Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ФАЙЛОВЫЙ МЕНЕДЖЕР

БГУИР КП 1-40 02 01 309 ПЗ

Студент: Каленчиц А. В.

Руководитель: Ассистент кафедры ЭВМ
Марзалюк А. В.

МИНСК 2022

СОДЕРЖАНИЕ

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ……...5

ВВЕДЕНИЕ……..7

1 ОБЗОР НА ЛИТЕРАТУРУ…….…….....8

2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ…….....9

2.1 Класс *MainWindow* ……...9

2.2 Класс *AdditionalWindow*……..10

2.3 Класс *ConfirmDelete*……11

2.4 Класс *List*……..11

2.5 Класс *Iterator*……....12

2.6 Класс *CreateChoise*……..12

2.7 Класс *Exception*……13

2.8 Класс *RenameWindow*……..13

2.9 Класс *SearchWindow*……14

2.10 Класс *System*……...14

2.11 Класс *File*……....15

2.12 Класс *Folder*……...15

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ……17

3.1 Алгоритм по шагам метода создания объекта файловой системы (o*n_btnCreate_clicked())*……17

3.2 Алгоритм по шагам метода удаления выбранного объекта (*on_btnDelete_clicked()*)……20
3.3 Блок-схема метода *on_lvSource_clicked()*……..22
3.4 Блок-схема метода *on_lvSource_doubleClicked()*……..22
4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ……23
ЗАКЛЮЧЕНИЕ…….26
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ…..…….27
ПРИЛОЖЕНИЕ А…….28
ПРИЛОЖЕНИЕ Б……..61
ПРИЛОЖЕНИЕ В……..68
ПРИЛОЖЕНИЕ Г……..69
ПРИЛОЖЕНИЕ Д…..……70

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Тема проекта – «Файловый менеджер». Проект должен быть простым в использовании и понятным пользователю, облегчать работу с файловой системой и управлять данными, содержащимися на устройстве.

Необходимо отразить основные функции и свойства данного вида программного обеспечения, раскрыть его достоинства и показать возможные причины некоторых недостатков.

Язык программирования C++.

ВВЕДЕНИЕ

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, являющихся экземплярами определённых классов, каждый из которых представляет собой часть иерархии наследования. Основными принципами объектно-ориентированного программирования являются абстракция (контекстное понимание предмета, формализуемое в виде класса), инкапсуляция (упаковывание данных и поведения в обособленные части единого компонента), наследование (концепция, позволяющая абстрактному типу данных наследовать данные и функционал существующего абстрактного типа данных) и полиморфизм (способность функции обрабатывать данные разных типов).

Среди многих современных языков программирования язык C++ выделяется ориентацией на объектно-ориентированное программирование.

C++ — компилируемый статически типизированный язык программирования общего назначения. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. В сравнении с его предшественником — языком C — наибольшее внимание уделено поддержке объектно-ориентированного и обобщенного программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также компьютерных игр. Поэтому данный язык программирования является хорошим выбором для реализации курсового проекта.

1 ОБЗОР НА ЛИТЕРАТУРУ

Файловый менеджер — программа, предоставляющая интерфейс пользователя для работы с файловой системой. Файловый менеджер позволяет выполнять наиболее частые операции над файлами — создание, открытие/проигрывание/просмотр, редактирование, перемещение, переименование, копирование, удаление, отображение атрибутов и свойств, поиск файлов и др. Помимо основных функций, многие файловые менеджеры включают ряд дополнительных возможностей, например, таких, как резервное копирование, управление принтерами и пр.

Выделяют различные типы файловых менеджеров, например:

- Навигационные (пространственные). Представители данного типа при работе с очень большим количеством файлов значительно заменяют работу системы.

  Отсутствие второй панели для копирования или перемещения файлов иногда расценивается как недостаток данного типа файловых менеджеров (если учесть, что большинство файловых менеджеров позволяют управлять всеми действиями с клавиатуры, то скорость работы с двумя панелями может быть больше, чем скорость работы одной панели).

  Обычный Проводник Windows является представителем пространственных файловых менеджеров.
- Двухпанельные (ортодоксальные). В общем случае имеют две равноценные панели для списка файлов, дерева каталогов и т. п.

  К ортодоксальным файловым менеджерам относятся: Norton Commander, Total Commander и др.
- Консольные. Такие менеджеры могут быть очень полезны в повседневных задачах, при управлении файлами на локальном

компьютере. Консольным файловым менеджерам требуется меньше системных ресурсов, чем аналогичным по функциональности файловым менеджерам с графическим интерфейсом.

Консольными файловыми менеджерами являются FAR Manager, GNU Midnight Commander и др.

В данном курсовом проекте будет представлен файловый менеджер навигационного типа. Данный тип выбран как более привычный и интуитивно понятный пользователю операционной системы Windows, на которой реализована программа для курсового проекта.

Для написания программы и реализации графического интерфейса была выбрана среда разработки Qt Creator, так как она включает фреймворк Qt, который является полезным инструментом в реализации программ, отражающих работу внутренней системы устройства.

Qt — фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++.

Qt является полностью объектно-ориентированным и позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения (элементы графического интерфейса, классы для работы с базами данных).

2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

Диаграмма классов UML представлена в ПРИЛОЖЕНИИ В.

Для создания навигационного файлового менеджера в программе были реализованы различные классы в зависимости от требуемого функционала.

2.1 Класс *MainWindow*

Отображает внешний вид и работу основного окна приложения. Атрибуты класса:

filePath - абсолютный путь выбранного файла;

fileName – имя выбранного файла;

dirPath – абсолютный путь выбранной директории;

dirName – имя выбранной директории;

window - объект класса *SearchWindow* для вывода результатов поиска;

*model - указатель для связи с моделью данных для файловой системы;

f - объект класса *File* для выполнения операций с текстовыми файлами;

d - объект класса *Folder* для выполнения операций с директориями;

**file* - указатель на объект класса *System* для получения адреса объекта класса *File;*

**folder* - указатель на объект класса *System* для получения адреса объекта класса *Folder;*

**ui* - указатель для связи с главным окном приложения в классе основной формы.

Методы класса:

on_lvSource_doubleClicked(const QModelIndex &index) - слот двойного нажатия на панель *"lvSource"*;

on_lvSource_clicked(const QModelIndex &index) - слот нажатия на панель *"lvSource"*;

on_btnCreate_clicked() - слот нажатия на кнопку *"Create"*;

on_btnDelete_clicked() - слот нажатия на кнопку *"Delete"*;

on_btnCopy_clicked() - слот нажатия на кнопку *"Copy"*;

on_btnReplace_clicked() - слот нажатия на кнопку *"Replace"*;

on_btnRename_clicked() - слот нажатия на кнопку *"Rename"*;

on_btnSearch_clicked() - слот нажатия на кнопку *"Search"*;

on_lineSearch_textEdited(const QString &arg1) - слот ввода имени для поиска;

on_listWidget_currentTextChanged(const QString &currentText) - слот копирования текущего местоположения в буфер обмена.

2.2 Класс *AdditionalWindow*

Отображает дополнительное окно и его функционал при копировании или перемещении системного объекта (здесь файла или директории).

Атрибуты класса:

**ui* - указатель для связи с соответствующим ui-файлом;

**model* - указатель для связи с моделью данных для файловой системы;

dir - переменная для хранения информации об определенной директории;

cancel - флаг срабатывания отмены;

Методы класса:

SetCancel(bool) - метод установки отмены;

Cancel() - метод срабатывания отмены;

SetQDir(QDir &) - метод получения переменной директории;

GetQDir() - метод передачи переменной директории;

on_btnOK_clicked() - слот нажатия на *"OK";*

on_btnCancel_clicked() - слот нажатия на *"Cancel";*

on_listView_doubleClicked(const QModelIndex &index) - слот нажатия на панель *"listView".*

2.3 Класс *ConfirmDelete*

Отвечает за работу всплывающего окна для подтверждения удаления.

Атрибуты класса:

**ui* - указатель для связи с соответствующим ui-файлом;

confirm - флаг установки подтверждения.

Методы класса:

SetConfirm(bool) - метод установки подтверждения удаления;

Confirm() - метод срабатывания подтверждения удаления;

on_btnOK_clicked() - слот нажатия на *"OK";*

on_btnCancel_clicked() - слот нажатия на *"Cancel".*

2.4 Класс *List*

Класс контейнера списочного типа, предназначенный для хранения путей файлов, имена которых совпали с поисковым запросом.

Атрибуты класса:

**node* - указатель на текущий элемент;

**head* - указатель на первый элемент;

**tail* - указатель на последний элемент;

Iterator - вложенный итератор контейнера *List.*

Методы класса:

Push(QString) - метод добавления элемента;

PeekHead(bool flag) - метод передачи первого либо текущего элемента;

Pop() - метод очистки контейнера;

IsEmpty() - метод проверки, пуст ли контейнер.

2.5 Класс *Iterator*

Вложенный итератор контейнера *List*.

Атрибуты класса:

**current* - указатель на текущий элемент.

Методы класса:

SetIter(Node node)* - метод установки итератора на элемент, связанный с указателем **node*;

operator ++() - перегруженный оператор для перехода к следующему элементу;

operator *() - перегруженный оператор для получения информации текущего элемента.

2.6 Класс *CreateChoise*

Реализует работу всплывающего окна для выбора системного объекта при его создании.

Атрибуты класса:

**ui* - указатель для связи с соответствующим ui-файлом;

fileName - переменная для хранения имени созданного файла;

folderName - переменная для хранения имени созданной директории;

file - флаг выбора файла;

folder - флаг выбора директории.

Методы класса:

ChooseFile(bool) - метод установки создания файла;

ChooseFolder(bool) - метод установки создания директории;

GetFile() - метод, сообщающий о выборе создании файла;

GetFolder() - метод, сообщающий о выборе создания директории;

on_btnFile_clicked() - слот выбора о создании файла;

on_btnFolder_clicked() - слот выбора о создании директории;

on_btnCancel_clicked() - слот нажатия на *"Cancel"*.

2.7 Класс *Exception*

Данный класс является базовым для пяти других классов (*RootDirectoryException*, *ChooseException*, *PerformationException*, *BadAllocException*, *ExceptionEmpty*), отображающих тип возможных исключений во время работы приложения.

Атрибуты класса:

nameOperation - название операции, при которой обработано исключение;

problem - суть проблемы.

Методы класса:

SetException(QString str1, QString str2) - создание информации, которая будет предоставлена пользователю;

GetException(QWidget parent)* - предоставление пользователю информации об исключении.

2.8 Класс *RenameWindow*

Создан для возможности переименования системного объекта во всплывающем окне.

Атрибуты класса:

**ui* - указатель для связи с соответствующим ui-файлом;

name - переменная для хранения нового имени выбранного файла.

Методы класса:

ChooseFile(bool) - метод установки создания файла;

ChooseFolder(bool) - метод установки создания директории;

GetFile() - метод, сообщающий о выборе создании файла;

GetFolder() - метод, сообщающий о выборе создания директории;

on_btnFile_clicked() - слот выбора о создании файла;

on_btnFolder_clicked() - слот выбора о создании директории;

on_btnCancel_clicked() - слот нажатия на *"Cancel"*.

2.9 Класс *SearchWindow*

Реализует поиск системного объекта по имени, указанном в главном окне приложения в соответствующей текстовой строке.

Атрибуты класса:

**ui* - указатель для связи с соответствующим ui-файлом;

searchName - переменная для хранения имени для поиска;

list - контейнер для хранения путей файлов, имена которых совпали с введенным;

iter - итератор контейнера для управления его элементами.

Методы класса:

SetName(const QString) - метод получения имени для поиска;

Search(QDir &) - метод поиска по имени;

SetUi() - метод передачи результатов поиска для отображения;

ResetUi() - метод очистки окна отображения результатов поиска;

on_btnOK_clicked() - слот нажатия на *"OK"*.

2.10 Класс *System*

Данный класс, предназначенный для работы с системными объектами, является абстрактным базовым классом для двух производных от него классов: *File* и *Folder*.

Методы класса:

Create() – чисто виртуальный метод создания системного объекта;

Delete() – чисто виртуальный метод удаления системного объекта;

Rename(QString) – чисто виртуальный метод переименования системного объекта;

Copy(QString) - чисто виртуальный метод копирования системного объекта;

SetPath(QString) – чисто виртуальный метод установки пути системного объекта;

GetPath() - чисто виртуальный метод установки пути системного объекта.

2.11 Класс *File*

Класс предназначен для работы с текстовыми файлами.

Атрибуты класса:

filePath – путь выбранного файла.

Методы класса:

Create() - метод создания текстового файла;

Delete() - метод удаления текстового файла;

Rename(QString) - метод переименования текстового файла;

Copy(QString) - метод копирования текстового файла;

SetPath(QString) - метод установки пути текстового файла;

GetPath() - метод передачи пути текстового файла.

2.12 Класс *Folder*

Класс предназначен для работы с директориями.

Атрибуты класса:

dirPath – путь выбранной директории.

Методы класса:

Create() - метод создания директории;

Delete() - метод удаления директории;

Rename(QString) - метод переименования директории;

Copy(QString) - метод копирования директории;

SetPath(QString) - метод установки пути директории;

GetPath() - метод передачи пути директории.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Алгоритм по шагам метода создания объекта файловой системы (o*n_btnCreate_clicked())*

Шаг 1. Получение текущего местоположения в файловой системе: создание объекта *dir* класса *QDir* со значением *model->filePath(ui->lvSource->rootIndex()),* приведенным к типу *QDir.*

Шаг 2. Переход в охранный блок *try.*

Шаг 3. Если текущее местоположение (*QString dir.absolutePath()*) – корневая директория, переход к шагу 4, иначе переход к шагу 6.

Шаг 4. Переход в блок *catch(RootDirectoryException error)*.

Шаг 5. Конец.

Шаг 6. Создание объекта *window* класса *CreateWindow*.

Шаг 7. Вызов всплывающего окна для выбора типа системного объекта и переход в область выполнения класса *CreateWindow.*

Шаг 8. Если будущий системный объект - файл (метод *QString GetFile()* класса *CreateWindow* передает непустое значение), переход к шагу 9, иначе переход к шагу 33.

Шаг 9. Создание объекта *isFile* класса *RenameWindow.*

Шаг 10. Вызов всплывающего окна для ввода имени файла и переход в область выполнения класса *RenameWindow*.

Шаг 11. Инициализация пути *QString createPath* для создания файла.

Шаг 12. Установка флага *bool fileExists* в ложное значение.

Шаг 13. Переход в цикл *foreach* для прохода по директории, в которой будет создан файл, для поиска файлов и директорий с введенным именем. Если пройдены все системные объекты данной директории, переход к шагу 17 (выход из цикла).

Шаг 14. Если системный объект с таким именем существует (*QString files.fileName()* равен *QString isFile.GetName()*), переход к шагу 15, иначе переход к шагу 16.

Шаг 15. Установка флага *bool fileExists* в *true* и переход к шагу 17 (выход из цикла).

Шаг 16. Переход к шагу 13 (переход к следующему системному объекту).

Шаг 17. Если файлов и директорий с таким именем нет (флаг *bool fileExists* принимает ложное значение), переход к шагу 18, иначе переход к шагу 31.

Шаг 18. Если в имени файла есть расширение *“.txt”*, переход к шагу 19, иначе переход к шагу 24.

Шаг 19. Инициализация пути файла значением *QString createPath.*

Шаг 20. Если метод *bool Create()* класса *File* возвращает истинное значение, переход к шагу 21, иначе переход к шагу 22.

Шаг 21. Создание файла с именем, возвращаемым методом *QString GetFile()* класса *RenameWindow,* и путем *QString createPath* выполнено успешно. Переход к шагу 23.

Шаг 22. Переход в блок *catch(PerformationException error).*

Шаг 23. Конец.

Шаг 24. Если в имени файла расширение не совпадает с *“.txt”,* переход к шагу 25.

Шаг 25. Создание объекта *file* класса *QFile* с путем *QString createPath.*

Шаг 26. Если не удалось открыть файл *QFile file* (файл не создан), переход к шагу 27, иначе переход к шагу 29.

Шаг 27. Переход в блок *catch(PerformationException error).*

Шаг 28. Конец.

Шаг 29. Файл существует. Закрытие файла *QFile file*.

Шаг 30. Конец.

Шаг 31. Переход в блок *catch(PerformationException error).*

Шаг 32. Конец.

Шаг 33. Если будущий системный объект – директория (метод *QString GetFolder()* класса *CreateWindow* передает непустое значение), переход к шагу 34, иначе переход к шагу .

Шаг 34. Создание объекта *isDir* класса *RenameWindow.*

Шаг 35. Вызов всплывающего окна для ввода имени директории и переход в область выполнения класса *RenameWindow.*

Шаг 36. Инициализация пути *QString createPath* для создания директории.

Шаг 37. Установка флага *bool dirExists* в ложное значение.

Шаг 38. Переход в цикл *foreach* для прохода по директории, в которой будет создана директория, для поиска файлов и директорий с введенным именем. Если пройдены все системные объекты данной директории, переход к шагу 42 (выход из цикла).

Шаг 39. Если системный объект с таким именем существует (*QString dirs.fileName()* равен *QString isDir.GetName()*), переход к шагу 40, иначе переход к шагу 41.

Шаг 40. Установка флага *bool dirExists* в *true* и переход к шагу 42 (выход из цикла).

Шаг 41. Переход к шагу 38 (переход к следующему системному объекту).

Шаг 42. Если файлов и директорий с таким именем нет (флаг *bool dirExists* принимает ложное значение), переход к шагу 43, иначе переход к шагу 46.

Шаг 43. Инициализация пути директории значением *QString createPath.*

Шаг 44. Если метод *bool Create()* класса *Folder* возвращает истинное значение, переход к шагу 45, иначе переход к шагу 46.

Шаг 45. Создание директории с именем, возвращаемым методом *QString GetFolder()* класса *RenameWindow,* и путем *QString createPath* выполнено успешно. Переход к шагу 47.

Шаг 46. Переход в блок *catch(PerformationException error).*

Шаг 47. Конец.

3.2 Алгоритм по шагам метода удаления выбранного объекта (*on_btnDelete_clicked()*)

Шаг 1. Получение текущего местоположения в файловой системе: создание объекта *dir* класса *QDir* со значением *model->filePath(ui->lvSource->rootIndex()),* приведенным к типу *QDir.*

Шаг 2. Переход в охранный блок *try.*

Шаг 3. Если текущее местоположение (*QString dir.absolutePath()*) – корневая директория, переход к шагу 4, иначе переход к шагу 6.

Шаг 4. Переход в блок *catch(RootDirectoryException error)*.

Шаг 5. Конец.

Шаг 6. Если системный объект не выбран (*QString filePath* и *QString dirPath* принимают значение “”), переход к шагу 7, иначе переход к шагу 9.

Шаг 7. Переход в блок *catch(ChoiseException error).*

Шаг 8. Конец.

Шаг 9. Если выбран файл (*QString filePath* не пустой, а *QString dirPath* принимают значение “”), переход к шагу 10, иначе переход к шагу 29.

Шаг 10. Создание объекта *window* класса *ConfirmDelete.*

Шаг 11. Вызов всплывающего окна для подтверждения удаления и переход в область выполнения класса *ConfirmDelete.*

Шаг 12. Если метод *bool Confirm()* класса *ConfirmDelete* передает ложное значение, переход к шагу 13, иначе переход к шагу 15.

Шаг 13. Переход в блок *catch(PerformationException error).*

Шаг 14. Конец.

Шаг 15. Если в имени файла есть расширение *“.txt”*, переход к шагу 16, иначе переход к шагу 23.

Шаг 16. Инициализация пути файла значением *QString filePath.*

Шаг 17. Если метод *bool Delete()* класса *File* возвращает истинное значение, переход к шагу 18, иначе переход к шагу 19.

Шаг 18. Удаление файла с путем *QString filePath* выполнено успешно. Переход к шагу 21.

Шаг 19. Переход в блок *catch(PerformationException error).*

Шаг 20. Конец.

Шаг 21. Очистка пути файла.

Шаг 22. Конец.

Шаг 23. Если в имени файла расширение не совпадает с *".txt",* переход к шагу 22.

Шаг 24. Если метод *bool remove(QString)* класса *QFile* передает ложное значение, переход к шагу 25, иначе переход к шагу 27.

Шаг 25. Переход в блок *catch(PerformationException error).*

Шаг 26. Конец.

Шаг 27. Удаление файла с путем *QString filePath* выполнено успешно.

Шаг 28. Очистка пути файла. Конец.

Шаг 29. Если выбрана директория (*QString dirPath* не пустой, а *QString filePath* принимает значение ""), переход к шагу 30.

Шаг 30. Создание объекта *window* класса *ConfirmDelete.*

Шаг 31. Вызов всплывающего окна для подтверждения удаления и переход в область выполнения класса *ConfirmDelete.*

Шаг 32. Если метод *bool Confirm()* класса *ConfirmDelete* передает ложное значение, переход к шагу 33, иначе переход к шагу 35.

Шаг 33. Переход в блок *catch(PerformationException error).*

Шаг 34. Конец.

Шаг 35. Получение местоположения выбранной директориии в файловой системе: создание объекта *dir* класса *QDir* со значением *QString dirPath,* приведенным к типу класса *QDir.*

Шаг 36. Если выбранная директория не пустая, переход к шагу 37, иначе переход к шагу 40.

Шаг 37. Если функция *bool RecursiveDelete(QDir&, System*, System*)* передает ложное значение, переход к шагу 38, иначе переход к шагу 40.

Шаг 38. Переход в блок *catch (PerformationException error).*

Шаг 39. Конец.

Шаг 40. Директория пуста. Инициализация пути директории значением *QString dirPath.*

Шаг 41. Если метод *bool Delete()* класса *Folder* возвращает истинное значение, переход к шагу 42, иначе переход к шагу 43.

Шаг 42. Удаление директории с путем *QString dirPath* выполнено успешно. Переход к шагу 45.

Шаг 43. Переход в блок *catch (PerformationException error).*

Шаг 44. Конец.

Шаг 45. Очистка пути директории.

Шаг 46. Конец.

3.3 Блок-схема метода *on_lvSource_clicked()*

Данный метод отвечает за получение пути системного объекта, на который произвели нажатие.

Блок-схема данного метода приведена в приложении Г.

3.4 Блок-схема метода *on_lvSource_doubleClicked()*

Данный метод предназначен для открытия системного объекта посредством двойного нажатия.

Блок-схема данного метода приведена в приложении Г.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В директории, содержащей файл проекта, помимо файлов, отвечающих за корректную работу приложения, находятся еще четыре директории(с именами “.h”, “.cpp”, “.ui” и “resources”), в которых содержатся основные файлы программы: заголовочные файлы, файлы исходников и форм ui, а также изображения, которые используются в программе.

Чтобы воспользоваться программой, необходимо перейти в Qt Creator, создать проект Qt Widget Application, добавить в него директории “.h”, “.cpp”, “.ui” и “resources” и запустить проект.

Появится главное окно программы с основными элементами (см. рисунок Б.1).

В верхней части окна представлены кнопки с иконками, изображающими возможные операции над файлом или директорией. Хотя иконки интуитивно понятны, при наведении на кнопки с ними всплывают подсказки, поясняющие работу каждой кнопки (см. рисунок Б.2).

Если во время выполнения программа обработает какое-либо исключение, появится предупреждение об этом (см. рисунок Б.3).

Вышеуказанный пример лишь иллюстрация одного из обработанных исключений. Подробно осмотреть возможные исключения можно непосредственно в коде программы.

При нажатии на кнопку для создания появится окно для отмены текущей операции, либо для выбора типа создаваемого объекта: файла или директории (см. рисунок Б.4).

При выборе одного из предложенных типов появится окно для указания имени создаваемого объекта (см. рисунок Б.5).

Чтобы совершить операции удаления, копирования, перемещения и переименования системного объекта, необходимо нажать на него в панели отображения файловой системы.

При нажатии на кнопку для удаления появится дополнительное окно для подтверждения текущей операции, либо для ее отмены (см. рисунок Б.6).

Рекомендуется обратить внимание на то, что реализованная программа выполняет удаление в обход Корзины, то есть вернуть удаленные данные невозможно!

При нажатии на кнопку копирования и перемещения объекта появится дополнительное окно для выбора директории, в которую он будет скопирован или перемещен (см. рисунок Б.7).

При нажатии на кнопку переименования объекта появится окно для ввода нового имени (см. рисунок Б.8).

Рекомендуется обратить внимание на то, что, если нужно ввести имя файла, необходимо указать его расширение (желательно то же, что и было до переименования)! Для предупреждения недоразумений (потеря данных при изменении расширения) при наведении на область ввода имени появится соответствующая подсказка.

Ниже кнопок основных операций над системными объектами располагается строка, показывающая текущее местонахождение пользователя в файловой системе (см. рисунок Б.9).

По центру главного окна программы расположена панель, которая выполняет обзор файловой системы. С помощью этой панели производится движение по файловой системе путем двойного клика по ее элементам.

Элементы “.” и “..” реализованы программно. Они выполняют переходы в корневую и родительскую директории соответственно.

Ниже панели обзора файловой системы расположены основные свойства выбранного объекта. Свойства состоят из даты последнего изменения выбранного объекта, его размера и типа (см. рисунок Б.10).

В нижней части главного окна программы расположена строка для поиска системных объектов по имени. Как и в случае с переименованием, для поиска вместе с названием файла необходимо указать и его расширение. В

противном случае будет выполнен поиск по директориям. Для строки поиска также реализована соответствующая подсказка (см. рисунок Б.11).

После нажатия на кнопку “Search” появится окно с результатами поиска (см. рисунок Б.12).

В результате программа позволяет выполнять все основные манипулирования системными объектами: создание, удаление, копирование, перемещение, переименование, поиск, отображение основных свойств, визуальное представление.

ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта были изучены такие темы, применение объектно-ориентированного программирования при создании приложений, реализация графической оболочки приложения при помощи фреймворка Qt, система сигналов и слотов в среде разработки Qt Creator.

Реализованный файловый менеджер навигационного типа хоть и имеет понятный интерфейс и довольно удобный функционал, все же показан как пример и не является пределом совершенства, может быть доработан и улучшен в различных направлениях. Такими направлениями могут быть:

- Возможность множественного выбора системных объектов.
- Реализация контейнера, выполняющего функцию Корзины.
- Возможность отмены уже совершенных действий.
- Доступ к файлам и директориям рабочего стола.
- Синхронизация с облачным хранилищем.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Страуструп, Б. Язык программирования C++/ Б.Страуструп; специальное издание. Пер. с англ. — СПб.: BHV, 2008. – 1098 с.

[2] Лафоре, Р. Объектно-ориентированное программирование в C++, 4-е издание / Р. Лафоре — СПб.: Питер, 2004.

[3] Официальный сайт документации Qt [Электронный ресурс]. – Режим доступа - https://doc.qt.io/ – Дата доступа: 11.12.2022