;

this->model = nullptr; set\_model(model);

this->material = nullptr; set\_material(material);

}

HeatingEl::~HeatingEl()

{

delete[] model;

delete[] material;

}

void HeatingEl::set\_p(int value)

{

p = value;

}

void HeatingEl::set\_cop(int value)

{

cop = value;

}

void HeatingEl::set\_noise(int value)

{

noise = value;

}

void HeatingEl::set\_model(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (model)

{

delete[] model; model = nullptr;

}

model = new char[(strlen(value) + 1)];

strcpy(model, value);

}

void HeatingEl::set\_material(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (material)

{

delete[] material; material = nullptr;

}

material = new char[(strlen(value) + 1)];

strcpy(material, value);

}

void HeatingEl::set\_s(double value)

{

s = value;

}

void HeatingEl::set\_wt(double value)

{

wt = value;

}

int HeatingEl::get\_cop()

{

return cop;

}

int HeatingEl::get\_noise()

{

return noise;

}

const char\* HeatingEl::get\_material()

{

return material;

}

double HeatingEl::get\_s()

{

return s;

}

double HeatingEl::get\_wt()

{

return wt;

}

void HeatingEl::parse\_fields(cli::array<String^>^ fields)

{

if (!Int32::TryParse(fields[2], this->p)) throw "Неверное значение для поля p";

if (!Int32::TryParse(fields[3], this->cop)) throw "Неверное значение для поля cop";

if (!Int32::TryParse(fields[5], this->noise)) throw "Неверное значение для поля noise";

set\_model(SysStringToChar(fields[0]));

set\_material(SysStringToChar(fields[1]));

if (!Double::TryParse(fields[4], this->s)) throw "Неверное значение для поля s";

if (!Double::TryParse(fields[6], this->wt)) throw "Неверное значение для поля wt";

}

ElectricHE.h

#pragma once

#include "HeatingEl.h"

class ElectricHE : public HeatingEl

{

public:

ElectricHE();

ElectricHE(const ElectricHE&);

ElectricHE(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type);

~ElectricHE();

void set\_u(int value);

void set\_w(int value);

void set\_type(char\*);

int get\_u();

int get\_w();

const char\* get\_type();

void parse\_fields(cli::array<String^>^ fields) override;

protected:

int u; // Рабочее напряжения

int w; // Потребляемая мощность

char\* type; // Тип нагревательного элемента

};

ElectricHE.cpp

#include "ElectricHE.h"

ElectricHE::ElectricHE()

{

u = w = 0;

type = nullptr;

}

ElectricHE::ElectricHE(const ElectricHE& ref) : HeatingEl(ref)

{

u = ref.u; w = ref.w;

type = nullptr; set\_type(ref.type);

}

ElectricHE::ElectricHE(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type) : HeatingEl(p, cop, noise, model, material, s, wt)

{

this->u = u;

this->w = w;

this->type = nullptr; set\_type(type);

}

ElectricHE::~ElectricHE()

{

delete[] type;

}

void ElectricHE::set\_u(int value)

{

u = value;

}

void ElectricHE::set\_w(int value)

{

w = value;

}

void ElectricHE::set\_type(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (type)

{

delete[] type; type = nullptr;

}

type = new char[(strlen(value) + 1)];

strcpy(type, value);

}

int ElectricHE::get\_u()

{

return u;

}

int ElectricHE::get\_w()

{

return w;

}

const char\* ElectricHE::get\_type()

{

return type;

}

void ElectricHE::parse\_fields(cli::array<String^>^ fields)

{

HeatingEl::parse\_fields(fields);

if (!Int32::TryParse(fields[7], this->u)) throw "Неверное значение для поля u";

if (!Int32::TryParse(fields[8], this->w)) throw "Неверное значение для поля w";

set\_type(SysStringToChar(fields[9]));

}

NonElectricHE.h

#pragma once

#include "HeatingEl.h"

class NonElectricHE : public HeatingEl

{

public:

NonElectricHE();

NonElectricHE(const NonElectricHE&);

NonElectricHE(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel);

~NonElectricHE();

void set\_fuel(char\*);

const char\* get\_fuel();

void parse\_fields(cli::array<String^>^ fields) override;

protected:

char\* fuel; // Тип топлива

};

NonElectricHE.cpp

#include "No

nElectricHE.h"

NonElectricHE::NonElectricHE()

{

fuel = nullptr;

}

NonElectricHE::NonElectricHE(const NonElectricHE& ref) : HeatingEl(ref)

{

fuel = nullptr; set\_fuel(ref.fuel);

}

NonElectricHE::NonElectricHE(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel) : HeatingEl(p, cop, noise, model, material, s, wt)

{

this->fuel = nullptr; set\_fuel(fuel);

}

NonElectricHE::~NonElectricHE()

{

delete[] fuel;

}

void NonElectricHE::set\_fuel(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (fuel)

{

delete[] fuel; fuel = nullptr;

}

fuel = new char[(strlen(value) + 1)];

strcpy(fuel, value);

}

const char\* NonElectricHE::get\_fuel()

{

return fuel;

}

void NonElectricHE::parse\_fields(cli::array<String^>^ fields)

{

HeatingEl::parse\_fields(fields);

set\_fuel(SysStringToChar(fields[7]));

}

Convector.h

#pragma once

#include "ElectricHE.h"

class Convector : public ElectricHE

{

public:

Convector();

Convector(String^ serialized\_object);

Convector(const Convector&);

Convector(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type);

~Convector();

int get\_p() override;

const char\* get\_model() override;

void print\_info(DataTable^ data) override;

void save() override;

void load(String^ serialized\_object) override;

void parse\_fields(cli::array<String^>^ fields) override;

private:

String^ serialize();

};

Convector.cpp

#include "Convector.h"

Convector::Convector(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type) : ElectricHE(p, cop, noise, model, material, s, wt, u, w, type)

{

}

int Convector::get\_p()

{

return p;

}

const char\* Convector::get\_model()

{

return model;

}

void Convector::print\_info(DataTable^ data)

{

data->Rows->Add(serialize()->Split('|'));

}

void Convector::save()

{

StreamWriter^ writer = gcnew StreamWriter("convectors.txt", true);

writer->WriteLine(serialize()); writer->Close();

}

void Convector::load(String^ serialized\_object)

{

parse\_fields(serialized\_object->Split('|'));

}

void Convector::parse\_fields(cli::array<String^>^ fields)

{

ElectricHE::parse\_fields(fields);

}

String^ Convector::serialize()

{

return gcnew String(CharToSysString(model) + "|" + CharToSysString(material) + "|" + p.ToString() + "|" + cop.ToString() + "|" +

s.ToString() + "|" + noise.ToString() + "|" + wt.ToString() + "|" +

u.ToString() + "|" + w.ToString() + "|" + CharToSysString(type));

}

Radiator.h

#pragma once

#include "ElectricHE.h"

class Radiator : public ElectricHE

{

public:

Radiator();

Radiator(const Radiator&);

Radiator(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type, int sections, char\* coolant);

~Radiator();

void set\_sections(int value);

void set\_coolant(char\*);

int get\_sections();

const char\* get\_coolant();

int get\_p() override;

const char\* get\_model() override;

void print\_info(DataTable^ data) override;

void save() override;

void load(String^ serialized\_object) override;

void parse\_fields(cli::array<String^>^ fields) override;

private:

int sections; // Количество секций

char\* coolant; // Тип теплоносителя

String^ serialize();

};

Radiator.cpp

#include "Radiator.h"

Radiator::Radiator()

{

sections = 0;

coolant = nullptr;

}

Radiator::Radiator(const Radiator& ref) : ElectricHE(ref)

{

sections = ref.sections;

coolant = nullptr; set\_coolant(ref.coolant);

}

Radiator::Radiator(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type, int sections, char\* coolant) : ElectricHE(p, cop, noise, model, material, s, wt, u, w, type)

{

this->sections = sections;

this->coolant = nullptr; set\_coolant(coolant);

}

Radiator::~Radiator()

{

delete[] coolant;

}

void Radiator::set\_sections(int value)

{

sections = value;

}

void Radiator::set\_coolant(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (coolant)

{

delete[] coolant; coolant = nullptr;

}

coolant = new char[(strlen(value) + 1)];

strcpy(coolant, value);

}

int Radiator::get\_sections()

{

return sections;

}

const char\* Radiator::get\_coolant()

{

return coolant;

}

int Radiator::get\_p()

{

return p;

}

const char\* Radiator::get\_model()

{

return model;

}

void Radiator::print\_info(DataTable^ data)

{

String^ str = serialize();

cli::array<String^>^ ff = str->Split('|');

data->Rows->Add(serialize()->Split('|'));

}

void Radiator::save()

{

StreamWriter^ writer = gcnew StreamWriter("radiators.txt", true);

writer->WriteLine(serialize()); writer->Close();

}

void Radiator::load(String^ serialized\_object)

{

parse\_fields(serialized\_object->Split('|'));

}

void Radiator::parse\_fields(cli::array<String^>^ fields)

{

ElectricHE::parse\_fields(fields);

if (!Int32::TryParse(fields[10], this->sections)) throw "Неверное значение для поля sections";

set\_coolant(SysStringToChar(fields[11]));

}

String^ Radiator::serialize()

{

return gcnew String(CharToSysString(model) + "|" + CharToSysString(material) + "|" + p.ToString() + "|" + cop.ToString() + "|" +

s.ToString() + "|" + noise.ToString() + "|" + wt.ToString() + "|" +

u.ToString() + "|" + w.ToString() + "|" + CharToSysString(type) + "|" +

sections.ToString() + "|" + CharToSysString(coolant));

}

FunHeater.h

#pragma once

#include "ElectricHE.h"

class FunHeater : public ElectricHE

{

public:

FunHeater();

FunHeater(const FunHeater&);

FunHeater(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type, int afr, bool ventmode);

~FunHeater();

void set\_afr(int value);

void set\_ventmode(bool);

int get\_afr();

bool get\_ventmode();

int get\_p() override;

const char\* get\_model() override;

void print\_info(DataTable^ data) override;

void save() override;

void load(String^ serialized\_object) override;

void parse\_fields(cli::array<String^>^ fields) override;

private:

int afr; // Расход воздуха

bool ventmode; // Режим вентиляции без нагрева

String^ serialize();

};

FunHeater.cpp

#include "FunHeater.h"

FunHeater::FunHeater()

{

afr = ventmode = 0;

}

FunHeater::FunHeater(const FunHeater& ref) : ElectricHE(ref)

{

afr = ref.afr; ventmode = ref.ventmode;

}

FunHeater::FunHeater(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type, int afr, bool ventmode) : ElectricHE(p, cop, noise, model, material, s, wt, u, w, type)

{

this->afr = afr; this->ventmode = ventmode;

}

FunHeater::~FunHeater()

{

}

void FunHeater::set\_afr(int value)

{

afr = value;

}

void FunHeater::set\_ventmode(bool value)

{

ventmode = value;

}

int FunHeater::get\_afr()

{

return afr;

}

bool FunHeater::get\_ventmode()

{

return ventmode;

}

int FunHeater::get\_p()

{

return p;

}

const char\* FunHeater::get\_model()

{

return model;

}

void FunHeater::print\_info(DataTable^ data)

{

data->Rows->Add(serialize()->Split('|'));

}

void FunHeater::save()

{

StreamWriter^ writer = gcnew StreamWriter("funheaters.txt", true);

writer->WriteLine(serialize()); writer->Close();

}

void FunHeater::load(String^ serialized\_object)

{

parse\_fields(serialized\_object->Split('|'));

}

void FunHeater::parse\_fields(cli::array<String^>^ fields)

{

ElectricHE::parse\_fields(fields);

if (!Int32::TryParse(fields[10], this->afr)) throw "Неверное значение для поля afr";

if (fields[11] == "Есть") ventmode = true;

}

String^ FunHeater::serialize()

{

String^ ventmode\_str;

if (ventmode) ventmode\_str = "Есть";

else ventmode\_str = "Нет";

return gcnew String(CharToSysString(model) + "|" + CharToSysString(material) + "|" + p.ToString() + "|" + cop.ToString() + "|" +

s.ToString() + "|" + noise.ToString() + "|" + wt.ToString() + "|" +

u.ToString() + "|" + w.ToString() + "|" + CharToSysString(type) + "|" +

afr.ToString() + "|" + ventmode\_str);

}

ElectricFireplace.h

#pragma once

#include "ElectricHE.h"

class ElectricFireplace : public ElectricHE

{

public:

ElectricFireplace();

ElectricFireplace(const ElectricFireplace&);

ElectricFireplace(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type, char\* fuelEffect);

~ElectricFireplace();

void set\_fuelEffect(char\*);

const char\* get\_fuelEffect();

int get\_p() override;

const char\* get\_model() override;

void print\_info(DataTable^ data) override;

void save() override;

void load(String^ serialized\_object) override;

void parse\_fields(cli::array<String^>^ fields) override;

private:

char\* fuelEffect; // Эффект топлива

String^ serialize();

};

ElectricFireplace.cpp

#include "ElectricFireplace.h"

ElectricFireplace::ElectricFireplace()

{

fuelEffect = nullptr;

}

ElectricFireplace::ElectricFireplace(const ElectricFireplace& ref) : ElectricHE(ref)

{

fuelEffect = nullptr; set\_fuelEffect(ref.fuelEffect);

}

ElectricFireplace::ElectricFireplace(int p, int cop, int noise, char\* model, char\* material, double s, double wt, int u, int w, char\* type, char\* fuelEffect) : ElectricHE(p, cop, noise, model, material, s, wt, u, w, type)

{

this->fuelEffect = nullptr; set\_fuelEffect(fuelEffect);

}

ElectricFireplace::~ElectricFireplace()

{

delete[] fuelEffect;

}

void ElectricFireplace::set\_fuelEffect(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (fuelEffect)

{

delete[] fuelEffect; fuelEffect = nullptr;

}

fuelEffect = new char[(strlen(value) + 1)];

strcpy(fuelEffect, value);

}

const char\* ElectricFireplace::get\_fuelEffect()

{

return fuelEffect;

}

int ElectricFireplace::get\_p()

{

return p;

}

const char\* ElectricFireplace::get\_model()

{

return model;

}

void ElectricFireplace::print\_info(DataTable^ data)

{

data->Rows->Add(serialize()->Split('|'));

}

void ElectricFireplace::save()

{

StreamWriter^ writer = gcnew StreamWriter("electricfireplaces.txt", true);

writer->WriteLine(serialize()); writer->Close();

}

void ElectricFireplace::load(String^ serialized\_object)

{

parse\_fields(serialized\_object->Split('|'));

}

void ElectricFireplace::parse\_fields(cli::array<String^>^ fields)

{

ElectricHE::parse\_fields(fields);

set\_fuelEffect(SysStringToChar(fields[10]));

}

String^ ElectricFireplace::serialize()

{

return gcnew String(CharToSysString(model) + "|" + CharToSysString(material) + "|" + p.ToString() + "|" + cop.ToString() + "|" +

s.ToString() + "|" + noise.ToString() + "|" + wt.ToString() + "|" +

u.ToString() + "|" + w.ToString() + "|" + CharToSysString(type) + "|" +

CharToSysString(fuelEffect));

}

Furnace.h

#pragma once

#include "NonElectricHE.h"

class Furnace : public NonElectricHE

{

public:

Furnace();

Furnace(const Furnace&);

Furnace(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel, double depth, char\* cookingPanel);

~Furnace();

void set\_depth(double);

void set\_cookingPanel(char\*);

double get\_depth();

const char\* get\_cookingPanel();

int get\_p() override;

const char\* get\_model() override;

void print\_info(DataTable^ data) override;

void save() override;

void load(String^ serialized\_object) override;

void parse\_fields(cli::array<String^>^ fields) override;

private:

double depth;

char\* cookingPanel;

String^ serialize();

};

Furnace.cpp

#include "Furnace.h"

Furnace::Furnace()

{

depth = 0;

cookingPanel = nullptr;

}

Furnace::Furnace(const Furnace& ref) : NonElectricHE(ref)

{

depth = ref.depth;

cookingPanel = nullptr; set\_cookingPanel(ref.cookingPanel);

}

Furnace::Furnace(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel, double depth, char\* cookingPanel) : NonElectricHE(p, cop, noise, model, material, s, wt, fuel)

{

this->depth = depth;

this->cookingPanel = nullptr; set\_cookingPanel(cookingPanel);

}

Furnace::~Furnace()

{

delete[] cookingPanel;

}

void Furnace::set\_depth(double value)

{

depth = value;

}

void Furnace::set\_cookingPanel(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (cookingPanel)

{

delete[] cookingPanel; cookingPanel = nullptr;

}

cookingPanel = new char[(strlen(value) + 1)];

strcpy(cookingPanel, value);

}

double Furnace::get\_depth()

{

return depth;

}

const char\* Furnace::get\_cookingPanel()

{

return cookingPanel;

}

int Furnace::get\_p()

{

return p;

}

const char\* Furnace::get\_model()

{

return model;

}

void Furnace::print\_info(DataTable^ data)

{

data->Rows->Add(serialize()->Split('|'));

}

void Furnace::save()

{

StreamWriter^ writer = gcnew StreamWriter("furnaces.txt", true);

writer->WriteLine(serialize()); writer->Close();

}

void Furnace::load(String^ serialized\_object)

{

parse\_fields(serialized\_object->Split('|'));

}

void Furnace::parse\_fields(cli::array<String^>^ fields)

{

NonElectricHE::parse\_fields(fields);

if (!Double::TryParse(fields[8], this->depth)) throw "Неверное значение для поля depth";

set\_cookingPanel(SysStringToChar(fields[9]));

}

String^ Furnace::serialize()

{

return gcnew String(CharToSysString(model) + "|" + CharToSysString(material) + "|" + p.ToString() + "|" + cop.ToString() + "|" +

s.ToString() + "|" + noise.ToString() + "|" + wt.ToString() + "|" +

CharToSysString(fuel) + "|" +

depth.ToString() + "|" + CharToSysString(cookingPanel));

}

Fireplace.h

#pragma once

#include "NonElectricHE.h"

class Fireplace : public NonElectricHE

{

public:

Fireplace();

Fireplace(const Fireplace&);

Fireplace(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel, char\* setupType);

~Fireplace();

void set\_setupType(char\*);

const char\* get\_setupType();

int get\_p() override;

const char\* get\_model() override;

void print\_info(DataTable^ data) override;

void save() override;

void load(String^ serialized\_object) override;

void parse\_fields(cli::array<String^>^ fields) override;

private:

char\* setupType;

String^ serialize();

};

Fireplace.cpp

#include "Fireplace.h"

Fireplace::Fireplace()

{

setupType = nullptr;

}

Fireplace::Fireplace(const Fireplace& ref) : NonElectricHE(ref)

{

setupType = nullptr; set\_setupType(ref.setupType);

}

Fireplace::Fireplace(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel, char\* setupType) : NonElectricHE(p, cop, noise, model, material, s, wt, fuel)

{

this->setupType = nullptr; set\_setupType(setupType);

}

Fireplace::~Fireplace()

{

delete[] setupType;

}

void Fireplace::set\_setupType(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (setupType)

{

delete[] setupType; setupType = nullptr;

}

setupType = new char[(strlen(value) + 1)];

strcpy(setupType, value);

}

const char\* Fireplace::get\_setupType()

{

return setupType;

}

int Fireplace::get\_p()

{

return p;

}

const char\* Fireplace::get\_model()

{

return model;

}

void Fireplace::print\_info(DataTable^ data)

{

data->Rows->Add(serialize()->Split('|'));

}

void Fireplace::save()

{

StreamWriter^ writer = gcnew StreamWriter("fireplaces.txt", true);

writer->WriteLine(serialize()); writer->Close();

}

void Fireplace::load(String^ serialized\_object)

{

parse\_fields(serialized\_object->Split('|'));

}

void Fireplace::parse\_fields(cli::array<String^>^ fields)

{

NonElectricHE::parse\_fields(fields);

set\_setupType(SysStringToChar(fields[8]));

}

String^ Fireplace::serialize()

{

return gcnew String(CharToSysString(model) + "|" + CharToSysString(material) + "|" + p.ToString() + "|" + cop.ToString() + "|" +

s.ToString() + "|" + noise.ToString() + "|" + wt.ToString() + "|" +

CharToSysString(fuel) + "|" +

CharToSysString(setupType));

}

Boiler.h

#pragma once

#include "NonElectricHE.h"

class Boiler : public NonElectricHE

{

public:

Boiler();

Boiler(const Boiler&);

Boiler(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel, int numConturs, char\* heaterType, char\* combustionСhamber);

~Boiler();

void set\_numConturs(int value);

void set\_heaterType(char\*);

void set\_combustionСhamber(char\*);

int get\_numConturs();

const char\* get\_heaterType();

const char\* get\_combustionСhamber();

int get\_p() override;

const char\* get\_model() override;

void print\_info(DataTable^ data) override;

void save() override;

void load(String^ serialized\_object) override;

void parse\_fields(cli::array<String^>^ fields) override;

private:

int numConturs;

char\* heaterType;

char\* combustionСhamber;

String^ serialize();

};

Boiler.cpp

#include "Boiler.h"

Boiler::Boiler()

{

numConturs = 0;

heaterType = nullptr; combustionСhamber = nullptr;

}

Boiler::Boiler(const Boiler& ref) : NonElectricHE(ref)

{

numConturs = ref.numConturs;

heaterType = nullptr; set\_heaterType(ref.heaterType);

combustionСhamber = nullptr; set\_combustionСhamber(ref.combustionСhamber);

}

Boiler::Boiler(int p, int cop, int noise, char\* model, char\* material, double s, double wt, char\* fuel, int numConturs, char\* heaterType, char\* combustionСhamber) : NonElectricHE(p, cop, noise, model, material, s, wt, fuel)

{

this->numConturs = numConturs;

this->heaterType = nullptr; set\_heaterType(heaterType);

this->combustionСhamber = nullptr; set\_combustionСhamber(combustionСhamber);

}

Boiler::~Boiler()

{

delete[] heaterType;

delete[] combustionСhamber;

}

void Boiler::set\_numConturs(int value)

{

numConturs = value;

}

void Boiler::set\_heaterType(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (heaterType)

{

delete[] heaterType; heaterType = nullptr;

}

heaterType = new char[(strlen(value) + 1)];

strcpy(heaterType, value);

}

void Boiler::set\_combustionСhamber(char\* value)

{

if (value == nullptr || strlen(value) == 0) throw "Попытка установки пустого значения.";

if (combustionСhamber)

{

delete[] combustionСhamber; combustionСhamber = nullptr;

}

combustionСhamber = new char[(strlen(value) + 1)];

strcpy(combustionСhamber, value);

}

int Boiler::get\_numConturs()

{

return numConturs;

}

const char\* Boiler::get\_heaterType()

{

return heaterType;

}

const char\* Boiler::get\_combustionСhamber()

{

return combustionСhamber;

}

int Boiler::get\_p()

{

return p;

}

const char\* Boiler::get\_model()

{

return model;

}

void Boiler::print\_info(DataTable^ data)

{

data->Rows->Add(serialize()->Split('|'));

}

void Boiler::save()

{

StreamWriter^ writer = gcnew StreamWriter("boilers.txt", true);

writer->WriteLine(serialize()); writer->Close();

}

void Boiler::load(String^ serialized\_object)

{

parse\_fields(serialized\_object->Split('|'));

}

void Boiler::parse\_fields(cli::array<String^>^ fields)

{

NonElectricHE::parse\_fields(fields);

if (!Int32::TryParse(fields[8], this->numConturs)) throw "Неверное значение для поля numConturs";

set\_heaterType(SysStringToChar(fields[9]));

set\_combustionСhamber(SysStringToChar(fields[10]));

}

String^ Boiler::serialize()

{

return gcnew String(CharToSysString(model) + "|" + CharToSysString(material) + "|" + p.ToString() + "|" + cop.ToString() + "|" +

s.ToString() + "|" + noise.ToString() + "|" + wt.ToString() + "|" +

CharToSysString(fuel) + "|" +

numConturs.ToString() + "|" + CharToSysString(heaterType) + "|" + CharToSysString(combustionСhamber));

}

GeneralizedContainer.h

#pragma once

#include <list>

#include "IHeatingEl.h"

class GeneralizedContainer

{

protected:

void \_push\_front(void\* const& p);

void \_push\_back(void\* const& p);

void \_pop\_front();

void \_pop\_back();

void\*& \_back();

void\*& \_front();

private:

list<void\*> ListInterfacePointers;

};

GeneralizedContainer.cpp

#include "GeneralizedContainer.h"

void GeneralizedContainer::\_push\_front(void\* const& p)

{

ListInterfacePointers.push\_front(p);

}

void GeneralizedContainer::\_push\_back(void\* const& p)

{

ListInterfacePointers.push\_back(p);

}

void GeneralizedContainer::\_pop\_front()

{

ListInterfacePointers.pop\_front();

}

void GeneralizedContainer::\_pop\_back()

{

ListInterfacePointers.pop\_back();

}

void\*& GeneralizedContainer::\_front()

{

return ListInterfacePointers.front();

}

void\*& GeneralizedContainer::\_back()

{

return ListInterfacePointers.back();

}

TemplateContainer.h

#pragma once

#include "GeneralizedContainer.h"

template <class T>

class TemplateContainer : private GeneralizedContainer

{

public:

void push\_front(const T& p);

void push\_back(const T& p);

void pop\_front();

void pop\_back();

T& back();

T& front();

};

// --------------------------------------------------------------------------------

template <class T>

void TemplateContainer<T>::push\_front(const T& p)

{

\_push\_front(p);

}

template <class T>

void TemplateContainer<T>::push\_back(const T& p)

{

\_push\_back(p);

}

template <class T>

void TemplateContainer<T>::pop\_front()

{

\_pop\_front();

}

template <class T>

void TemplateContainer<T>::pop\_back()

{

\_pop\_back();

}

template<class T>

T& TemplateContainer<T>::back()

{

return reinterpret\_cast<T&>(\_back());

}

template<class T>

T& TemplateContainer<T>::front()

{

return reinterpret\_cast<T&>(\_front());

}

Storage.h

#pragma once

// Подключаем классы-контейнеры

#include "TemplateContainer.h"

#include "DynamicMatrix.h"

// Подключаем классы иерархии

#include "Convector.h"

#include "Radiator.h"

#include "FunHeater.h"

#include "ElectricFireplace.h"

#include "Furnace.h"

#include "Fireplace.h"

#include "Boiler.h"

class Storage

{

public:

// Контейнер под указатели

static TemplateContainer<IHeatingEl\*> container;

// Динамические матрицы для хранения объектов

static DynamicMatrix<Convector> convectors;

static DynamicMatrix<Radiator> radiators;

static DynamicMatrix<FunHeater> funheaters;

static DynamicMatrix<ElectricFireplace> electricfireplaces;

static DynamicMatrix<Furnace> furnaces;

static DynamicMatrix<Fireplace> fireplaces;

static DynamicMatrix<Boiler> boilers;

private:

Storage();

};

Storage.cpp

#include "Storage.h"

TemplateContainer<IHeatingEl\*> Storage::container;

DynamicMatrix<Convector> Storage::convectors;

DynamicMatrix<Radiator> Storage::radiators;

DynamicMatrix<FunHeater> Storage::funheaters;

DynamicMatrix<ElectricFireplace> Storage::electricfireplaces;

DynamicMatrix<Furnace> Storage::furnaces;

DynamicMatrix<Fireplace> Storage::fireplaces;

DynamicMatrix<Boiler> Storage::boilers;

Functions.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <string.h>

#include <typeinfo>

#include <memory>

using namespace std;

using namespace System;

using namespace System::IO;

using namespace System::Data;

using namespace System::Collections;

using namespace System::Runtime::InteropServices;

String^ CharToSysString(const char\* ch);

char\* SysStringToChar(String^ string);

Functions.cpp

#include "Functions.h"

String^ CharToSysString(const char\* ch)

{

return gcnew String(ch);

}

char\* SysStringToChar(String^ string)

{

return(char\*)(void\*)Marshal::StringToHGlobalAnsi(string);

}

Processing.h

#pragma once

#include "Storage.h"

template <typename T>

void PrintInfoObjects(T matrix, DataTable^ data);

template <typename T>

void UploadObjects(T matrix);

template <typename T>

String^ GetAllModels(T matrix);

template <typename T>

int GetSummP(T matrix);

void DownloadObjects();

// ------------------------------------------------------------------------------------------------------

template<typename T>

inline void PrintInfoObjects(T matrix, DataTable^ data)

{

data->Rows->Clear();

for (int i = 0; i < matrix->current\_size; Storage::container.front() = &matrix->at(i), Storage::container.front()->print\_info(data), i++);

}

template<typename T>

inline void UploadObjects(T matrix)

{

for (; !matrix->empty(); Storage::container.front() = &matrix->back(), Storage::container.front()->save(), matrix->pop\_back());

Storage::container.pop\_front();

}

template<typename T>

inline String^ GetAllModels(T matrix)

{

String^ str = "";

for (int i = 0; i < matrix->current\_size; i++)

str += CharToSysString(matrix->at(i).get\_model()) + ", ";

str += "\n";

return str;

}

template<typename T>

inline int GetSummP(T matrix)

{

int sum = 0;

for (int i = 0; i < matrix->current\_size; i++)

sum += matrix->at(i).get\_p();

return sum;

}

Processing.cpp

#include "Processing.h"

void DownloadObjects()

{

String^ path = "convectors.txt";

Storage::container.push\_back(new Convector);

StreamReader^ reader = gcnew StreamReader(path);

auto convector = make\_shared<Convector>();

for (; !reader->EndOfStream; convector->load(reader->ReadLine()), Storage::convectors.push\_back(\*convector));

reader->Close();

path = "radiators.txt";

Storage::container.push\_back(new Radiator);

reader = gcnew StreamReader(path);

auto radiator = make\_shared<Radiator>();

for (; !reader->EndOfStream; radiator->load(reader->ReadLine()), Storage::radiators.push\_back(\*radiator));

reader->Close();

path = "funheaters.txt";

Storage::container.push\_back(new FunHeater);

reader = gcnew StreamReader(path);

auto funheater = make\_shared<FunHeater>();

for (; !reader->EndOfStream; funheater->load(reader->ReadLine()), Storage::funheaters.push\_back(\*funheater));

reader->Close();

path = "electricfireplaces.txt";

Storage::container.push\_back(new ElectricFireplace);

reader = gcnew StreamReader(path);

auto electricfireplace = make\_shared<ElectricFireplace>();

for (; !reader->EndOfStream; electricfireplace->load(reader->ReadLine()), Storage::electricfireplaces.push\_back(\*electricfireplace));

reader->Close();

path = "furnaces.txt";

Storage::container.push\_back(new Furnace);

reader = gcnew StreamReader(path);

auto furnace = make\_shared<Furnace>();

for (; !reader->EndOfStream; furnace->load(reader->ReadLine()), Storage::furnaces.push\_back(\*furnace));

reader->Close();

path = "fireplaces.txt";

Storage::container.push\_back(new Fireplace);

reader = gcnew StreamReader(path);

auto fireplace = make\_shared<Fireplace>();

for (; !reader->EndOfStream; fireplace->load(reader->ReadLine()), Storage::fireplaces.push\_back(\*fireplace));

reader->Close();

path = "boilers.txt";

Storage::container.push\_back(new Boiler);

reader = gcnew StreamReader(path);

auto boiler = make\_shared<Boiler>();

for (; !reader->EndOfStream; boiler->load(reader->ReadLine()), Storage::boilers.push\_back(\*boiler));

reader->Close();

}

Приложение 3. Руководство пользователя

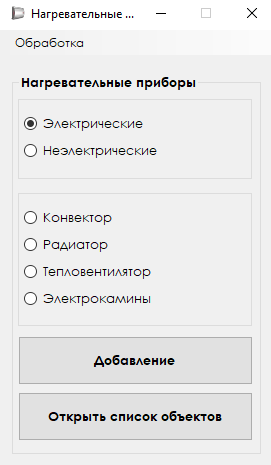
1. Для запуска программы, необходимо на приложенном носителе, в папке KR – Debug найти исполняемый файл KR.exe и запустить его.
2. Главное окно программы состоит из меню с пунктом «Обработка», группы радио переключателей и двух кнопок: «Добавление» и «Открыть список объектов» (рис. П3.1).

Рис. П3.1. Главное окно программы.

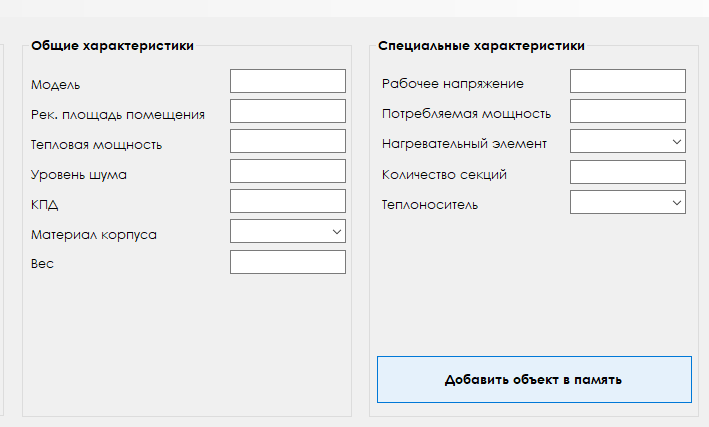
1. Для добавления объекта в память необходимо выбрать тип объекта с помощью радио переключателей и нажать кнопку «Добавление», затем заполнить все имеющееся поля и нажать кнопку «Добавить объект в память» (рис. П3.2).
2. Для сохранения добавленных объектов в файл необходимо выбрать пункт меню «Обработка» – «Выгрузить объекты в файлы».

Рис. П3.2. Добавление объекта.

1. Для загрузки объектов из файлов необходимо выбрать пункт меню «Обработка» – «Загрузить объекты из файлов».
2. Для того чтобы вывести суммарную мощность всех нагревательных приборов, загруженных в памяти на данный момент, необходимо выбрать пункт меню «Обработка» – «Подсчитать суммарную мощность».

Для того чтобы вывести модели всех нагревательных приборов, загруженных в памяти на данный момент, необходимо выбрать пункт меню «Обработка» – «Вывести все модели».

1. Для просмотра объектов какого-либо типа необходимо выбрать тип объекта с помощью радио переключателей и нажать кнопку «Открыть список объектов» (рис. П3.3).

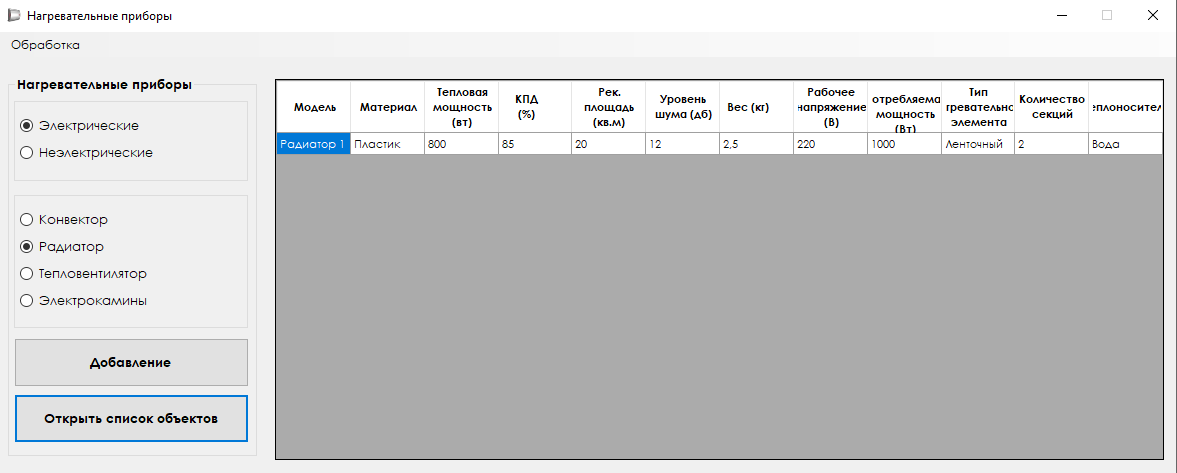


Рис. П3.3. Просмотр объектов.