**ПРИЛОЖЕНИЕ А**

(*обязательное*)

Листинг программы с комментариями

// Book.h:

#ifndef BOOK\_H

#define BOOK\_H

#include <QString>

#include <QXmlStreamReader>

#include "List.h"

// класс, содержащий в себе информацию о книге

class Book

{

public:

Book() = default;

~Book() = default;

void SetPathToBookFile(QString);

QString GetPathToBookFile() const;

void ParseBookFile();

List<QString> GetBookText() const;

void ResetBookText();

int GetTotalPagesNumber() const;

void SetCurrentPageNumber(int);

int GetCurrentPageNumber() const;

int GetMaxStringAmount() const;

Book& operator=(const Book&);

private:

void SetBookText(QXmlStreamReader&);

void SetTitleStyle(QXmlStreamReader&);

void SetParagraphStyle(QString);

void SetTotalPagesNumber();

QString pathToBookFile;

List<QString> bookText;

const int MAX\_STRING\_LENGTH = 95;

const int MAX\_STRING\_AMOUNT = 29;

int totalPagesNumber;

int currentPageNumber;

};

#endif

// Book.cpp:

#include <QFile>

#include <QMessageBox>

#include "../include/Book.h"

#include "../include/Exception.h"

void Book::SetPathToBookFile(QString \_pathToBookFile)

{

if (\_pathToBookFile.isEmpty())

{

throw ArgumentNullException("Файл не выбран!");

}

pathToBookFile = \_pathToBookFile;

}

QString Book::GetPathToBookFile() const

{

return pathToBookFile;

}

void Book::ParseBookFile()

{

QFile bookFile(pathToBookFile);

if (!bookFile.open(QIODevice::ReadOnly | QIODevice::Text)) // попытка открыть файл

{

QMessageBox::critical(nullptr, "", "Что-то пошло не так. Файл невозможно открыть!");

return;

}

QXmlStreamReader xmlFile(&bookFile);

while (!xmlFile.atEnd())

{

xmlFile.readNextStartElement();

if (xmlFile.isEndElement()) // перейти на следующую итерацию, если следующий считанный элемент является закрывающим тегом

{

continue;

}

QString tag = xmlFile.name().toString();

if (tag == "body")

{

SetBookText(xmlFile); // если найден тег "body" задать текст книги

break;

}

}

bookFile.close();

}

void Book::SetBookText(QXmlStreamReader& xmlFile)

{

QString tag;

while (tag != "body") // цикл до тех пор, пока не найден конец тега "body"

{

xmlFile.readNextStartElement(); // считать следующий тег

tag = xmlFile.name().toString(); // считанный тег

if (xmlFile.isEndElement()) // перейти на следующую итерацию, если следующий считанный элемент является закрывающим тегом

{

continue;

}

while (tag != "p" && tag != "title") // цикл до тех пор, пока не найден тег "P" (абзац) или тег "title" (заголовок)

{

xmlFile.readNextStartElement();

tag = xmlFile.name().toString();

}

if (tag == "p")

{ SetParagraphStyle(xmlFile.readElementText(QXmlStreamReader::ReadElementTextBehaviour::IncludeChildElements)); // если тег "p" - задать стиль текста абзаца

}

else

{

SetTitleStyle(xmlFile); // иначе задать стиль текста заголовка

}

}

SetTotalPagesNumber(); // установить общее число страниц книги

}

void Book::SetTotalPagesNumber()

{ // метод установки общего числа страниц книги

if (bookText.GetSize() % MAX\_STRING\_AMOUNT == 0)

{

totalPagesNumber = bookText.GetSize() / MAX\_STRING\_AMOUNT;

}

else

{

totalPagesNumber = bookText.GetSize() / MAX\_STRING\_AMOUNT + 1;

}

}

List<QString> Book::GetBookText() const

{

return bookText;

}

void Book::SetTitleStyle(QXmlStreamReader& xmlFile)

{

QString tag;

while (tag != "title") // цикл до тех пор, пока не найден конец тега "title"

{

xmlFile.readNextStartElement();

tag = xmlFile.name().toString();

if (xmlFile.isEndElement()) // перейти на следующую итерацию, если следующий считанный элемент является закрывающим тегом

{

continue;

}

while (tag != "p") // цикл до тех пор, пока не найден текста тега "title"

{

xmlFile.readNextStartElement();

tag = xmlFile.name().toString();

}

// задать элементу тега "title" жирный шрифт, выравненный по центру

bookText.Append("<p align = 'center'><b>" + xmlFile.readElementText(QXmlStreamReader::ReadElementTextBehaviour::IncludeChildElements) +

"</p></b>");

}

}

void Book::SetParagraphStyle(QString paragraphText)

{

if (paragraphText.length() <= MAX\_STRING\_LENGTH) // если абзац меньше, чем максимальная длина одной строки

{

bookText.Append("<p style = \"margin-left: 30; margin-right: 0; margin-top: 0; margin-bottom: 0\">" + paragraphText + "</p>");

return;

}

bool isFirstLineOfParagraph = true;

while (paragraphText.length() > MAX\_STRING\_LENGTH) // пока абзац длинне, чем максимальная длина одной строки

{

QString textToAppend; // текст строки абзаца

for (int i = 0; i < MAX\_STRING\_LENGTH; i++)

{

textToAppend.append(paragraphText[i]);

}

int length = textToAppend.length();

length--;

while (textToAppend[length] != ',' && textToAppend[length] != ';' && textToAppend[length] != ':' && textToAppend[length] != '.'

&& textToAppend[length] != '?' && textToAppend[length] != '!' && (textToAppend[length] != ' ' && paragraphText[length] != ' '))

{ // удалять символ строки до тех пор, пока он не равен знаку препинания, пробелу или концу слова

textToAppend.remove(length, 1);

length--;

}

length++;

paragraphText = paragraphText.remove(0, length); // удалить из текста абзаца считанную строку

if (isFirstLineOfParagraph)

{ // задать текст строки с отступом, если первая строка абзаца

bookText.Append("<p style = \"margin-left: 30; margin-right: 0; margin-top: 0; margin-bottom: 0\">" + textToAppend + "</p>");

isFirstLineOfParagraph = false;

}

else

{

bookText.Append("<p style = \"margin: 0\">" + textToAppend + "</p>"); // задать текст строки

}

}

bookText.Append("<p style = \"margin: 0\">" + paragraphText + "</p>"); // задать текст последней строки абзаца

}

void Book::ResetBookText()

{

bookText.Clear(); // удалить текст книги

}

int Book::GetTotalPagesNumber() const

{

return totalPagesNumber;

}

int Book::GetCurrentPageNumber() const

{

return currentPageNumber;

}

void Book::SetCurrentPageNumber(int \_currentPageNumber)

{

if (\_currentPageNumber > totalPagesNumber || \_currentPageNumber < 1)

{

throw OutOfRangeException("Вы не можете открыть данную страницу!");

}

currentPageNumber = \_currentPageNumber;

}

int Book::GetMaxStringAmount() const

{

return MAX\_STRING\_AMOUNT;

}

Book& Book::operator=(const Book& book)

{

this->currentPageNumber = book.currentPageNumber;

this->pathToBookFile = book.pathToBookFile;

return \*this;

}

// Exception.h:

#ifndef EXCEPTION\_H

#define EXCEPTION\_H

#include <QString>

class Exception

{

public:

Exception(QString);

Exception() = default;

QString GetErrorMessage() const;

private:

QString errorMessage;

};

class ArgumentNullException : public Exception

{

public:

ArgumentNullException(QString);

~ArgumentNullException() = default;

};

class OutOfRangeException : public Exception

{

public:

OutOfRangeException(QString);

~OutOfRangeException() = default;

};

#endif

// Exception.cpp:

#include "../include/Exception.h"

Exception::Exception(QString \_errorMessage) : errorMessage(\_errorMessage)

{ }

QString Exception::GetErrorMessage() const

{

return errorMessage;

}

ArgumentNullException::ArgumentNullException(QString \_errorMessage) : Exception(\_errorMessage)

{ }

OutOfRangeException::OutOfRangeException(QString \_errorMessage) : Exception(\_errorMessage)

{ }

// List.h:

#ifndef LIST\_H

#define LIST\_H

template<class T>

class Iterator;

template<class T>

class Node

{

public:

T data;

Node<T>\* next;

Node (T);

~Node() = default;

};

template<class T>

Node<T>::Node(T \_data) : data(\_data)

{ }

template<class T>

class List

{

public:

List();

~List() = default;

void Append(T);

void Pop();

void Clear();

T operator[](int);

int GetSize() const;

friend class Iterator<T>;

private:

Node<T>\* head;

Node<T>\* tail;

int size;

};

template<class T>

List<T>::List() : head(nullptr), tail(nullptr), size(0)

{ }

template<class T>

void List<T>::Append(T data)

{

Node<T>\* temp = new Node(data);

if (!head)

{

head = temp;

tail = temp;

}

else

{

tail->next = temp;

tail = temp;

}

size++;

}

template<class T>

void List<T>::Pop()

{

if (head)

{

Node<T>\* temp = head;

head = head->next;

delete temp;

}

size--;

}

template<class T>

void List<T>::Clear()

{

int tmpSize = size;

for (int i = 0; i < tmpSize; i++)

{

Pop();

}

head = nullptr;

tail = nullptr;

}

template<class T>

T List<T>::operator[](int index)

{

Iterator<T> iterator = head;

for (int i = 0; i < index; i++)

{

++iterator;

}

return \*iterator;

}

template<class T>

int List<T>::GetSize() const

{

return size;

}

template<class T>

class Iterator

{

public:

Iterator(Node<T>\*);

~Iterator() = default;

Iterator<T>& operator++();

T& operator\*() const;

private:

Node<T>\* currentNode;

};

template<class T>

Iterator<T>::Iterator(Node<T>\* node) : currentNode(node)

{ }

template<class T>

Iterator<T> &Iterator<T>::operator++()

{

currentNode = currentNode->next;

return \*this;

}

template<class T>

T &Iterator<T>::operator\*() const

{

return currentNode->data;

}

#endif

// MainWindow.h:

#ifndef MAIN\_WINDOW\_H

#define MAIN\_WINDOW\_H

#include <QMainWindow>

#include "Book.h"

namespace Ui { class MainWindow; }

// класс для работы с главным окном

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget\* parent = nullptr);

~MainWindow();

private slots:

void on\_aboutProgramButton\_clicked();

void on\_toStartMenuButton\_clicked();

void on\_chooseFileButton\_clicked();

void on\_nextPageButton\_clicked();

void on\_previousPageButton\_clicked();

void on\_findPageButton\_clicked();

void on\_chooseRecentOpenedFileButton\_clicked();

private:

void ConfigureMainWindow();

void ConfigureTabWidget();

void ConfigureMainWindowLabel();

void ConfigureAboutProgramLabel();

void ConfigureBookTab();

void ConfigurePageButtons();

void SetBookLabelText(List<QString>, int);

void AppendPageNumberToBookFile(int);

Ui::MainWindow\* ui;

Book book;

const int WINDOW\_WIDTH = 1360;

const int WINDOW\_HEIGHT = 800;

};

#endif

// MainWindow.cpp:

#include <QFileDialog>

#include <QMessageBox>

#include <QFile>

#include "../include/MainWindow.h"

#include "../include/RecentOpenedFilesWindow.h"

#include "../include/WidgetStyle.h"

#include "../include/Exception.h"

#include "ui\_mainwindow.h"

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

{

ui->setupUi(this);

ConfigureMainWindow();

ConfigureTabWidget();

ConfigureMainWindowLabel();

ConfigureAboutProgramLabel();

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_aboutProgramButton\_clicked()

{

ui->programTab->setCurrentIndex(2); // открыть меню с информацией

}

void MainWindow::on\_toStartMenuButton\_clicked()

{

ui->programTab->setCurrentIndex(0); // открыть стартовое меню

}

void MainWindow::on\_chooseFileButton\_clicked()

{

QString path = QFileDialog::getOpenFileName(this, "Выберите файл", "C:", "Book file (\*.fb2)"); // открыть диалоговое окно для выбора fb2 файла

try

{

book.SetPathToBookFile(path); // попытка задать путь до файла

}

catch (const ArgumentNullException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

book.ResetBookText(); // обнулить текст книги

book.ParseBookFile(); // считать текст книги

book.SetCurrentPageNumber(1); // задать номер текущей страницы

ConfigureBookTab(); // настроить стили меню с книгой

SetBookLabelText(book.GetBookText(), 1); // отобразить текст страницы на окне

QString bookFileName = (QFileInfo(QFile(path).fileName())).fileName();

if (!QFile::exists("RecentOpenedFiles/" + bookFileName))

{

if (!QDir("RecentOpenedFiles").exists())

{

QDir().mkpath("RecentOpenedFiles"); // если директория для ранее открытых файлов не создана, создать ее

}

QFile::copy(path, "RecentOpenedFiles/" + bookFileName); // создать копию файла для возможности открыть его на последней открытой странице

}

AppendPageNumberToBookFile(book.GetCurrentPageNumber()); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_nextPageButton\_clicked()

{

int currentPageNumber = book.GetCurrentPageNumber() + 1;

try

{

book.SetCurrentPageNumber(currentPageNumber); // попытка задать новый номер текущей страницы

}

catch (const OutOfRangeException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

ui->currentPageNumberLabel->setText(QString::number(currentPageNumber)); // отобразить номер текущей страницы

SetBookLabelText(book.GetBookText(), currentPageNumber); // отобразить текст новой страницы

AppendPageNumberToBookFile(book.GetCurrentPageNumber()); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_previousPageButton\_clicked()

{

int currentPageNumber = book.GetCurrentPageNumber() - 1;

try

{

book.SetCurrentPageNumber(currentPageNumber); // попытка задать новый номер текущей страницы

}

catch (const OutOfRangeException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

ui->currentPageNumberLabel->setText(QString::number(currentPageNumber)); // отобразить номер текущей страницы

SetBookLabelText(book.GetBookText(), currentPageNumber); // отобразить текст новой страницы

AppendPageNumberToBookFile(book.GetCurrentPageNumber()); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_findPageButton\_clicked()

{

int pageNumber = ui->findPageLineEdit->text().toInt();

ui->findPageLineEdit->setText(""); // обнулить текст после ввода

try

{

book.SetCurrentPageNumber(pageNumber);

}

catch (const OutOfRangeException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

ui->currentPageNumberLabel->setText(QString::number(pageNumber)); // отобразить номер текущей страницы

SetBookLabelText(book.GetBookText(), pageNumber); // отобразить текст новой страницы

AppendPageNumberToBookFile(pageNumber); // добавить номер страницы в конец файла книги

}

void MainWindow::on\_chooseRecentOpenedFileButton\_clicked()

{

ui->programTab->setCurrentIndex(0);

QDir recentOpenedFilesDirectory("RecentOpenedFiles/");

if (recentOpenedFilesDirectory.isEmpty())

{

QMessageBox::warning(this, "", "Ранее открытые файлы отсутсвуют!"); // если ранее открытые файлы отсутсвуют

return;

}

RecentOpenedFilesWindow window(this);

window.show();

window.exec();

try

{

book = window.GetBook(); // попытка получить книгу, если она была выбрана

}

catch (const ArgumentNullException& exception)

{

QMessageBox::warning(this, "", exception.GetErrorMessage());

return;

}

book.ResetBookText(); // обнулить текст книги

book.ParseBookFile(); // считать текст книги

ConfigureBookTab(); // настроить стили меню с книгой

SetBookLabelText(book.GetBookText(), book.GetCurrentPageNumber()); // отобразить текст новой страницы

}

void MainWindow::ConfigureMainWindow()

{

setFixedSize(WINDOW\_WIDTH, WINDOW\_HEIGHT); // задать размер окна

setWindowTitle("Book Reader"); // задать заголовок окна

setWindowIcon(QIcon(":/img/program\_icon.png")); // задать икноку окна

QPixmap labelPicture(":/img/book\_image.jpg"); // отобразить картинку на окне

labelPicture = labelPicture.scaled(labelPicture.width() \* 0.8f, labelPicture.height() \* 0.8f);

ui->imageLabel->setFixedSize(labelPicture.size());

ui->imageLabel->setPixmap(labelPicture);

ui->toStartMenuButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

ui->chooseFileButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

ui->chooseRecentOpenedFileButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

ui->aboutProgramButton->setStyleSheet(WidgetStyle::GetMainWindowButtonsStyle());

}

void MainWindow::ConfigureTabWidget()

{

ui->programTab->setStyleSheet(WidgetStyle::GetTabStyle()); // задать стили для QTabWidget

ui->programTab->setCurrentIndex(0);

}

void MainWindow::ConfigureMainWindowLabel()

{

ui->mainWindowLabel->setStyleSheet(WidgetStyle::GetMainWindowLabelStyle()); // задать стили для стартового окна

ui->mainWindowLabel->setText("Book Reader"); // задать текст на стартовом окне

}

void MainWindow::ConfigureAboutProgramLabel()

{

ui->aboutProgramLabel->setStyleSheet(WidgetStyle::GetAboutPogramLabelStyle()); // задать стили для меню с информацией

ui->aboutProgramLabel->setText(" Эта программа предназначена для чтения электронных книг, имеющих расширение .fb2." // устноавить текст информации

" Для того, чтобы начать читать, необходимо:\n"

"1) нажать на кнопку \"Выбрать файл\" в левой части окна;\n"

"2) в открывшемся диалоговом окне выбрать файл, имеющий расширение .fb2;\n"

"3) выбрав нужный файл, нажать кнопку \"Открыть\".\n"

"После открытия файла для перехода на следующую страницу необходимо нажать на кнопку \"->\" в нижней части экрана, "

"для перехода на предыдущую страницу — нажать на кнопку \"<-\".\n"

" Если необходимо перейти на конкретную страницу, то необходимо в поле снизу ввести номер страницы и "

"нажать на кнопку с изображенной на ней лупой.\n"

" При нажатии на кнопку \"Выбрать ранее открытый файл\" можно открыть один из ранее открытых файлов на"

"последней открытой странице либо удалить этот файл.");

}

void MainWindow::ConfigureBookTab()

{ // задать стили для меню с книгой

ui->programTab->setCurrentIndex(1);

ui->bookLabel->setStyleSheet(WidgetStyle::GetBookLabelStyle());

// задать стили для части окна с номерами текущего и обшего числа страниц

ui->totalPagesNumberLabel->setStyleSheet(WidgetStyle::GetPageNumberLabelsStyle());

ui->totalPagesNumberLabel->setText(QString::number(book.GetTotalPagesNumber()));

ui->currentPageNumberLabel->setStyleSheet(WidgetStyle::GetPageNumberLabelsStyle());

ui->currentPageNumberLabel->setText(QString::number(book.GetCurrentPageNumber()));

ui->slashLabel->setStyleSheet("font-size: 18px;");

// задать стили для меню поиска страницы

QIcon findPageButtonIcon(":/img/magnifier\_icon.png");

ui->findPageButton->setIcon(findPageButtonIcon);

ui->findPageButton->setIconSize(ui->findPageButton->size());

ui->findPageButton->setStyleSheet(WidgetStyle::GetFindPageButtonStyle());

ui->findPageLineEdit->setStyleSheet(WidgetStyle::GetFindPageLineEditStyle());

}

void MainWindow::SetBookLabelText(List<QString> bookText, int currentPageNumber)

{

QString pageText;

int maxStringAmount = book.GetMaxStringAmount();

for (int i = 0; i < maxStringAmount; i++) // пока не будет добавлено максимальное число строк к тексту страницы

{

int index = (currentPageNumber - 1) \* maxStringAmount + i;

if (index > bookText.GetSize() - 1) // если на последней странице строк меньше, чем из максимальное количество

{

break;

}

pageText.append(bookText[index]); // добавить строку к тексту страницы

}

ui->bookLabel->setText(pageText); // отобразить текст страницы

}

void MainWindow::AppendPageNumberToBookFile(int pageNumber)

{

QString bookFileName = (QFileInfo(QFile(book.GetPathToBookFile()).fileName())).fileName(); // получить имя файла

QFile bookFile("RecentOpenedFiles/" + bookFileName);

bookFile.open(QIODevice::Text | QIODevice::Append); // открыть файл

QTextStream bookFileStream(&bookFile);

bookFileStream << '\n' << pageNumber; // добавить номер страницы в файл

}

// RecentOpenedFilesWindow.h:

#ifndef RECENT\_OPENED\_FILES\_WINDOW\_H

#define RECENT\_OPENED\_FILES\_WINDOW\_H

#include <QDialog>

#include <QPushButton>

#include "Book.h"

namespace Ui { class RecentOpenedFilesWindow; }

// класс для работы с окном со списком ранее открытых файлов

class RecentOpenedFilesWindow : public QDialog

{

Q\_OBJECT

public:

RecentOpenedFilesWindow(QWidget \*parent = nullptr);

~RecentOpenedFilesWindow();

Book GetBook() const;

private slots:

void RecentOpenedFileButtonClicked();

void DeleteRecentOpenedFileButtonClicked();

private:

void ConfigureDialogWindow();

void SetRecentOpenedFiles();

void SetLastOpenedPageNumber(QString);

void CreateRecentOpenedFilesButtons();

void CreateDeleteRecentOpenedFilesButtons();

void ChangeRecentOpenedFilesButtonsPosition(int);

Ui::RecentOpenedFilesWindow \*ui;

const int WINDOW\_WIDTH = 800;

const int WINDOW\_HEIGHT = 600;

QList<QString> recentOpenedFiles;

QList<QPushButton\*> recentOpenedFilesButtons;

QList<QPushButton\*> deleteRecentOpenedFilesButtons;

Book book;

};

#endif

// RecentOpenedFilesWindow.cpp:

#include <QDir>

#include "../include/RecentOpenedFilesWindow.h"

#include "../include/WidgetStyle.h"

#include "../include/Exception.h"

#include "ui\_recentopenedfileswindow.h"

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

{

ui->setupUi(this);

ConfigureDialogWindow();

SetRecentOpenedFiles();

CreateRecentOpenedFilesButtons();

CreateDeleteRecentOpenedFilesButtons();

}

RecentOpenedFilesWindow::~RecentOpenedFilesWindow()

{

delete ui;

}

Book RecentOpenedFilesWindow::GetBook() const

{

if ((book.GetPathToBookFile()).isEmpty())

{

throw ArgumentNullException("Файл не выбран");

}

return book;

}

void RecentOpenedFilesWindow::RecentOpenedFileButtonClicked()

{

QPushButton\* btn = (QPushButton\*)sender();

book.SetPathToBookFile("RecentOpenedFiles/" + (btn->text()).append(".fb2")); // получить имя выбранной книги

book.ParseBookFile();

SetLastOpenedPageNumber((btn->text()).append(".fb2")); // задать последнюю страницы выбранной книги

}

void RecentOpenedFilesWindow::DeleteRecentOpenedFileButtonClicked()

{

QPushButton\* btn = (QPushButton\*)sender();

int deleteButtonY = btn->y(); // получить y координату выбранной кнопки удалить (она равна y координате кнопки с книгой)

for (int i = 0; i < recentOpenedFilesButtons.size(); i++)

{

if (deleteButtonY == recentOpenedFilesButtons[i]->y())

{

ChangeRecentOpenedFilesButtonsPosition(i); // если найден номер кнопки с названием книги, который необходимо удалить

break;

}

}

}

void RecentOpenedFilesWindow::ConfigureDialogWindow()

{

setFixedSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);

setWindowTitle("Ранее открытые файлы");

setWindowIcon(QIcon(":/img/program\_icon.png"));

setWindowModality(Qt::ApplicationModal);

}

void RecentOpenedFilesWindow::SetRecentOpenedFiles()

{

QDir recentOpenedFilesDirectory("RecentOpenedFiles/");

recentOpenedFiles = recentOpenedFilesDirectory.entryList(QDir::NoDotAndDotDot | QDir::AllEntries); // получить список всех ранее открытых файлов из директории

}

void RecentOpenedFilesWindow::CreateRecentOpenedFilesButtons()

{

const int BUTTON\_WIDTH = WINDOW\_WIDTH; // длина кнопки с названием книги

const int BUTTON\_HEIGHT = 30; // высота кнопки с названием книги

for (int i = 0; i < recentOpenedFiles.size(); i++) // цикл по всем ранее открытым книгам

{

recentOpenedFilesButtons.append(new QPushButton(this)); // добавление кнопки в список

recentOpenedFilesButtons[i]->setText(recentOpenedFiles[i].remove(recentOpenedFiles[i].size() - 4, 4)); // текст кнопки - имя файла без расширения

recentOpenedFilesButtons[i]->setFixedSize(BUTTON\_WIDTH, BUTTON\_HEIGHT); // установить размеры кнопки

recentOpenedFilesButtons[i]->move(0, i \* BUTTON\_HEIGHT); // задать координаты кнопки

recentOpenedFilesButtons[i]->setStyleSheet(WidgetStyle::GetRecentOpenedFilesButtonsStyle()); // установка стилей для кнопки

// обработка нажатия кнопки

connect(recentOpenedFilesButtons[i], SIGNAL(clicked()), this, SLOT(RecentOpenedFileButtonClicked()));

connect(recentOpenedFilesButtons[i], SIGNAL(clicked()), this, SLOT(close()));

}

}

void RecentOpenedFilesWindow::CreateDeleteRecentOpenedFilesButtons()

{

const int BUTTON\_WIDTH = 50; // длина кнопки удаления

const int BUTTON\_HEIGHT = 30; // высота кнопки удаления

for (int i = 0; i < recentOpenedFiles.size(); i++) // цикл по всем ранее открытым книгам

{

deleteRecentOpenedFilesButtons.append(new QPushButton(this)); // добавление кнопки в список

deleteRecentOpenedFilesButtons[i]->setFixedSize(BUTTON\_WIDTH, BUTTON\_HEIGHT); // установить размеры кнопки

deleteRecentOpenedFilesButtons[i]->move(WINDOW\_WIDTH - 50, i \* BUTTON\_HEIGHT); // задать координаты кнопки

deleteRecentOpenedFilesButtons[i]->setStyleSheet(WidgetStyle::GetDeleteRecentOpenedFilesButtonsStyle()); // установка стилей для кнопки

// создание иконки кнопки

QIcon deleteRecentOpenedFileButtonIcon(":/img/delete\_file\_icon.png");

deleteRecentOpenedFilesButtons[i]->setIcon(deleteRecentOpenedFileButtonIcon);

deleteRecentOpenedFilesButtons[i]->setIconSize(deleteRecentOpenedFilesButtons[i]->size());

// обработка нажатия кнопки

connect(deleteRecentOpenedFilesButtons[i], SIGNAL(clicked()), this, SLOT(DeleteRecentOpenedFileButtonClicked()));

}

}

void RecentOpenedFilesWindow::SetLastOpenedPageNumber(QString fileName)

{

QFile bookFile("RecentOpenedFiles/" + fileName);

bookFile.open(QIODevice::Text | QIODevice::ReadOnly); // открытие выбранного файла

QTextStream bookFileStream(&bookFile);

QString bookFileText = bookFileStream.readAll(); // получить весь текст файла

QString lastOpenedPage;

for (int i = bookFileText.length() - 1; bookFileText[i] != '\n'; i--) // цикл до конца последней строки файла

{

lastOpenedPage.append(bookFileText[i]); // добавить символ с конца файла (в конец файла добавлялись номера страниц)

}

std::reverse(lastOpenedPage.begin(), lastOpenedPage.end()); // реверс строки

book.SetCurrentPageNumber(lastOpenedPage.toInt()); // перевод номера последней открытой страницы в int

}

void RecentOpenedFilesWindow::ChangeRecentOpenedFilesButtonsPosition(int i)

{

QFile::remove("RecentOpenedFiles/" + (recentOpenedFilesButtons[i]->text()).append(".fb2")); // удаление выбранного файла из директории

// закрытие кнопок, соответствующих выбранному файлу

recentOpenedFilesButtons[i]->close();

deleteRecentOpenedFilesButtons[i]->close();

// смещение кнопок, если нажатая кнопка не последняя

if (i < recentOpenedFilesButtons.size() - 1)

{

for (int j = i + 1; j < recentOpenedFilesButtons.size(); j++)

{

recentOpenedFilesButtons[j]->move(0, 30 \* (j - 1)); // сместить вверх все кнопки ниже той, которая была удалена

deleteRecentOpenedFilesButtons[j]->move(WINDOW\_WIDTH - 50, 30 \* (j - 1));

}

}

// удаление выбранного файла и кнопок из списков

recentOpenedFiles.removeAt(i);

recentOpenedFilesButtons.removeAt(i);

deleteRecentOpenedFilesButtons.removeAt(i);

}

// WidgetStyle.h:

#ifndef WIDGET\_STYLE\_H

#define WIDGET\_STYLE\_H

#include <QString>

// класс, содержащий в себе стили для всех виджетов

class WidgetStyle

{

public:

static QString GetTabStyle();

static QString GetAboutPogramLabelStyle();

static QString GetMainWindowLabelStyle();

static QString GetPageNumberLabelsStyle();

static QString GetBookLabelStyle();

static QString GetFindPageButtonStyle();

static QString GetFindPageLineEditStyle();

static QString GetRecentOpenedFilesButtonsStyle();

static QString GetDeleteRecentOpenedFilesButtonsStyle();

static QString GetMainWindowButtonsStyle();

};

#endif

// WidgetStyle.cpp:

#include "../include/WidgetStyle.h"

QString WidgetStyle::GetTabStyle()

{

return "QTabBar::tab {"

"width: 0;"

"height: 0;"

"}"

"QTabWidget::pane {"

"background: #FFFAF0;"

"border: 1px solid gray;"

"}";

}

QString WidgetStyle::GetAboutPogramLabelStyle()

{

return "QLabel {"

"qproperty-wordWrap: true;"

"font-size: 20px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetMainWindowLabelStyle()

{

return "QLabel {"

"qproperty-wordWrap: false;"

"font-size: 70px;"

"font-family: \"Times New Roman\";"

"font-style: italic;"

"font-weight: 500;"

"}";

}

QString WidgetStyle::GetPageNumberLabelsStyle()

{

return "QLabel {"

"qproperty-wordWrap: false;"

"font-size: 16px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetBookLabelStyle()

{

return "QLabel {"

"qproperty-wordWrap: true;"

"font-size: 20px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetFindPageButtonStyle()

{

return "QPushButton {"

"background: #FFFAF0;"

"border: 0px solid white;"

"}";

}

QString WidgetStyle::GetFindPageLineEditStyle()

{

return "QLineEdit {"

"background: #FFFAF0;"

"border: 1px solid gray;"

"font-size: 16px;"

"font-family: \"Times New Roman\";"

"}";

}

QString WidgetStyle::GetRecentOpenedFilesButtonsStyle()

{

return "QPushButton {"

"border: 1px solid gray;"

"font-size: 16px;"

"font-family: \"Times New Roman\";"

"}"

"QPushButton:hover {"

"border: 1px solid #1E90FF;"

"}"

"QPushButton::pressed {"

"border: 1px solid #005499;"

"background-color: #CCE4F7;"

"}";

}

QString WidgetStyle::GetDeleteRecentOpenedFilesButtonsStyle()

{

return "QPushButton {"

"border: 0px solid white;"

"}";

}

QString WidgetStyle::GetMainWindowButtonsStyle()

{

return "QPushButton {"

"font-size: 12px;"

"}";

}

// main.cpp:

#include <QApplication>

#include "../include/MainWindow.h"

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

{

QApplication app(argc, argv);

MainWindow w;

w.show();

return app.exec();

}