Содержимое файла *confirmdelete.h*:

#ifndef CONFIRMDELETE\_H

#define **CONFIRMDELETE\_H**

#include <QDialog>

*namespace* **Ui** {

*class* **ConfirmDelete**;

}

*class* **ConfirmDelete** : *public* QDialog //*класс для подтверждения удаления*

{

Q\_OBJECT

*public*:

*explicit* **ConfirmDelete**(QWidget \***parent** = *nullptr*);

~**ConfirmDelete**();

void **SetConfirm**(bool); *//* *метод* *установки* *подтверждения* *удаления*

bool **Confirm**(); *//метод* *срабатывания* *подтверждения* *удаления*

*private* slots:

void **on\_btnOK\_clicked**(); *//* *слот* *нажатия* *на* *"OK"*

void **on\_btnCancel\_clicked**(); *//* *слот* *нажатия* *на* *"Cancel"*

*private*:

Ui::ConfirmDelete \***ui**;

bool **confirm**=*false*; *//флаг* *установки* *подтверждения*

};

#endif *//* *CONFIRMDELETE\_H*

Содержимое файла *container.h*:

#ifndef CONTAINER\_H

#define **CONTAINER\_H**

#include<QFileSystemModel>

*struct* **Node** *//элемент* *контейнера* *List*

{

QString **info**=""; *//структурный* *атрибут* *для* *хранения* *файловых* *путей*

Node\* **next**=*nullptr*; *//указатель* *на* *следующий* *элемент* *контейнера*

};

*class* **List** *//контейнер* *списочного* *типа*

{

Node\* **node**=*nullptr*; *//указатель* *на* *текущий* *элемент*

Node\* **head**=*nullptr*; *//указатель* *на* *первый* *элемент*

Node\* **tail**=*nullptr*; *//указатель* *на* *последний* *элемент*

*public*:

**List**() = *default*;

~**List**() = *default*;

void **Push**(QString); *//метод* *добавления* *элемента*

Node\* **PeekHead**(bool **flag**); *//метод* *передачи* *первого* *либо* *текущего* *элемента*

void **Pop**(); *//метод* *очистки* *контейнера*

bool **IsEmpty**(); *//метод* *проверки,* *пуст* *ли* *контейнер*

*class* **Iterator** *//вложенный* *итератор* *контейнера* *List*

{

Node \***current**=*nullptr*; *//указатель* *на* *текущий* *элемент*

*public*:

**Iterator**() = *default*;

~**Iterator**()= *default*;

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

bool *operator* ++(); *//перегруженный* *оператор* *для* *перехода* *к* *следующему* *элементу*

Node\* *operator* \*(); *//перегруженный* *оператор* *для* *получения* *информации* *текущего* *элемента*

};

};

#endif *//* *CONTAINER\_H*

Содержимое файла *createchoise.h*:

#ifndef CREATECHOISE\_H

#define **CREATECHOISE\_H**

#include <QDialog>

*namespace* **Ui** {

*class* **CreateChoise**;

}

*class* **CreateChoise** : *public* QDialog *//класс* *для* *выбора* *создаваемого* *объекта*

{

Q\_OBJECT

*public*:

*explicit* **CreateChoise**(QWidget \***parent** = *nullptr*);

~**CreateChoise**();

void **ChooseFile**(bool); *//метод* *установки* *создания* *файла*

void **ChooseFolder**(bool); *//метод* *установки* *создания* *директории*

bool **GetFile**(); *//метод,* *сообщающий* *о* *выборе* *создании* *файла*

bool **GetFolder**(); *//метод,* *сообщающий* *о* *выборе* *создания* *директории*

*private* slots:

void **on\_btnFile\_clicked**(); *//* *слот* *выбора* *о* *создании* *файла*

void **on\_btnFolder\_clicked**(); *//* *слот* *выбора* *о* *создании* *директории*

void **on\_btnCancel\_clicked**(); *//* *слот* *нажатия* *на* *"Cancel"*

*private*:

Ui::CreateChoise \***ui**; *//указатель* *для* *связи* *с* *соответствующим* *ui-файлом*

QString **fileName**=""; *//переменная* *для* *хранения* *имени* *созданного* *файла*

QString **folderName**=""; *//переменная* *для* *хранения* *имени* *созданной* *директории*

bool **file**=0; *//флаг* *выбора* *файла*

bool **folder**=0; *//флаг* *выбора* *директории*

};

#endif *//* *CREATECHOISE\_H*

Содержимое файла *exception.h*:

#ifndef EXCEPTION\_H

#define **EXCEPTION\_H**

#include<QMessageBox> *//предоставляет* *модальный* *диалог* *для* *информирования* *пользователя*

#include<iostream>

*using* *namespace* std;

*class* **Exception** *//базовый* *класс* *исключений*

{

QString **nameOperation**=""; *//название* *операции,* *при* *которой* *обработано* *исключение*

QString **problem**=""; *//суть* *проблемы*

*public*:

**Exception**()=*default*;

~**Exception**()=*default*;

void **SetException**(QString **str1**, QString **str2**) *//создание* *информации,* *предоставленной* *пользователю*

{

nameOperation=str1;

problem=str2;

}

void **GetException**(QWidget\* **parent**)

{

QMessageBox::warning(*parent*, nameOperation, problem); *//предоставление* *пользователю* *информации* *об* *исключении*

}

};

*class* **RootDirectoryException**: *public* Exception *//производный* *класс,* *обрабатывающий* *исключение* *выполнения* *операций* *в* *корневой* *папке*

{

QString **nameOperation**="";

QString **problem**="";

*public*:

**RootDirectoryException**()=*default*;

**RootDirectoryException**(QString **str1**, QString **str2**) : nameOperation(str1), problem(str2)

{

Exception::SetException(nameOperation, problem); *//вызов* *метода* *базового* *класса*

}

~**RootDirectoryException**()=*default*;

};

*class* **ChoiseException**: *public* Exception *//производный* *класс,* *обрабатывающий* *исключение* *выполнения* *операции* *без* *выбранного* *объекта*

{

QString **nameOperation**="";

QString **problem**="";

*public*:

**ChoiseException**()=*default*;

**ChoiseException**(QString **str1**, QString **str2**): nameOperation(str1), problem(str2)

{

Exception::SetException(nameOperation, problem); *//вызов* *метода* *базового* *класса*

}

~**ChoiseException**()=*default*;

};

*class* **PerformationException**: *public* Exception *//производный* *класс,* *обрабатывающий* *исключение* *невыполненной* *операции*

{

QString **nameOperation**="";

QString **problem**="";

*public*:

**PerformationException**()=*default*;

**PerformationException**(QString **str1**, QString **str2**): nameOperation(str1), problem(str2)

{

Exception::SetException(nameOperation, problem); *//вызов* *метода* *базового* *класса*

}

~**PerformationException**()=*default*;

};

*class* **BadAllocException**:*public* Exception *//производный* *класс,* *обрабатывающий* *исключение* *выделения* *памяти*

{

QString **nameOperation**="";

QString **problem**="";

*public*:

**BadAllocException**()=*default*;

**BadAllocException**(QString **str1**, QString **str2**): nameOperation(str1), problem(str2)

{

Exception::SetException(nameOperation, problem); *//вызов* *метода* *базового* *класса*

}

~**BadAllocException**()=*default*;

};

*class* **ExceptionEmpty**: *public* Exception *//производный* *класс,* *обрабатывающий* *исключение* *пустой* *строки* *для* *ввода*

{

QString **nameOperation**="";

QString **problem**="";

*public*:

**ExceptionEmpty**()=*default*;

**ExceptionEmpty**(QString **str1**, QString **str2**): nameOperation(str1), problem(str2)

{

Exception::SetException(nameOperation, problem); *//вызов* *метода* *базового* *класса*

}

~**ExceptionEmpty**()=*default*;

};

#endif *//* *EXCEPTION\_H*

Содержимое файла *mainwindow.h*:

#ifndef MAINWINDOW\_H

#define **MAINWINDOW\_H**

#include<QMainWindow> *//предоставляет* *главное* *окно* *приложения*

#include<QClipboard> *//обеспечивает* *доступ* *к* *системному* *буферу* *обмена* *окна*

#include ".h/searchwindow.h"

#include ".h/systemfiles.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow *//класс* *главного* *окна* *приложения*

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \***parent** = *nullptr*);

~**MainWindow**();

*private* slots:

void **on\_lvSource\_doubleClicked**(*const* QModelIndex &**index**); *//слот* *двойного* *нажатия* *на* *панель"lvSource"*

void **on\_lvSource\_clicked**(*const* QModelIndex &**index**); *//слот* *нажатия* *на* *панель"lvSource"*

void **on\_btnCreate\_clicked**(); *//слот* *нажатия* *на* *кнопку* *"Сreate"*

void **on\_btnDelete\_clicked**(); *//слот* *нажатия* *на* *кнопку* *"Delete"*

void **on\_btnCopy\_clicked**(); *//слот* *нажатия* *на* *кнопку* *"Сopy"*

void **on\_btnReplace\_clicked**(); *//слот* *нажатия* *на* *кнопку* *"Replace"*

void **on\_btnRename\_clicked**(); *//слот* *нажатия* *на* *кнопку* *"Rename"*

void **on\_btnSearch\_clicked**(); *//слот* *нажатия* *на* *кнопку* *"Search"*

void **on\_lineSearch\_textEdited**(*const* QString &**arg1**); *//слот* *ввода* *имени* *для* *поиска*

void **on\_listWidget\_currentTextChanged**(*const* QString &**currentText**); *//слот* *копирования* *текущего* *местоположения* *в* *буфер* *обмена*

*private*:

Ui::MainWindow \***ui**; *//указатель* *на* *объект* *соотв* *типа* *в* *классе* *основной* *формы*

QFileSystemModel \***model**; *//указатель* *для* *связи* *с* *моделью* *данных* *для* *файловой* *системы*

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

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

System \***file**=&f; *//указатель* *на* *объект* *класса* *System* *для* *получения* *адреса* *объекта* *класса* *File*

System \***folder**=&d; *//указатель* *на* *объект* *класса* *System* *для* *получения* *адреса* *объекта* *класса* *Folder*

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

QString **filePath**=""; *//переменная* *для* *хранения* *пути* *выбранного* *файла*

QString **fileName**=""; *//переменная* *для* *хранения* *имени* *выбранного* *файла*

QString **dirPath**=""; *//переменная* *для* *хранения* *пути* *выбранной* *директории*

QString **dirName**=""; *//переменная* *для* *хранения* *имени* *выбранной* *директории*

};

bool **RecursiveDelete**(QDir&, System \***f**, System \***d**); *//функция* *рекурсивного* *удаления* *содержимого* *выбранной* *папки*

void **RecursiveCopyList**(QDir &,QFileInfoList &); *//функция* *рекурсивного* *наполнения* *содержимым* *списка* *для* *копирования*

#endif *//* *MAINWINDOW\_H*

Содержимое файла *searchwindow.h*:

#ifndef SEARCHWINDOW\_H

#define **SEARCHWINDOW\_H**

#include ".h/container.h"

#include <QDialog>

#include <QDir>

#include <QListWidget> *//предоставляет* *виджет* *списка* *на* *основе* *элементов*

*namespace* **Ui** {

*class* **SearchWindow**;

}

*class* **SearchWindow** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **SearchWindow**(QWidget \***parent** = *nullptr*);

~**SearchWindow**();

void **SetName**(*const* QString); *//метод* *получения* *имени* *для* *поиска*

void **Search**(QDir &); *//метод* *поиска* *по* *имени*

void **SetUi**(); *//метод* *передачи* *результатов* *поиска* *для* *отображения*

void **ResetUi**(); *//метод* *очистки* *окна* *отображения* *результатов* *поиска*

*private* slots:

void **on\_btnOK\_clicked**(); *//* *слот* *нажатия* *на* *"OK"*

*private*:

Ui::SearchWindow \***ui**; *//указатель* *на* *объект* *соотв* *типа* *в* *классе* *основной* *формы*

QString **searchName**=""; *//переменная* *для* *хранения* *имени* *для* *поиска*

List **list**; *//контейнер* *для* *хранения* *пктей* *файлов,* *имена* *которых* *совпали* *с* *введенным*

List :: Iterator **iter**; *//итератор* *контейнера* *для* *управления* *его* *элементами*

};

#endif *//* *SEARCHWINDOW\_H*

Содержимое файла *systemfiles.h*:

#ifndef OOP\_H

#define **OOP\_H**

#include<QFileSystemModel>

#include<QDir>

#include<QString>

#include<iostream>

#include<io.h>

#include<fstream>

#include<direct.h>

#include<cstdio>

*using* *namespace* std;

*class* **System** *//базовый* *класс* *Sytem* *для* *работы* *с* *системными* *объектами*

{

*public*:

**System**()=*default*;

*virtual* ~**System**()=*default*;

*//ниже* *представлены* *чисто* *виртуальные* *методы* *для* *работы* *с* *системными* *объектами*

*//описаны* *они* *будут* *в* *производных* *классах*

*virtual* bool ***Create***()=0;

*virtual* bool ***Delete***()=0;

*virtual* bool ***Rename***(QString)=0;

*virtual* bool ***Copy***(QString)=0;

*virtual* void ***SetPath***(QString)=0;

*virtual* *const* char\* ***GetPath***()=0;

};

*class* **File**: *public* System *//производный* *класс* *File* *для* *работы* *с* *текстовыми* *файлами*

{

*const* char\* **filePath**="";

*public*:

**File**()=*default*;

~**File**()=*default*;

bool ***Create***(); *//метод* *создания* *текстового* *файла*

bool ***Delete***(); *//метод* *удаления* *текстового* *файла*

bool ***Rename***(QString); *//метод* *переименования* *текстового* *файла*

bool ***Copy***(QString); *//метод* *копирования* *текстового* *файла*

void ***SetPath***(QString); *//метод* *установки* *пути* *текстового* *файла*

*const* char\* ***GetPath***(); *//метод* *передачи* *пути* *текстового* *файла*

};

*class* **Folder**: *public* System *//производный* *класс* *Folder* *для* *работы* *с* *директориями*

{

*const* char\* **dirPath**="";

*public*:

**Folder**()=*default*;

~**Folder**()=*default*;

bool ***Create***(); *//метод* *создания* *директории*

bool ***Delete***(); *//метод* *удаления* *директории*

bool ***Rename***(QString); *//метод* *переименования* *директории*

bool ***Copy***(QString); *//метод* *копирования* *директории*

void ***SetPath***(QString); *//метод* *установки* *пути* *директории*

*const* char\* ***GetPath***(); *//метод* *передачи* *пути* *директории*

};

#endif *//* *OOP\_H*

Содержимое файла *additionalwindow.cpp*:

#include ".h/additionalwindow.h"

#include ".h/exception.h"

#include "ui\_additionalwindow.h"

#include<QMessageBox>

AdditionalWindow::**AdditionalWindow**(QWidget \***parent**) : QDialog(*parent*), ui(*new* Ui::AdditionalWindow)

{

ui->setupUi(*this*); *//настраивает* *пользовательский* *интерфейс* *для* *указанного* *виджета*

setWindowTitle("FileManager"); *//установка* *имени* *окна*

model = *new* QFileSystemModel(*this*); *//* *выделение* *памяти* *под* *указатель*

model->setFilter(QDir::QDir::*AllEntries*); *//* *метод* *позволяет* *отображать* *некоторые* *элементы* *файловой* *системы(в* *нашем* *случае* *все)*

model->setRootPath(""); *//* *метод* *позволяет* *определить* *место* *в* *системе* *для* *отслеживания* *изменений(указана* *корневая* *папка)*

*//система* *представлена* *в* *виде* *структурЫ* *типа* *List*

ui->listView->*setModel*(*model*); *//установка* *новой* *модели* *выбора*

}

AdditionalWindow::~**AdditionalWindow**()

{

*delete* ui;

*delete* model;

}

void AdditionalWindow::**SetQDir**(QDir &**Dir**) *//метод* *получения* *переменной* *директории*

{

dir=Dir;

}

QDir AdditionalWindow::**GetQDir**() *//метод* *передачи* *переменной* *директории*

{

*return* dir;

}

void AdditionalWindow::**SetCancel**(bool **i**) *//метод* *установки* *отмены*

{

cancel=i;

}

bool AdditionalWindow::**Cancel**() *//метод* *срабатывания* *отмены*

{

*return* cancel;

}

void AdditionalWindow::**on\_btnOK\_clicked**() *//* *слот* *нажатия* *на* *"OK"*

{

QDir **dir**=QDir(model->filePath(ui->listView->rootIndex())); *//получение* *текущей* *директории*

*try* {

*if*(dir.absolutePath()=="D:/Qt/Projects/build-QWERTY-Desktop\_Qt\_6\_4\_0\_MinGW\_64\_bit-Debug") *//если* *это* *корневая* *директория*

*throw* RootDirectoryException("", "You are in a root directory! Please choose an another directory");

*else*

{

SetCancel(*false*); *//установка* *подтверждения*

SetQDir(*dir*); *//метод* *получения* *переменной* *директории*

hide(); *//метод* *закрытия* *окна*

}

}

*catch*(RootDirectoryException **error**)

{

error.GetException(*this*);

}

*catch* (...)

{

QMessageBox::warning(*this*, "", "Unknown error! Please try again!");

}

}

void AdditionalWindow::**on\_btnCancel\_clicked**() *//* *слот* *нажатия* *на* *"Cancel"*

{

hide(); *//метод* *закрытия* *окна*

}

void AdditionalWindow::**on\_listView\_doubleClicked**(*const* QModelIndex &**index**) *//* *слот* *нажатия* *на* *панель* *"listView"*

{

QListView\* **listView** = (QListView\*) sender();*//получение* *указателя* *на* *обЪект* *который* *принял* *сигнал*

*//приведение* *объекта* *источника* *методом* *sender()* *к* *типу* *listview*

QFileInfo **info** = model->fileInfo(index); *//получение* *пути* *элемента,* *который* *соответствует* *этому* *индексу*

*//* *в* *зависимости* *от* *того,* *что* *это* *за* *элемент,* *алгоритм* *дальнейших* *дествий* *ветвится*

*if*(info.fileName()=="..")

{

QDir **dir** = info.dir();*//получение* *объекта* *класса* *QDir*

dir.cd("..");*//dir.cdUp();* *навигация.* *в* *данном* *случае* *переход* *в* *родительскую* *папку*

listView->*setRootIndex*(model->index(dir.absolutePath()));*//* *получение* *индекса* *по* *пути*

*//чтобы* *показать* *эелемент,* *полученный* *через* *индекс(listView->setRootIndex)*

*//так* *работает* *списочное* *представление* *(в* *конкретный* *момент* *времени* *показано* *содержимое* *одной* *папки)*

}

*else* *if*(info.fileName()==".")

{

listView->*setRootIndex*(model->index(""));*//переход* *в* *корневую* *папку*

*//показать* *корневую* *папку*

}

*else* *if* (info.isDir())*//* *если* *выбранный* *элемент* *-* *директория*

{

listView->*setRootIndex*(index);*//элемент* *с* *этим* *индексом* *становится* *корневым*

}

}

Содержимое файла *confirmdelete.cpp*:

#include ".h/confirmdelete.h"

#include "ui\_confirmdelete.h"

ConfirmDelete::**ConfirmDelete**(QWidget \***parent**) : QDialog(*parent*), ui(*new* Ui::ConfirmDelete)

{

ui->setupUi(*this*); *//настраивает* *пользовательский* *интерфейс* *для* *указанного* *виджета*

setWindowTitle("Delete"); *//установка* *имени* *окна*

}

ConfirmDelete::~**ConfirmDelete**()

{

*delete* ui;

}

void ConfirmDelete::**SetConfirm**(bool **i**) *//* *метод* *установки* *подтверждения* *удаления*

{

confirm=i;

}

bool ConfirmDelete::**Confirm**() *//метод* *срабатывания* *подтверждения* *удаления*

{

*return* confirm;

}

void ConfirmDelete::**on\_btnOK\_clicked**() *//* *слот* *нажатия* *на* *"OK"*

{

SetConfirm(*true*);

hide(); *//метод* *закрытия* *окна*

}

void ConfirmDelete::**on\_btnCancel\_clicked**() *//* *слот* *нажатия* *на* *"Cancel"*

{

hide();

}

Содержимое файла *container.cpp:*

#include ".h/container.h"

#include ".h/exception.h"

void List::**Push**(QString **bolt**) *//метод* *добавления* *элемента*

{

*try* {

*if*(!(node=*new* Node)) *//если* *память* *не* *выделилась*

*throw* BadAllocException("", "");

}

*catch*(BadAllocException **error**)

{

qDebug()<<"Node of List was not create!\n";

}

*catch* (...)

{

qDebug()<<"Unknown Error!\n";

}

node->info=bolt; *//наполнение* *информацией* *атрибут* *структуры*

*if* (!head||!tail) *//если* *контейнер* *пуст*

{

*//создание* *первого* *элемента*

tail = node;

head = node;

}

*else*

{

*//добавление* *нового* *элемента*

tail->next=node;

tail=node;

}

}

Node\* List::**PeekHead**(bool **flag**) *//метод* *передачи* *первого* *либо* *текущего* *элемента*

{

*return* flag?head:node;

}

void List::**Pop**() *//метод* *очистки* *контейнера*

{

*while*(head) *//пока* *первый* *элемент* *есть*

{

node = head->next; *//установка* *текущего* *на* *следующий* *элемент* *за* *первым*

*delete* head; *//удаление* *первого* *элемента*

head = node; *//следующий* *за* *первым* *становится* *первым*

}

tail = node; *//установка* *последнего* *элемента* *на* *пустой* *указатель*

}

bool List::**IsEmpty**() *//метод* *проверки,* *пуст* *ли* *контейнер*

{

*return* tail==*nullptr*;

}

void List::Iterator::**SetIter**(Node\* **node**) *//метод* *установки* *итератора* *на* *элемент,* *связанный* *с* *указателем* *\*node*

{

current=node;

}

bool List::Iterator::*operator* ++() *//перегруженный* *оператор* *для* *С*

{

*if*(*this*->current->next) *//если* *указатель* *следующего* *элемента* *не* *пустой*

{

*this*->current=*this*->current->next; *//переход* *к* *следующему* *элемненту*

*return* *true*;

}

*return* *false*;

}

Node\* List::Iterator::*operator* \*() *//перегруженный* *оператор* *для* *получения* *информации* *текущего* *элемента*

{

*return* *this*->current;

}

Содержимое файла *createchoise.cpp:*

#include ".h/createchoise.h"

#include "ui\_createchoise.h"

CreateChoise::**CreateChoise**(QWidget \***parent**) : QDialog(*parent*), ui(*new* Ui::CreateChoise)

{

ui->setupUi(*this*); *//настраивает* *пользовательский* *интерфейс* *для* *указанного* *виджета*

setWindowTitle("Create"); *//установка* *имени* *окна*

}

CreateChoise::~**CreateChoise**()

{

*delete* ui;

}

void CreateChoise::**on\_btnFile\_clicked**() *//* *слот* *выбора* *о* *создании* *файла*

{

ChooseFile(*true*);

hide();

}

void CreateChoise::**on\_btnFolder\_clicked**() *//* *слот* *выбора* *о* *создании* *директории*

{

ChooseFolder(*true*);

hide();

}

void CreateChoise::**ChooseFile**(bool **i**) *//метод* *установки* *создания* *файла*

{

file=i;

}

void CreateChoise::**ChooseFolder**(bool **i**) *//метод* *установки* *создания* *директории*

{

folder=i;

}

bool CreateChoise::**GetFile**() *//метод,* *сообщающий* *о* *выборе* *создании* *файла*

{

*return* file;

}

bool CreateChoise::**GetFolder**() *//метод,* *сообщающий* *о* *выборе* *создания* *директории*

{

*return* folder;

}

void CreateChoise::**on\_btnCancel\_clicked**() *//* *слот* *нажатия* *на* *"Cancel"*

{

hide();

}

Содержимое файла *mainwindow.cpp:*

#include ".h/additionalwindow.h"

#include ".h/confirmdelete.h"

#include ".h/createchoise.h"

#include ".h/exception.h"

#include ".h/mainwindow.h"

#include ".h/renamewindow.h"

#include "ui\_mainwindow.h"

#include<QMessageBox>

MainWindow::**MainWindow**(QWidget \***parent**) : QMainWindow(*parent*), ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*); *//настраивает* *пользовательский* *интерфейс* *для* *указанного* *виджета*

setWindowTitle("FileManager"); *//установка* *имени* *главного* *окна*

*try* {

*if*(!(model = *new* QFileSystemModel(*this*))) *//* *выделение* *памяти* *под* *указатель* *на* *объект* *этого* *класса*

*throw* BadAllocException("Memory allocation", "Model of QFileSystemModel was not created!");

}

*catch*(BadAllocException **error**)

{

error.GetException(*this*);

}

*catch* (...)

{

QMessageBox::warning(*this*, "Memory allocation", "Unknown error! Please try again!");

}

model->setFilter(QDir::QDir::*AllEntries*); *//* *метод* *позволяет* *отображать* *некоторые* *элементы* *файловой* *системы(в* *нашем* *случае* *все)*

model->setRootPath(""); *//* *метод* *позволяет* *определить* *место* *в* *системе* *для* *отслеживания* *изменений(указана* *корневая* *папка)*

*//система* *представлена* *в* *виде* *структурЫ* *типа* *List*

ui->lvSource->*setModel*(*model*); *//назначение* *\*model* *объектом* *представления* *панели* *Source*

*//устанока* *информации* *для* *подсказок*

ui->btnCreate->setToolTip("Create");

ui->btnDelete->setToolTip("Delete");

ui->btnCopy->setToolTip("Copy");

ui->btnReplace->setToolTip("Replace");

ui->btnRename->setToolTip("Rename");

ui->lineSearch->setToolTip("Please don't forget about the extension, if you want to enter a file name");

}

MainWindow::~**MainWindow**()

{

*delete* ui;

*delete* model;

}

void MainWindow::**on\_lvSource\_clicked**(*const* QModelIndex &**index**)

{

*//получение* *указателя* *на* *обЪект* *который* *принял* *сигнал*

*//приведение* *объекта* *источника* *методом* *sender()* *к* *типу* *listview*

QFileInfo **info** = model->fileInfo(index); *//получение* *пути* *элемента,* *который* *соответствует* *этому* *индексу*

*//* *в* *зависимости* *от* *того,* *что* *это* *за* *элемент,* *алгоритм* *дальнейших* *дествий* *ветвится*

QString **format** = "dddd, d MMMM yy hh:mm:ss"; *//формат* *вывода* *даты* *последнего* *изменения*

ui->label\_3->setText(info.lastModified().toString(format)); *//вывод* *даты* *последнего* *изменения*

*if*(info.isFile()) *//выбранный* *объект* *-* *файл*

{

QString **sizeFile**=""; *//переменная* *размера* *файла*

*if*(info.size()<1024) *//если* *размер* *файла* *меньше* *1* *килобайта*

sizeFile=sizeFile.append(QString::number(info.size())).append(" B"); *//приведение* *размера* *к* *виду* *"B"*

*else* *if*(info.size()>=1024 && info.size()<1048576) *//если* *размер* *файла* *больше* *1* *килобайта* *и* *меньше* *1* *мегабайта*

{

double **d**=info.size()/1024.;

sizeFile=sizeFile.append(QString::number(d)).append(" KB"); *//приведение* *размера* *к* *виду* *"KB"*

}

*else* *if*(info.size()>=1048576 && info.size()<1073741824) *//если* *размер* *файла* *больше* *1* *мегабайта* *и* *меньше* *1* *гигабайта*

{

double **d**=info.size()/1048576.;

sizeFile=sizeFile.append(QString::number(d)).append(" MB"); *//приведение* *размера* *к* *виду* *"MB"*

}

*else* *if*(info.size()>=1073741824) *//если* *размер* *файла* *больше* *1* *гигабайта*

{

double **d**=info.size()/1073741824.;

sizeFile=sizeFile.append(QString::number(d)).append(" GB"); *//приведение* *размера* *к* *виду* *"GB"*

}

QString **size**="Size : ";

QString **allSize**=size.append(sizeFile); *//строка* *для* *отображения* *размера* *файла*

ui->label\_4->setText(allSize); *//отображениe* *размера* *файла*

QString **type**="Type : ";

QString **typeInfo**=type.append(info.suffix()).append("-file"); *//строка* *для* *отображения* *типа* *файла*

ui->label\_5->setText(typeInfo); *//отображениe* *типа* *файла*

fileName = model->fileName(index); *//переменная* *для* *хранения* *имени* *выбранного* *файла*

filePath = model->filePath(index); *//переменная* *для* *хранения* *пути* *выбранного* *файла*

file->*SetPath*(filePath); *//установка* *пути* *выбранного* *файла*

*if*(dirPath!="") *//если* *до* *этого* *была* *выбрана* *директория*

{

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

}

*if*(info.isDir()) *//если* *выбранный* *объект* *-* *директория*

{

ui->label\_5->setText("Type: System directory"); *//отображение* *типа* *объекта*

dirName=info.fileName(); *//переменная* *для* *хранения* *имени* *выбранной* *директории*

dirPath=info.absoluteFilePath(); *//переменная* *для* *хранения* *пути* *выбранной* *директории*

*if*(filePath!="") *//если* *до* *этого* *был* *выбран* *файл*

{

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

}

}

*//слот* *обработки* *двойного* *клика* *мыши* *панели* *lvSource(переход* *к* *нажимаемому* *элементу)*

void MainWindow::**on\_lvSource\_doubleClicked**(*const* QModelIndex &**index**)

{

QListView\* **listView** = (QListView\*) sender();*//получение* *указателя* *на* *обЪект* *который* *принял* *сигнал*

*//приведение* *объекта* *источника* *методом* *sender()* *к* *типу* *listview*

*//Возвращает* *указатель* *на* *объект,* *отправивший* *сигнал,* *если* *он* *вызван* *в* *слоте,* *активированном* *сигналом;* *в* *противном* *случае* *он* *возвращает* *nullptr.*

QFileInfo **fileInfo** = model->fileInfo(index); *//получение* *информации* *элемента,* *который* *соответствует* *этому* *индексу*

*//* *в* *зависимости* *от* *того,* *что* *это* *за* *элемент,* *алгоритм* *дальнейших* *дествий* *ветвится*

*if*(fileInfo.fileName()=="..") *//если* *выбран* *выход* *из* *текущей* *папки*

{

QDir **dir** = fileInfo.dir();*//получение* *объекта* *класса* *QDir*

*if*(filePath!="") *//если* *до* *этого* *был* *выбран* *файл*

{

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

dir.cd("..");*//dir.cdUp();* *навигация.* *в* *данном* *случае* *переход* *в* *родительскую* *папку*

listView->*setRootIndex*(model->index(dir.absolutePath()));*//* *получение* *индекса* *по* *пути*

*//чтобы* *показать* *эелемент,* *полученный* *через* *индекс(listView->setRootIndex)*

*//так* *работает* *списочное* *представление* *(в* *конкретный* *момент* *времени* *показано* *содержимое* *одной* *папки)*

ui->listWidget->clear(); *//очистка* *панели* *"listWidget"*

ui->listWidget->addItem(fileInfo.absoluteFilePath()); *//отображение* *нового* *пути* *в* *панели* *"listWidget"*

*if*(dirPath!="") *//если* *до* *этого* *выбрана* *директория*

{

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

}

*else* *if*(fileInfo.fileName()==".") *//если* *выбран* *выход* *в* *корневую* *папку*

{

*if*(filePath!="") *//если* *до* *этого* *был* *выбран* *файл*

{

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

listView->*setRootIndex*(model->index(""));*//переход* *в* *корневую* *папку*

*//показать* *корневую* *папку*

ui->listWidget->clear(); *//очистка* *панели* *"listWidget"*

ui->listWidget->addItem(""); *//отображение* *нового* *пути* *в* *панели* *"listWidget"*

*if*(dirPath!="") *//если* *до* *этого* *выбрана* *директория*

{

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

}

*else* *if* (fileInfo.isDir()) *//* *если* *выбранный* *элемент* *-* *директория*

{

*if*(filePath!="") *//если* *до* *этого* *был* *выбран* *файл*

{

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

listView->*setRootIndex*(index);*//элемент* *с* *этим* *индексом* *становится* *корневым*

ui->listWidget->clear(); *//очистка* *панели* *"listWidget"*

ui->listWidget->addItem(fileInfo.absoluteFilePath()); *//отображение* *нового* *пути* *в* *панели* *"listWidget"*

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

*else* *if*(fileInfo.isFile()) *//* *если* *выбранный* *элемент* *-* *файл*

{

*if*(dirPath!="") *//если* *до* *этого* *выбрана* *директория*

{

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

QDesktopServices::openUrl(QUrl::fromUserInput(filePath)); *//открывает* *файл* *в* *файловой* *системе* *Windows*

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

}

void MainWindow::**on\_btnCreate\_clicked**() *//слот* *нажатия* *на* *кнопку* *"Сreate"*

{

QDir **dir**=QDir(model->filePath(ui->lvSource->rootIndex())); *//получение* *текущей* *директории*

*try* {

*if*(dir.absolutePath()=="D:/Qt/Projects/build-QWERTY-Desktop\_Qt\_6\_4\_0\_MinGW\_64\_bit-Debug") *//если* *это* *корневая* *директория*

*throw* RootDirectoryException("Create", "You are in a root directory! Please choose an another directory");

*else*

{

CreateChoise **window**;

window.*exec*(); *//метод* *выполняет* *появление* *окна* *для* *выбора* *типа* *создаваемого* *объекта*

*if*(window.GetFile()) *//если* *был* *выбран* *файл*

{

RenameWindow **isFile**;

isFile.*exec*(); *//выполняет* *появление* *окна* *для* *создания* *имени*

QString **createPath** = dir.absolutePath().append("/").append(isFile.GetName()); *//получение* *пути* *соозданного* *файла*

bool **fileExists**=*false*; *//флаг* *существования* *файлов* *с* *таким* *именем*

*//цикл* *прохода* *по* *текущей* *директории* *для* *поиска* *файлов* *с* *таким* *именем*

*foreach*(QFileInfo **files**, dir.entryInfoList(QDir::*Files*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(files.fileName()==isFile.GetName()) *//если* *файл* *с* *таким* *именем* *найден*

{

fileExists=*true*; *//флаг* *существования* *файла* *принимает* *истинное* *значение*

*break*;

}

}

*if*(!fileExists) *//если* *файлов* *с* *таким* *именем* *нет*

{

*if*(isFile.GetName().contains(".txt")) *//если* *это* *текстовый* *файл*

{

file->*SetPath*(createPath); *//установка* *пути* *файла*

*if*(!file->*Create*()) *//если* *файл* *не* *создан*

*throw* PerformationException("Create File", "The operation <<Create>> was not perfomed!");

}

*else*

{

*//создание* *нетекстового* *файла*

QFile **file**(createPath);

*if*(!file.*open*(QIODeviceBase::*WriteOnly*)) *//если* *файл* *не* *открыт*

*throw* PerformationException("Create File", "The operation <<Create>> was not perfomed!");

*else* file.*close*();

}

}

*else* *throw* PerformationException("Create File", "A file with this name exists!");

}

*if*(window.GetFolder()) *//если* *выбранный* *объект* *-* *директория*

{

RenameWindow **isDir**;

isDir.*exec*(); *//выполняет* *появление* *окна* *для* *создания* *имени*

QString **createPath** = dir.absolutePath().append("/").append(isDir.GetName()); *//получение* *пути* *сооздаваемой* *директории*

bool **dirExists**=*false*;*//флаг* *существования* *файлов* *с* *таким* *именем*

*//цикл* *прохода* *по* *текущей* *директории* *для* *поиска* *директории* *с* *таким* *именем*

foreach(QFileInfo **dirs**, dir.entryInfoList(QDir::*Dirs*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(dirs.fileName()==isDir.GetName()) *//если* *директория* *с* *таким* *именем* *найдена*

{

dirExists=*true*; *//флаг* *существования* *директории* *принимает* *истинное* *значение*

*break*;

}

}

*if*(!dirExists) *//если* *директорий* *с* *таким* *именем* *нет*

{

folder->*SetPath*(createPath); *//установка* *пути* *директории*

*if*(!folder->*Create*()) *//если* *директория* *не* *создана*

*throw* PerformationException( "Create Folder", "The operation <<Create>> was not perfomed!");

}

*else* *throw* PerformationException( "Create Folder", "A directory with this name exist!");

}

}

}

*catch*(RootDirectoryException **error**)

{

error.GetException(*this*);

}

*catch*(PerformationException **error**)

{

error.GetException(*this*);

}

*catch* (...)

{

QMessageBox::warning(*this*, "Create", "Unknown error! Please try again!");

}

}

bool **RecursiveDelete**(QDir &**dir**, System\* **file**, System\* **folder**) *//функция* *рекурсивного* *удаления* *содержимого* *выбранной* *папки*

{

*//цикл* *прохода* *по* *текущей* *директории* *для* *удаления* *файлов* *и* *директорий* *внутри*

foreach(QFileInfo **info**, dir.entryInfoList(QDir::*Files*|QDir::*Dirs*|QDir::*NoDotAndDotDot*, QDir::*Name*|QDir::*DirsFirst*))

{

*if*(info.isDir())*//если* *директория*

{

dir.cd(info.fileName());*//заходим* *в* *нее*

RecursiveDelete(*dir*, file, folder);*//* *рекурсивно* *удаляем* *внутренности*

*//теперь* *папка* *пуста* *и* *мы* *можем* *ее* *удалить*

folder->*SetPath*(dir.absolutePath());

*if*(!folder->*Delete*())

*return* *false*;

folder->*SetPath*("");

dir.cdUp();*//возврат*

}

*else* *if*(info.isFile()) *//если* *текущий* *объект* *-* *файл*

{

*if*(info.absoluteFilePath().contains(".txt")) *//* *если* *файл* *текстовый*

{

file->*SetPath*(info.absoluteFilePath()); *//установка* *пути* *файла*

*if*(!file->*Delete*()) *//если* *файл* *не* *удален*

*return* *false*;

*//очистка* *пути* *файла*

file->*SetPath*("");

}

*else*

{

*if*(!(QFile::remove(info.absoluteFilePath()))) *//если* *файл* *не* *удален*

*return* *false*;

}

}

}

*return* *true*;

}

void MainWindow::**on\_btnDelete\_clicked**() *//слот* *нажатия* *на* *кнопку* *"Delete"*

{

QDir **dir**=QDir(model->filePath(ui->lvSource->rootIndex())); *//получение* *текущей* *директории*

*try* {

*if*(dir.absolutePath()=="D:/Qt/Projects/build-QWERTY-Desktop\_Qt\_6\_4\_0\_MinGW\_64\_bit-Debug") *//если* *это* *корневая* *директория*

*throw* RootDirectoryException("Create", "You are in a root directory! Please choose an another directory");

*else*

{

*if*(filePath=="" && dirPath=="") *//если* *не* *выбран* *ни* *один* *объект*

*throw* ChoiseException( "Delete", "You was not choose a file or a directory! Please try again");

*else* *if*(filePath!="" && dirPath=="") *//если* *выбран* *файл*

{

ConfirmDelete **window**;

window.*exec*(); *//метод* *выполняет* *появление* *окна* *для* *подтверждения* *удаления*

*if*(!window.Confirm()) *//если* *операция* *отменена*

*throw* PerformationException( "Delete FIle", "The operation was canceled!");

*else*

{

*if*(fileName.contains(".txt")) *//если* *удалить* *текстовый* *файл*

{

file->*SetPath*(filePath); *//установка* *пути* *файла*

*if*(!file->*Delete*()) *//если* *удаление* *не* *выпонено*

*throw* PerformationException( "Delete File", "The operation <<Delete>> was not perfomed!");

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

*else*

{

*if*(!(QFile::remove(filePath))) *//если* *удаление* *не* *выпонено*

*throw* PerformationException( "Delete File", "The operation <<Delete>> was not perfomed!");

filePath=fileName=""; *//очистка* *пути* *файла*

}

}

}

*else* *if*(dirPath!="" && filePath=="") *//если* *выбрана* *директория*

{

ConfirmDelete **window**;

window.*exec*(); *//метод* *выполняет* *появление* *окна* *для* *подтверждения* *удаления*

*if*(!window.Confirm()) *//если* *операция* *отменена*

*throw* PerformationException( "Delete FIle", "The operation was canceled!");

*else*

{

QDir **dir**=QDir(dirPath); *//получение* *выбранной* *директории*

*if*(!dir.isEmpty()) *//если* *директория* *не* *пуста*

{

*if*(!RecursiveDelete(*dir*, *file*, *folder*)) *//если* *внутренние* *файлы* *не* *удалены*

*throw* PerformationException( "Delete Folder", "The operation <<Delete>> was not perfomed!");

}

*if*(dir.isEmpty()) *//если* *директория* *пуста*

{

*//установка* *пути* *директории*

folder->*SetPath*(dirPath);

*if*(!folder->*Delete*()) *//если* *удаление* *не* *выпонено*

*throw* PerformationException( "Delete Folder", "The operation <<Delete>> was not perfomed!");

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

}

}

}

}

*catch*(RootDirectoryException **error**)

{

error.GetException(*this*);

}

*catch*(ChoiseException **error**)

{

error.GetException(*this*);

}

*catch*(PerformationException **error**)

{

error.GetException(*this*);

}

*catch*(...)

{

QMessageBox::warning(*this*, "Delete", "Unknown error! Please try again!");

}

}

void **RecursiveCopyList**(QDir &**dir**,QFileInfoList &**copyList**) *//функция* *рекурсивного* *наполнения* *содержимым* *списка* *для* *копирования*

{

*//цикл* *прохода* *по* *текущей* *директории* *для* *создания* *контейнера* *с* *файлами* *и* *директориями* *внутри*

foreach(QFileInfo **info**, dir.entryInfoList(QDir::*Files*|QDir::*Dirs*|QDir::*NoDotAndDotDot*, QDir::*Name*|QDir::*DirsFirst*))

{

copyList.append(info); *//добавление* *элемента* *в* *контейнер*

*if*(info.isDir()) *//* *элемент* *-* *директория*

{

dir.cd(info.fileName());*//заходим* *в* *нее*

RecursiveCopyList(*dir*, *copyList*);*//* *рекурсивно* *копируем* *содержимое*

dir.cdUp();*//возврат*

}

}

}

void MainWindow::**on\_btnCopy\_clicked**() *//слот* *нажатия* *на* *кнопку* *"Сopy"*

{

QDir **dir**=QDir(dirPath); *//получение* *выбранной* *директории*

*try* {

*if*(fileName=="" && dirPath=="") *//если* *не* *выбран* *ни* *один* *объект*

*throw* ChoiseException( "Copy", "You was not choose a file or a directory! Please try again");

*else* *if*(fileName!="" && dirName=="") *//если* *выбран* *файл*

{

AdditionalWindow **window**;

window.*exec*(); *//метод* *выполняет* *появление* *дополнительного* *окна* *для* *копирования*

*if*(window.Cancel()) *//если* *операция* *отменена*

*throw* PerformationException( "Copy File", "The operation was canceled!");

*else*

{

QString **copyPath** = window.GetQDir().absolutePath().append("/").append(fileName); *//создание* *пути* *для* *копирования*

bool **fileExists**=*false*; *//флаг* *существования* *файла* *с* *таким* *именем*

*//цикл* *прохода* *по* *текущей* *директории* *для* *поиска* *файлов* *с* *таким* *именем*

foreach(QFileInfo **files**, window.GetQDir().entryInfoList(QDir::*Files*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(files.fileName()==fileName) *//если* *файл* *с* *таким* *именем* *есть*

{

fileExists=*true*; *//установка* *флаг* *на* *истинное* *значение*

*break*;

}

}

*if*(!fileExists) *//если* *файлов* *с* *таким* *именем* *нет*

{

*if*(fileName.contains(".txt")) *//если* *файл* *текстовый*

{

file->*SetPath*(filePath); *//установка* *пути* *файла*

*if*(!file->*Copy*(copyPath)) *//если* *копирование* *не* *произошло*

*throw* PerformationException( "Copy File", "The operation <<Copy>> was not perfomed!");

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

*else*

{

*if*(!(QFile::copy(filePath, copyPath))) *//если* *копировагние* *не* *произошло*

*throw* PerformationException( "Copy File", "The operation <<Copy>> was not perfomed!");

}

}

*else* *throw* PerformationException( "Copy File","A file with this name exists!");

}

}

*else* *if*(dirName!="" && fileName=="") *//если* *выбрана* *директория*

{

QFileInfoList **copyList**=QFileInfoList(); *//создание* *контейнера* *для* *хранения* *внутренних* *файлов* *выбранной* *директории*

RecursiveCopyList(*dir*, *copyList*); *//рекурсивное* *наполнение* *контейнера* *внутренними* *файлами* *директории*

AdditionalWindow **window**;

window.*exec*(); *//метод* *выполняет* *появление* *дополнительного* *окна* *для* *копирования*

*if*(window.Cancel()) *//если* *операция* *отменена*

*throw* PerformationException( "Copy", "The operation was canceled!");

*else*

{

window.GetQDir().mkdir(dirName); *//создание* *копии* *директории* *по* *выбранному* *пути*

window.GetQDir().cd(dirName); *//переход* *в* *созданную* *директорию*

*//цикл* *копирования* *элементов* *контейнера* *в* *созданную* *директорию*

foreach(QFileInfo **info**, *copyList*)

{

QString **copyPath** = info.filePath().replace(dir.absolutePath(), window.GetQDir().absolutePath()); *//создание* *пути* *для* *копирования*

*//если* *файл* *-* *копируем* *в* *файл*

bool **dirExists**=*false*; *//флаг* *существования* *директорий* *с* *таким* *именем*

*//проход* *по* *выбранной* *директории* *для* *поиска* *директории* *с* *именем* *копируемой*

foreach(QFileInfo **dirs**, window.GetQDir().entryInfoList(QDir::*Dirs*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(dirs.fileName()==dirName) *//если* *диреткория* *существует*

{

dirExists=*true*; *//установка* *флага* *на* *истинное* *значение*

*break*;

}

}

*if*(!dirExists) *//если* *директорий* *с* *таким* *именем* *нет*

{

*if*(info.isFile()) *//если* *текущий* *элемент* *контейнера* *-* *файл*

{

*if*(info.fileName().contains(".txt")) *//если* *файл* *текстовый*

{

file->*SetPath*(info.absoluteFilePath()); *//установки* *пути* *файла*

*if*(!file->*Copy*(copyPath)) *//если* *копирование* *не* *выполнено*

*throw* PerformationException( "Copy File", "The operation <<Copy>> was not perfomed!");

}

*else*

{

*if*(!(QFile::copy(info.absoluteFilePath(), copyPath))) *//если* *копирование* *не* *выполнено*

*throw* PerformationException( "Copy File", "The operation <<Copy>> was not perfomed!");

}

}

*if*(info.isDir()) *//если* *текущий* *элемент* *-* *директория*

{

*if*(!folder->*Copy*(copyPath)) *//если* *копирование* *не* *выполнено*

*throw* PerformationException( "Copy Folder", "The operation <<Copy>> was not perfomed!");

}

}

*else* *throw* PerformationException( "Copy Folder", "A directory with this name exists!");

}

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

}

}

*catch*(ChoiseException **error**)

{

error.GetException(*this*);

}

*catch*(PerformationException **error**)

{

error.GetException(*this*);

}

*catch* (...)

{

QMessageBox::warning(*this*, "Copy", "Unknown error! Please try again!");

}

}

void MainWindow::**on\_btnReplace\_clicked**() *//слот* *нажатия* *на* *кнопку* *"Replace"*

{

*try* {

*if*(fileName=="" && dirPath=="") *//если* *не* *выбран* *ни* *один* *объект*

*throw* ChoiseException( "Replace", "You was not choose a file or a directory! Please try again");

*else* *if*(fileName!="" && dirName=="") *//если* *выбран* *файл*

{

AdditionalWindow **window**;

window.*exec*(); *//метод* *выполняет* *появление* *дополнительного* *окна* *для* *перемещения*

*if*(window.Cancel()) *//если* *операция* *отменена*

*throw* PerformationException( "Replace File", "The operation was canceled!");

*else*

{

QString **newPath** = window.GetQDir().absolutePath().append("/").append(fileName); *//создание* *пути* *для* *перемещения*

bool **fileExists**=*false*; *//флаг* *существования* *файла* *с* *таким* *именем*

*//цикл* *прохода* *по* *текущей* *директории* *для* *поиска* *файлов* *с* *таким* *именем*

foreach(QFileInfo **files**, window.GetQDir().entryInfoList(QDir::*Files*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(files.fileName()==fileName) *//если* *файл* *с* *таким* *именем* *есть*

{

fileExists=*true*; *//установка* *флаг* *на* *истинное* *значение*

*break*;

}

}

*if*(!fileExists) *//если* *файлов* *с* *таким* *именем* *нет*

{

*if*(fileName.contains(".txt")) *//если* *файл* *текстовый*

{

file->*SetPath*(filePath); *//установка* *пути* *файла*

*if*(!file->*Copy*(newPath)) *//если* *копирование* *не* *произошло*

*throw* PerformationException( "Replace File", "The operation <<Copy>> was not perfomed!");

*if*(!file->*Delete*()) *//если* *удаление* *не* *произошло*

*throw* PerformationException( "Replace File", "The operation <<Delete>> was not perfomed!");

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

*else*

{

*if*(!(QFile::copy(filePath, newPath))) *//если* *копирование* *не* *произошло*

*throw* PerformationException( "Replace File", "The operation <<Copy>> was not perfomed!");

*if*(!(QFile::remove(filePath))) *//если* *удаление* *не* *произошло*

*throw* PerformationException( "Replace File", "The operation <<Delete>> was not perfomed!");

}

}

*else* *throw* PerformationException( "Replace File", "A file with this name exists!");

}

}

*else* *if*(dirName!="" && fileName=="") *//если* *выбрана* *директория*

{

AdditionalWindow **window**;

window.*exec*(); *//метод* *выполняет* *появление* *дополнительного* *окна* *для* *перемещения*

*if*(window.Cancel()) *//если* *операция* *отменена*

*throw* PerformationException( "Replace File", "The operation was canceled!");

*else*

{

QString **newPath** = window.GetQDir().absolutePath().append("/").append(dirName); *//создание* *пути* *для* *перемещения*

bool **dirExists**=*false*; *//флаг* *существования* *директории* *с* *таким* *именем*

*//цикл* *прохода* *по* *выбранной* *директории* *для* *поиска* *директории* *с* *таким* *именем*

foreach(QFileInfo **dirs**, window.GetQDir().entryInfoList(QDir::*Dirs*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(dirs.fileName()==dirName) *//если* *директория* *с* *таким* *именем* *есть*

{

dirExists=*true*; *//установка* *флаг* *на* *истинное* *значение*

*break*;

}

}

*if*(!dirExists) *//если* *директорий* *с* *таким* *именем* *нет*

{

folder->*SetPath*(dirPath); *//установка* *пути* *директории*

*if*(!folder->*Copy*(newPath)) *//если* *копирование* *не* *произошло*

*throw* PerformationException( "Replace Folder", "The operation <<Copy>> was not perfomed!");

*if*(!folder->*Delete*()) *//если* *удаление* *не* *произошло*

*throw* PerformationException("Replace Folder", "The operation <<Delete>> was not perfomed!");

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

*else* *throw* PerformationException( "Replace Folder", "A directory with this name exists!");

}

}

}

*catch*(ChoiseException **error**)

{

error.GetException(*this*);

}

*catch*(PerformationException **error**)

{

error.GetException(*this*);

}

*catch* (...)

{

QMessageBox::warning(*this*, "Replace", "Unknown error! Please try again!");

}

}

void MainWindow::**on\_btnRename\_clicked**() *//слот* *нажатия* *на* *кнопку* *"Rename"*

{

QDir **dir**=QDir(model->filePath(ui->lvSource->rootIndex())); *//если* *не* *выбран* *ни* *один* *объект*

*try* {

*if*(dir.absolutePath()=="D:/Qt/Projects/build-QWERTY-Desktop\_Qt\_6\_4\_0\_MinGW\_64\_bit-Debug") *//если* *это* *корневая* *директория*

*throw* RootDirectoryException("Create", "You are in a root directory! Please choose an another directory");

*else*

{

*if*(filePath=="" && dirPath=="") *//если* *не* *выбран* *ни* *один* *объект*

{

*throw* ChoiseException( "Rename", "You was not choose a file or a directory! Please try again");

}

*else* *if*(filePath!="" && dirPath=="") *//если* *выбран* *файл*

{

RenameWindow **name**;

name.*exec*(); *//метод* *выполняет* *появление* *окна* *для* *переименования* *файла*

*if*(name.GetName()=="") *//если* *имя* *не* *введено*

*throw* ExceptionEmpty( "Rename File", "A new file name is empty! Please try again");

*else*

{

QString **newPath**=dir.absolutePath().append("/").append(name.GetName()); *//создание* *нового* *пути* *с* *учетом* *переименования*

bool **fileExists**=*false*; *//флаг* *существования* *файла* *с* *таким* *именем*

*//цикл* *прохода* *по* *текущей* *директории* *для* *поиска* *файлов* *с* *таким* *именем*

foreach(QFileInfo **files**, dir.entryInfoList(QDir::*Files*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(files.fileName()==name.GetName()) *//если* *файл* *с* *таким* *именем* *есть*

{

fileExists=*true*; *//установка* *флаг* *на* *истинное* *значение*

*break*;

}

}

*if*(!fileExists)*//если* *файлов* *с* *таким* *именем* *нет*

{

*if*(fileName.contains(".txt")) *//если* *файл* *текстовый*

{

file->*SetPath*(filePath); *//установка* *пути* *файла*

*if*(!file->*Rename*(newPath)) *//если* *переименование* *не* *произошло*

*throw* PerformationException( "Rename File", "The operation <<Rename>> was not perfomed!");

*//очистка* *пути* *файла*

filePath=fileName="";

file->*SetPath*(filePath);

}

*else*

{

*if*(!(QFile::rename(filePath, newPath))) *//если* *переименование* *не* *произошло*

*throw* PerformationException( "Rename File", "The operation <<Rename>> was not perfomed!");

}

}

*else* *throw* PerformationException( "Rename File", "A file with this name exists!");

}

}

*else* *if*(dirPath!="" && filePath=="") *//если* *выбрана* *директория*

{

RenameWindow **name**;

name.*exec*(); *//метод* *выполняет* *появление* *окна* *для* *переименования* *директории*

*if*(name.GetName()=="") *//если* *имя* *не* *введено*

*throw* ExceptionEmpty("Rename Folder", "A new folder name is empty! Please try again");

*else*

{

QString **newPath**=dir.absolutePath().append("/").append(name.GetName()); *//создание* *нового* *пути* *с* *учетом* *переименования*

bool **dirExists**=*false*;*//флаг* *существования* *директории* *с* *таким* *именем*

*//цикл* *прохода* *по* *выбранной* *директории* *для* *поиска* *директории* *с* *таким* *именем*

foreach(QFileInfo **dirs**, dir.entryInfoList(QDir::*Dirs*|QDir::*NoDotAndDotDot*, QDir::*Name*))

{

*if*(dirs.fileName()==name.GetName()) *//если* *директория* *с* *таким* *именем* *есть*

{

dirExists=*true*; *//установка* *флаг* *на* *истинное* *значение*

*break*;

}

}

*if*(!dirExists) *//если* *директорий* *с* *таким* *именем* *нет*

{

folder->*SetPath*(dirPath); *//установка* *пути* *директории*

*if*(!folder->*Rename*(newPath)) *//если* *переименование* *не* *произошло*

*throw* PerformationException("Rename Folder", "The operation <<Rename>> was not perfomed!");

*//очистка* *пути* *директории*

dirPath=dirName="";

folder->*SetPath*(dirPath);

}

*else* *throw* PerformationException( "Rename Folder", "A directory with this name exists!");

}

}

}

}

*catch*(RootDirectoryException **error**)

{

error.GetException(*this*);

}

*catch*(ChoiseException **error**)

{

error.GetException(*this*);

}

*catch*(PerformationException **error**)

{

error.GetException(*this*);

}

*catch*(ExceptionEmpty **error**)

{

error.GetException(*this*);

}

*catch* (...)

{

QMessageBox::warning(*this*, "Rename", "Unknown error! Please try again!");

}

}

void MainWindow::**on\_lineSearch\_textEdited**(*const* QString &**arg1**) *//слот* *ввода* *имени* *для* *поиска*

{

window.SetName(arg1);

}

void MainWindow::**on\_btnSearch\_clicked**() *//слот* *нажатия* *на* *кнопку* *"Search"*

{

QDir **dir**=QDir(model->filePath(ui->lvSource->rootIndex())); *//получение* *текущей* *директории*

window.Search(*dir*); *//поиск* *по* *имени*

window.SetUi(); *//передача* *результатов* *в* *окно* *отображения*

window.*exec*(); *//выполнение* *появления* *онка* *с* *резуьтатами* *поиска*

window.ResetUi(); *//очистка* *контейнера* *с* *результатами* *и* *окна* *отображения* *результатов*

}

void MainWindow::**on\_listWidget\_currentTextChanged**(*const* QString &**currentText**) *//слот* *копирования* *текущего* *местоположения* *в* *буфер* *обмена*

{

QClipboard\* **pcb** = QApplication::clipboard(); *//создание* *объекта* *для* *взаимодействия* *с* *буфером* *обмена*

pcb->setText(currentText, QClipboard::*Clipboard*); *//копирование* *выбранного* *текста* *в* *буфер* *обмена*

}

Содержимое файла *renamewindow.cpp:*

#include ".h/renamewindow.h"

#include "ui\_renamewindow.h"

RenameWindow::**RenameWindow**(QWidget \***parent**) : QDialog(*parent*), ui(*new* Ui::RenameWindow) *//*

{

ui->setupUi(*this*); *//настраивает* *пользовательский* *интерфейс* *для* *указанного* *виджета*

setWindowTitle("Rename"); *//установка* *имени* *окна*

ui->name->setToolTip("Please don't forget about the extension, if you want to enter a file name"); *//устанока* *информации* *для* *подсказок*

}

RenameWindow::~**RenameWindow**()

{

*delete* ui;

}

QString RenameWindow::**GetName**() *//метод* *передачи* *имени*

{

*return* name;

}

void RenameWindow::**on\_btnOK\_clicked**() *//* *слот* *нажатия* *на* *"OK"*

{

hide(); *//закрытие* *текущего* *окна*

}

void RenameWindow::**on\_btnCancel\_clicked**() *//* *слот* *нажатия* *на* *"Cancel"*

{

name="";

hide();*//закрытие* *текущего* *окна*

}

void RenameWindow::**on\_name\_textEdited**(*const* QString &**arg1**) *//* *слот* *получения* *имени* *объекта*

{

name=arg1;

}

Содержимое файла *searchwindow.cpp:*

#include ".h/searchwindow.h"

#include "ui\_searchwindow.h"

SearchWindow::**SearchWindow**(QWidget \***parent**) : QDialog(*parent*), ui(*new* Ui::SearchWindow)

{

ui->setupUi(*this*); *//настраивает* *пользовательский* *интерфейс* *для* *указанного* *виджета*

setWindowTitle("Search"); *//установка* *имени* *окна*

}

SearchWindow::~**SearchWindow**()

{

*delete* ui;

}

void SearchWindow::**on\_btnOK\_clicked**() *//* *слот* *нажатия* *на* *"OK"*

{

hide(); *//закрытие* *текущего* *окна*

}

void SearchWindow::**SetName**(*const* QString **arg**) *//метод* *получения* *имени* *для* *поиска*

{

searchName=arg;

}

void SearchWindow::**Search**(QDir &**dir**) *//метод* *поиска* *по* *имени*

{

*//проход* *по* *текущему* *диску* *для* *поиска* *объектов* *с* *введенным* *именем*

foreach(QFileInfo **info**, dir.entryInfoList(QDir::*Files*|QDir::*Dirs*|QDir::*NoDotAndDotDot*, QDir::*Name*|QDir::*DirsFirst*))

{

*if*(info.fileName() == searchName)*//* *сравнение* *элементов* *по* *имени*

list.Push(dir.absoluteFilePath(searchName)); *//добавление* *элемента* *в* *контейнер*

*if*(info.isFile())*//* *если* *элемент* *-* *файл,* *переходим* *к* *след* *файлу*

*continue*;

*if*(info.isDir())*//если* *дошли* *до* *этого* *момента,* *значит* *элемент* *-* *директория*

{

dir.cd(info.fileName());*//заходим* *в* *нее*

Search(*dir*);*//* *рекурсивно* *ищем* *различия*

dir.cdUp();*//возврат*

}

}

}

void SearchWindow::**SetUi**() *//метод* *передачи* *результатов* *поиска* *для* *отображения*

{

*if*(list.IsEmpty()) *//если* *контейнер* *пуст*

ui->listWidget->addItem("There are no results with this name! Please try again");

*else*

{

iter.SetIter(list.PeekHead(*true*)); *//установка* *итератора* *на* *начало* *контейнера*

*//цикл* *добавления* *элементов* *контейнера* *в* *окно* *отображения* *результато* *поиска*

*do*

{

ui->listWidget->addItem((\*iter)->info);

}*while*(++iter);

}

}

void SearchWindow::**ResetUi**() *//метод* *очистки* *окна* *отображения* *результатов* *поиска*

{

iter.SetIter(list.PeekHead(*true*)); *//установка* *итератора* *на* *начало* *контейнера*

list.Pop(); *//очистка* *контейнера*

ui->listWidget->clear(); *//очистка* *окна* *вывода* *результатов* *поиска*

}

Содержимое файла *systemfiles.cpp:*

#include ".h/systemfiles.h"

#include<cstdio>

#include<QMessageBox>

*using* *namespace* std;

bool File::***Create***() *//метод* *создания* *текстового* *файла*

{

ofstream **file**; *//создание* *объекта* *класса* *ofstream*

file.open(filePath); *//открытие* *файла* *с* *указанным* *именем*

*if*(!file.is\_open()) *//если* *файл* *не* *открыт*

*return* *false*;

file.close(); *//если* *открыт*

*return* *true*;

}

bool File::***Delete***() *//метод* *удаления* *текстового* *файла*

{

*if*(!remove(filePath)) *//если* *удаление* *не* *выполнено*

*return* *true*;

*return* *false*; *//если* *выполнено*

}

bool File::***Copy***(QString **newPath**) *//метод* *копирования* *текстового* *файла*

{

*const* char\* **new\_path**=newPath.toLocal8Bit().constData(); *//преобразование* *строки* *типа* *QString* *в* *строку* *типа* *string*

ifstream **file**; *//создание* *объекта* *класса* *ifstream*

file.open(filePath); *//открытие* *файла* *с* *указанным* *именем*

ofstream **newFile**; *//создание* *объекта* *класса* *ofstream*

newFile.open(new\_path, ios::app); *//открытие* *файла* *с* *указанным* *именем* *и* *режимом* *записи* *в* *конец* *файла*

*if* (file.is\_open() && newFile.is\_open()) *//если* *оба* *файла* *открыты*

{

string **line**="";

*while* (getline(*file*, *line*)) *//пока* *можно* *производить* *чтение* *из* *файла*

newFile << line << endl; *//чтение* *информации* *из* *файла* *построчно*

}

*else* *//если* *закрыт* *хотя* *бы* *один* *из* *файлов*

{

*if*(file.is\_open())

file.close();

*if*(newFile.is\_open())

newFile.close();

*return* *false*;

}

*//если* *открыт* *хотя* *бы* *один* *из* *файлов*

*if*(file.is\_open())

file.close();

*if*(newFile.is\_open())

newFile.close();

*return* *true*;

}

bool File::***Rename***(QString **newPath**) *//метод* *переименования* *текстового* *файла*

{

*const* char\* **new\_path**=newPath.toLocal8Bit().constData(); *//преобразование* *строки* *типа* *QString* *в* *строку* *типа* *string*

*if*(!rename(filePath, new\_path)) *//если* *переименование* *не* *выполнено*

*return* *true*;

*return* *false*; *//если* *выполнено*

}

void File::***SetPath***(QString **path**) *//метод* *установки* *пути* *текстового* *файла*

{

filePath=path.toLocal8Bit().constData();*//* *преобразует* *тип* *данных* *QString* *в* *тип* *string*

}

*const* char\* File::***GetPath***() *//метод* *передачи* *пути* *текстового* *файла*

{

*return* filePath;

}

bool Folder::***Create***() *//метод* *создания* *директории*

{

*if* (!\_mkdir(dirPath)) *//если* *создание* *не* *выполнено*

*return* *true*;

*return* *false*;*//если* *выполнено*

}

bool Folder::***Delete***() *//метод* *удаления* *директории*

{

*if*(!rmdir(dirPath)) *//если* *удаление* *не* *выполнено*

*return* *true*;

*return* *false*;*//если* *выполнено*

}

bool Folder::***Rename***(QString **newPath**) *//метод* *переименования* *директории*

{

*const* char\* **new\_path**=newPath.toLocal8Bit().constData(); *//преобразование* *строки* *типа* *QString* *в* *строку* *типа* *string*

*if*(!rename(dirPath, new\_path)) *//если* *переименование* *не* *выполнено*

*return* *true*;

*return* *false*;*//если* *выполнено*

}

bool Folder::***Copy***(QString **newPath**) *//метод* *копирования* *директории*

{

*const* char\* **new\_path**=newPath.toLocal8Bit().constData(); *//преобразование* *строки* *типа* *QString* *в* *строку* *типа* *string*

*if* (!\_mkdir(new\_path)) *//если* *создание* *копии* *директории* *не* *выполнено*

*return* *true*;

*return* *false*;*//если* *выполнено*

}

void Folder::***SetPath***(QString **path**) *//метод* *установки* *пути* *директории*

{

dirPath=path.toLocal8Bit().constData();*//* *преобразует* *тип* *данных* *QString* *в* *тип* *string*

}

*const* char\* Folder::***GetPath***() *//метод* *передачи* *пути* *директории*

{

*return* dirPath;

}

Содержимое файла *main.cpp:*

#include ".h/mainwindow.h"

#include <QApplication> *//управляет* *потоком* *управления* *графическим* *интерфейсом* *приложения* *и* *основными* *настройками*

int main(int **argc**, char \***argv**[])

{

QApplication **a**(*argc*, *argv*); *//создание* *объекта* *основного* *класса* *приложения*

MainWindow **w**; *//создание* *объекта* *главного* *окна*

w.show(); *//выполняет* *появление* *главного* *окна*

*return* a.exec(); *//получает* *события* *из* *оконной* *системы* *и* *отправляет* *их* *виджетам* *приложения.*

}