МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ст. преподаватель |  |  |  | Е. О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ  Разработка приложения для организации взаимодействия объектов при заданных критериях |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |

РАБОТУ ВЫПОЛНИЛ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4932 |  |  |  | Н.С.Иванов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

**Задание**

**на курсовой проект по дисциплине**

**«Объектно-ориентированное программирование»**

Студенту группы 4932 Н.С.Иванову

№ группы Ф.И.О.

Тема «Разработка приложения для организации взаимодействия объектов при заданных критериях»

Исходные данные: Разработка системы классов для обеспечения работы электронной библиотеки

Проект должен содержать:

* анализ предметной области
* разработку классов
* разработку тестового приложения
* оформление пояснительной записки по результатам выполнения проекта
* создание презентации к проекту

Срок сдачи законченного проекта

Руководитель проекта ст.преп. Е.О.Шумова

Дата выдачи задания 01.09.2021 г.

**СОДЕРЖАНИЕ**

Содержание 2

[Цели и задачи курсового проектирования 5](#_Toc90468004)

[ВВЕДЕНИЕ 6](#_Toc90468005)

[1 Постановка задачи 7](#_Toc90468006)

[1.1 Анализ предметной области 7](#_Toc90468007)

[1.2 Формулировка технического задания 8](#_Toc90468008)

[2 Проектирование классов 8](#_Toc90468009)

[2.1 Классы сущностей: 8](#_Toc90468010)

[2.2 Управляющие классы: 9](#_Toc90468011)

[2.3 Интерфейсные классы: 10](#_Toc90468012)

[2.4 Диаграмма классов: 10](#_Toc90468013)

[3 Разработка приложения 11](#_Toc90468014)

[3.1.  Разработка интерфейса приложения 11](#_Toc90468015)

[3.2.  Реализация классов 16](#_Toc90468016)

[3.2.1 Класс ListShop 16](#_Toc90468017)

[3.2.2 Класс Man 16](#_Toc90468018)

[3.2.3 Класс Worker : Man, IFIOtoString 17](#_Toc90468019)

[3.2.4 Класс Food 17](#_Toc90468020)

[3.2.5 Класс Order 18](#_Toc90468021)

[3.2.7 Класс LoadAndSave 22](#_Toc90468024)

[3.2.8 Интерфейс IFIOtoString 23](#_Toc90468026)

[4.Тестирование 23](#_Toc90468027)

[ЗАКЛЮЧЕНИЕ 29](#_Toc90468028)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc90468029)

[ПРИЛОЖЕНИЕ A 31](#_Toc90468030)

# ВВЕДЕНИЕ

В данной работе рассматривается описание программы для обеспечения работы электронной библиотекина основе объектно-ориентированного подхода.

Программа демонстрирует удобный и практичный графический интерфейс, охватывающий полную модель обработки информации, чтобы взаимодействие с системой было качественным, а также любому человеку без технического образования было понятно, как с ним работать.

В качестве основного инструмента разработки применяется IntelliJ Idea, приложение будет разработано на языке Java с использованием JavaFx, так как на мой взгляд - это наиболее удобный и простой из известных на данный момент интерфейс для программирования приложений.

# Постановка задачи

## Анализ предметной области

Анализ предметной области программы для обеспечения работы электронной библиотеки выявил следующее:

Компонентами данной предметной области являются pdf документы, метки (теги) для документов, статистика их использования. В словарь предметной области можно записать такие понятия, как:

* Книга pdf
* Метка на книге
* Статистика использования книги
* Системная информация (такая как путь к файлу)

Данное приложение должно позволять пользователю создавать записи, удалять и обновлять.

Исходя из словаря предметной области для реализации работы с данными компонентами необходимо разработать классы:

* Книга
* Книга pdf
* Метка
* Статистика использования
* Системная информация
* И другие классы для взаимодействия между ними и работы приложения

В данной сфере нужно реализовать добавление и удаление книг, поиск информации в книге (сквозной поиск по полям, несущим смысловую нагрузку), добавление и удаления меток на книги для их удобной организации.

Для реализации поставленных задач должны выполняться следующие требования:

* Должна быть разработана система классов, соответствующая предметной области «Электронная библиотека»;
* Приложение должно функционировать в соответствии с разрабатываемой системой классов;
* Программа должна обрабатывать максимальное количество исключительных ситуаций.

## Формулировка технического задания

Исходя из анализа предметной области, можно выделить следующие функциональные требования:

* + Работа со списком книг;
  + Работа со списком меток;
  + Поиск по меткам
  + Поиск по информации о книге

# Проектирование классов

## Классы сущностей:

В Объектно Ориентированном Программировании главным элементом является класс, включающий множество объектов с одинаковыми свойствами, операциями и отношениями. Класс имеет внутреннее (реализацию) и внешнее представление – интерфейс. В данной программе реализованы классы Book, BookDatesAndCounts, BookDjvu, BookList, BookPdf, BookStatistic, BookSystemInfo, BookTag, BookTagList, BookType.

Класс Book

* author
* title
* date
* type
* statistic
* tagsContainer

Класс Book является родителем класса BookPdf. Он содержит общие характеристики, описывающие книгу, такие как автор, название, дата добавления в библиотеку.

Класс BookPdf

* BookPdf (конструктор)
* openTotal

Класс BookPdf расширяет класс Book, за счёт чего получается реализация книг pdf. Класс BookPdf определяет конструктор и атрибут количество открытий файла (djvu файл не может быть открыт т.к. его поддержка разрабатываемым приложением не осуществляется).

Класс BookDjvu

* BookDjvu (конструктор)

Класс BookDjvu демонстрационный. Выкидывает исключение в конструкторе, которое отображает ошибку, сообщающую о том, что расширение файла не поддерживается.

Класс BookTag

* Name
* Color

BookTag описывает метку для книги. Содержит такие атрибуты, как название метки, цвет метки в интерфейсе.

Класс BookStatistic

* countPagesTotal
* countPagesRead

Класс BookStatistic описывает статистику использования книги, содержит такие атрибуты, как количество страниц в книге, количество прочитанных страниц.

## Управляющие классы:

Классы, содержащие методы, осуществляющие связь между интерфейсными классами и классами сущностей:

Класс LibraryFacade

* addBookToLibrary
* rmBookFromLibrary
* getBooks
* openDocument

Класс LibraryFacade определяет базовые функции для работы с книгами

Класс ControllerMainWindow

* addFileToLib
* editFileInLib
* rmFileFromLib
* openFileInLib

Класс ControllerMainWindow определяет базовые функции для работы с файлами, некоторые из которых делегирует фасаду

Класс ControllerBookEditForm

* editTags
* saveBook

Класс ControllerBookEditForm определяет методы для сохранения изменений для книги и открытия окна редактирования меток для этой книги.

Класс ControllerBookAddForm

* saveBook

Класс ControllerBookAddForm определяет поля методы для сохранения информации о книге в библиотеке

Класс ControllerBookTagForm

* addNewTag
* deleteTag
* editTag
* saveTags

Класс ControllerBookTagForm определяет поля и методы для работы с тегами.

## Интерфейсные классы:

Интерфейс ExceptionDisplayAlert

* showAlert

Интерфейс ExceptionDisplayAlert объявляет метод showAlert который должны реализовывать классы, которые его реализовывают.

Интерфейс Controller

* openWindow

Интерфейс Controller позволяет объединить контроллеры приложения в одну коллекцию-синглтон для доступа к любому контроллеру из любого места программы. Объявляет метод openWindow, который должны реализовывать классы, которые его реализовывают.

## Диаграмма классов:

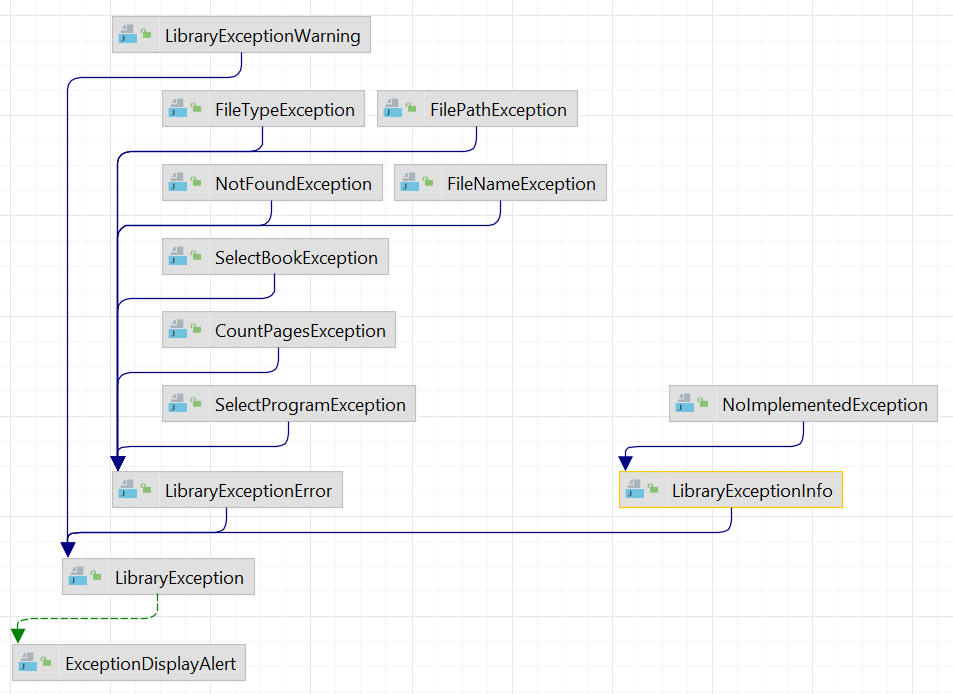


Рис 1. Структура классов предупреждений

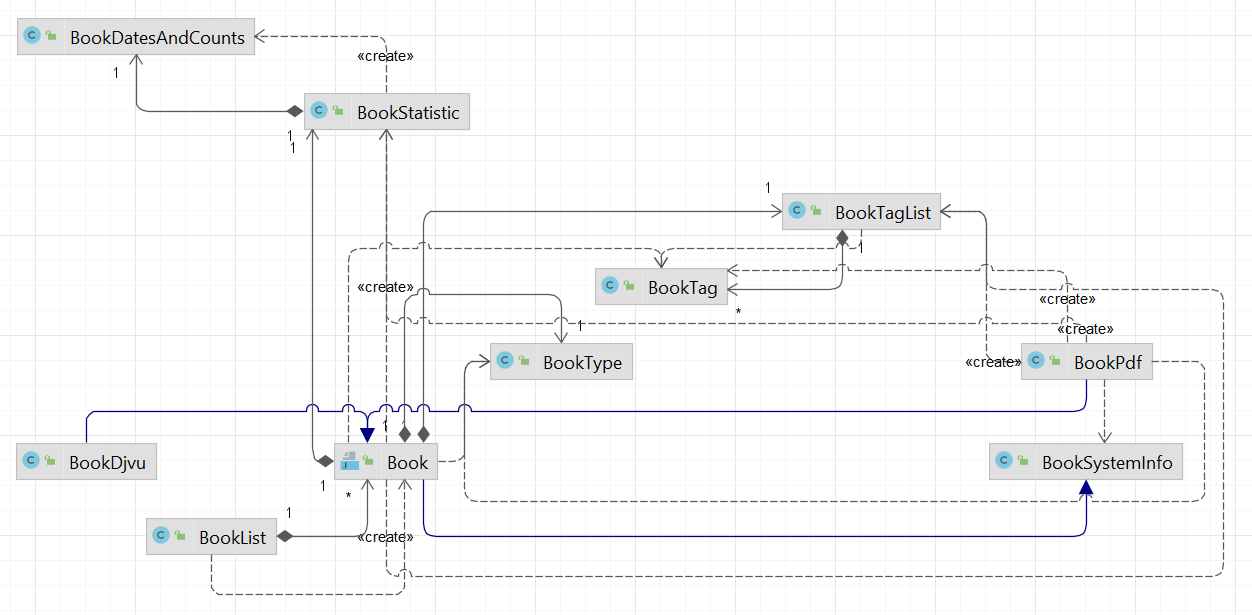


Рис 2. Структура классов сущностей

# Разработка приложения

## 3.1.  Разработка интерфейса приложения

Пользовательский интерфейс приложения реализуется на языке Java.

Реализовано главное окно в виде списка книг (Рис. 3)

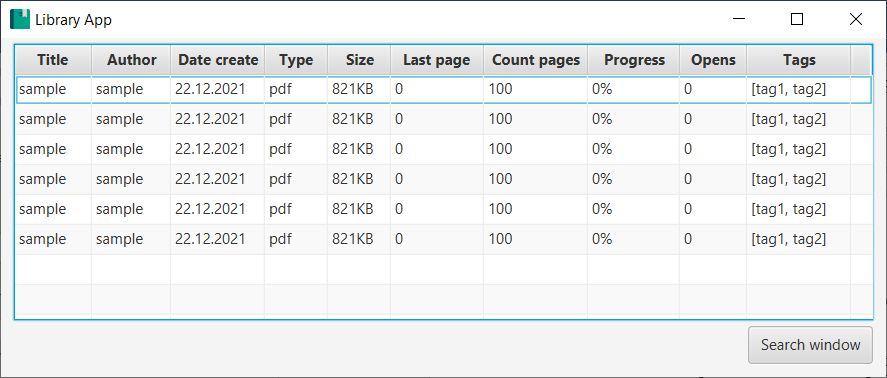


Рис 3. Главное окно

Из главного окна можно попасть в окно добавления файла, в котором представлены поля для заполнения

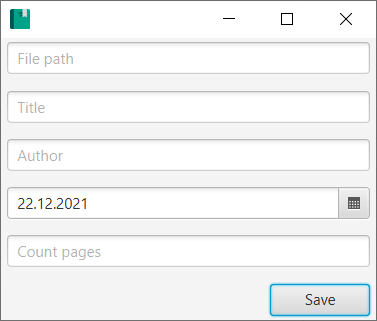


Рис 4. Окно добавления файла

Из главного окна можно попасть в окно изменения файла, в котором представлены поля для заполнения

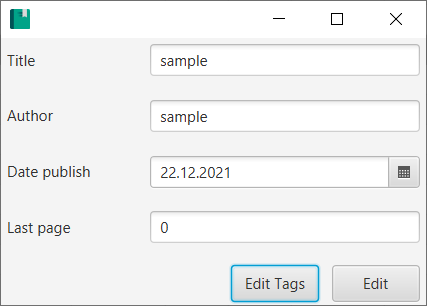


Рис 5. Меню редактирования файла

Из окна редактирования файла можно попасть в окно редактирования меток, в котором представлены поля для заполнения

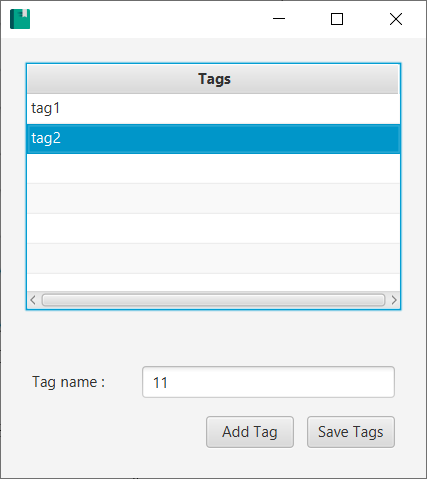


Рис 6. Меню редактирования меток

Из главного окна можно попасть в окно поиска, в котором представлены поля для заполнения

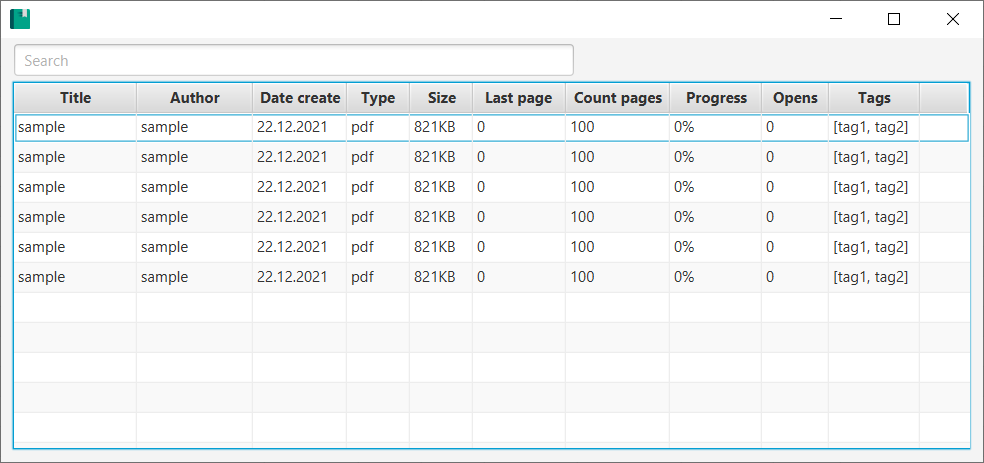
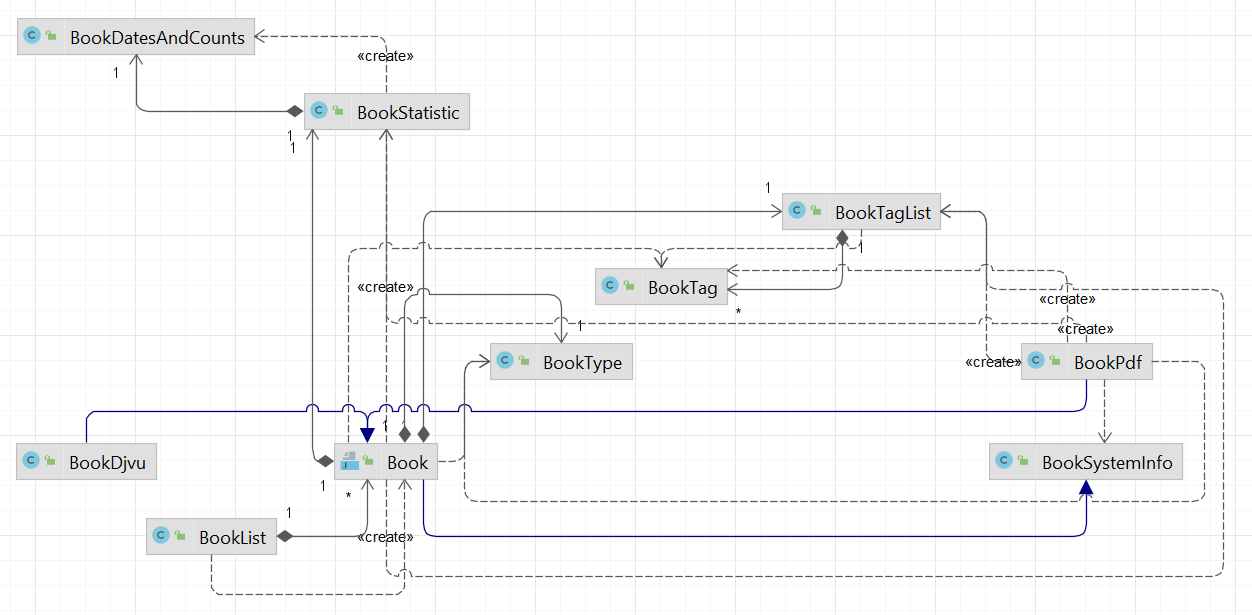


Рис 7. Окно поиска информации

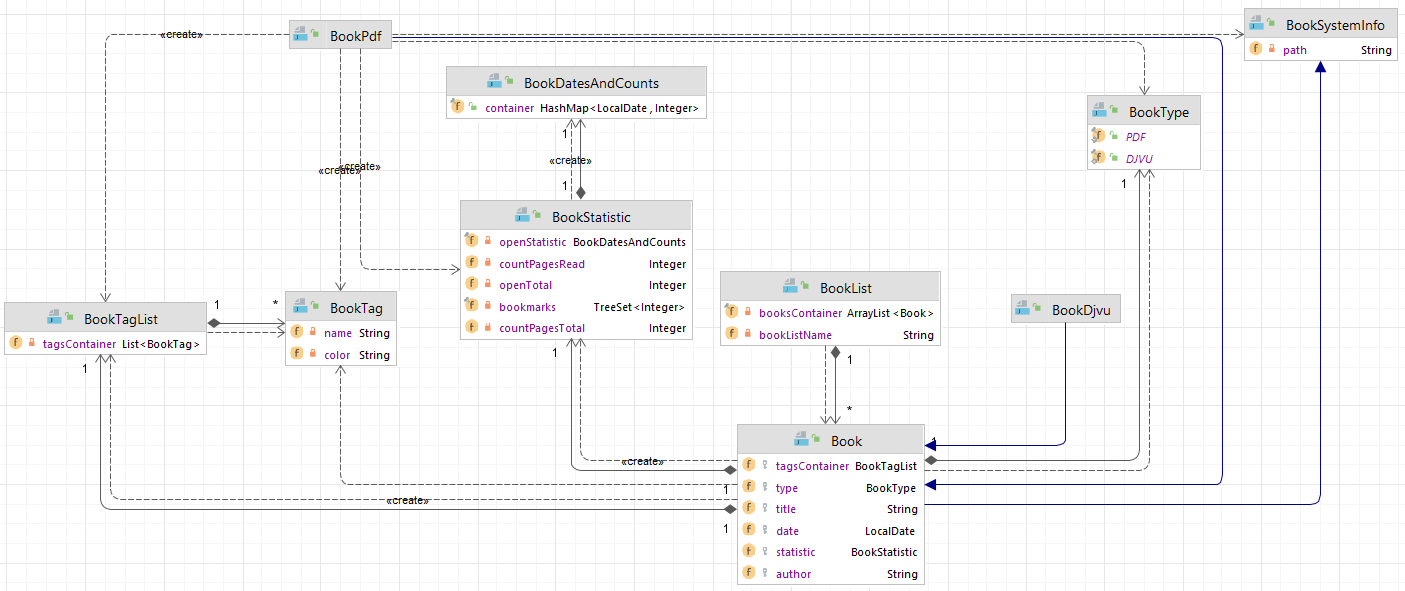
## 3.2.  Реализация классов

### Структура классов

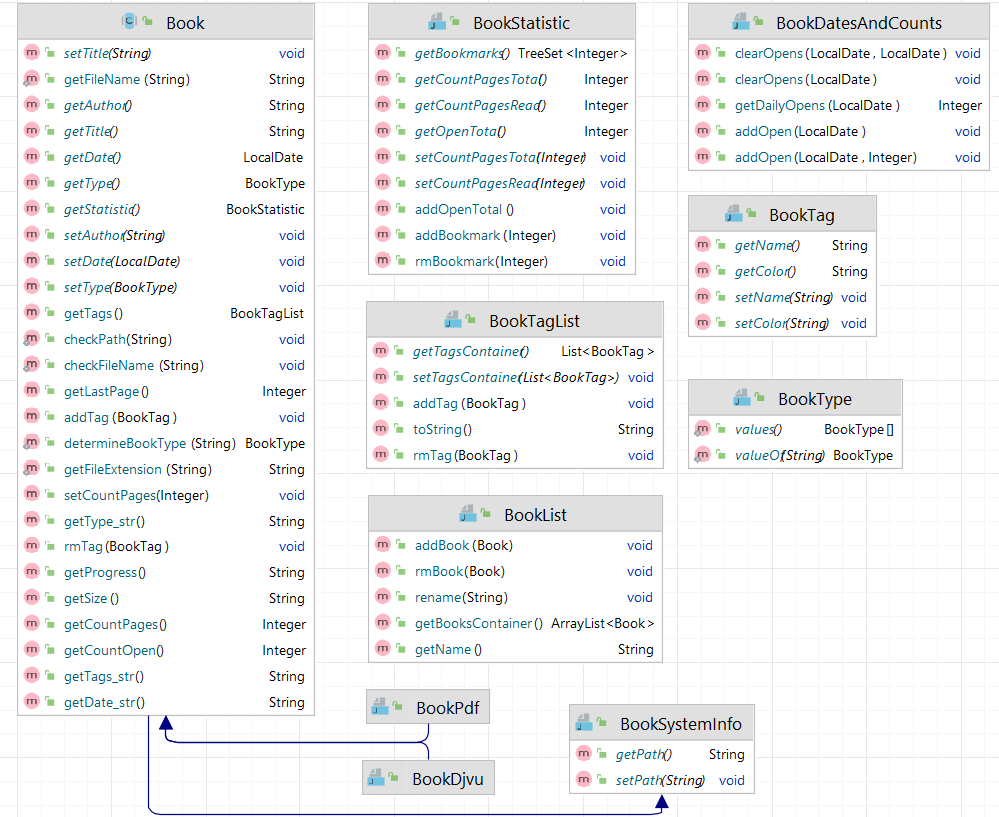
Зависимости:



Поля:



Методы:



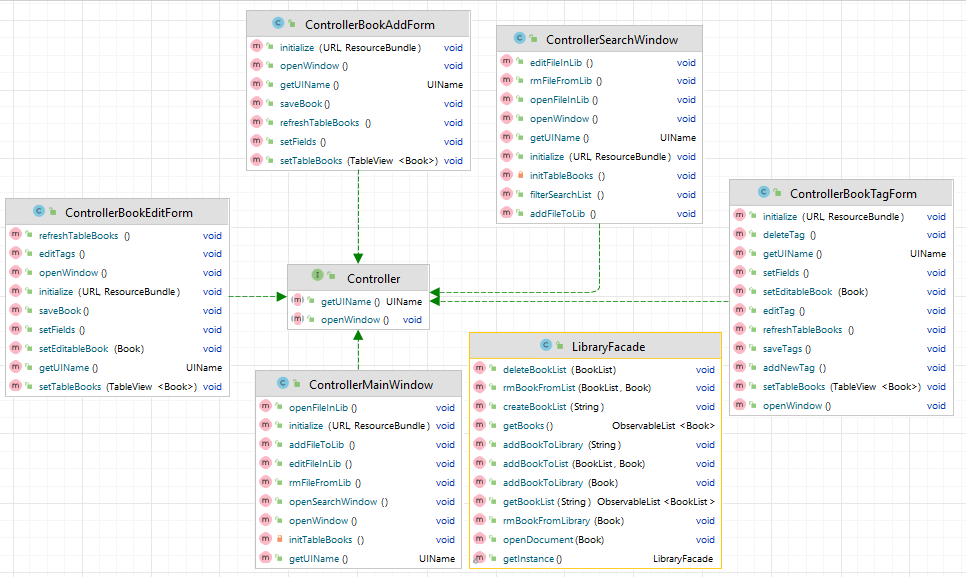
### Обоснование структуры классов

Основная информация содержится внутри класса Book, остальные расширяют его (BookStatistic), либо переопределяют некоторые его методы или свойства (BookPdf, BookDjvu)

Решение вынести статистику использования книги в отдельный класс было принято на основе идеи о экономии памяти. BookStatistic не инициализируется до тех пор пока для книги нет какой-либо статистики, а внутри класса Book хранится ссылка на null, а не на объект BookStatistic

Решение сделать несколько наследников для класса Book было принято для расширения его поведения (Можно сказать что в Book вынесена общая информация (которая дублировалась бы в наследниках) о книге, а в его наследниках уже специфическая для наследников)

### Управляющие классы



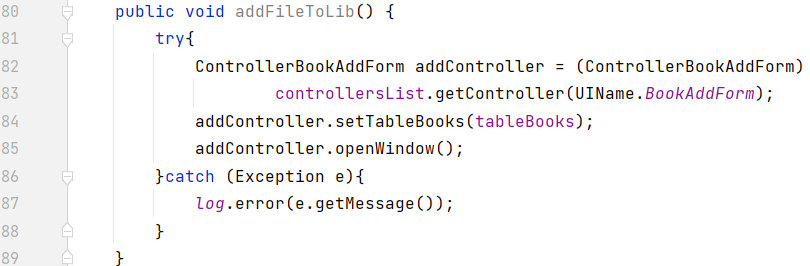
Действия пользователя обрабатываются контроллерами. По контроллеру на каждое окно.

Реализованы контроллеры для работы с книгами и метками, которые делегируют свою работу в фасад, а также реализуют интерфейс Controller, который позволяет вызывать окра друг из друга.

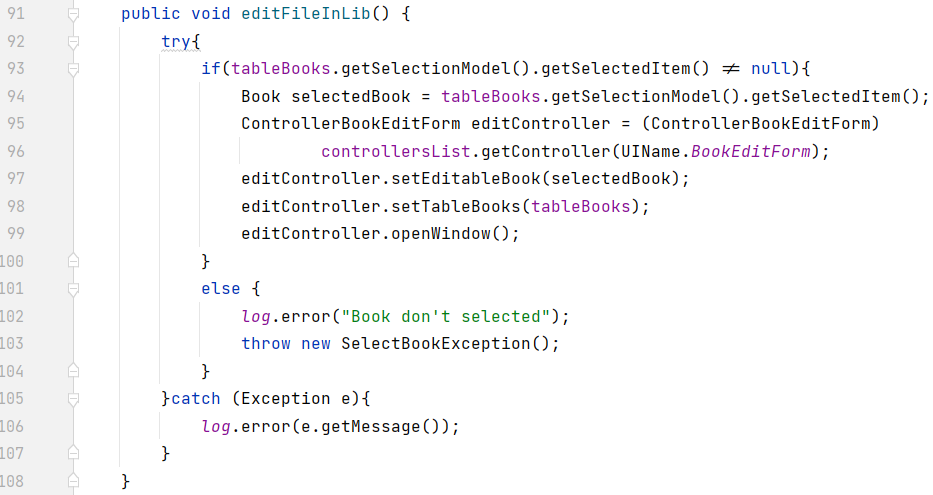
#### Основные функции



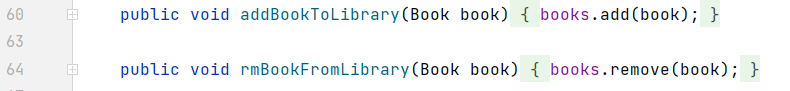
Функция открытия книги (ControllerMainWindow), которая делегирует свою работу фасаду



Функция добавления книги (ControllerBookAddForm), которая делегирует свою работу фасаду



Функция изменения книги (ControllerBookAddForm), которая делегирует свою работу фасаду



Функция добавления и удаления книг (Фасад библиотеки)

#### Функция фильтрации содержимого таблицы



# 4.Тестирование

Вызов контекстного меню

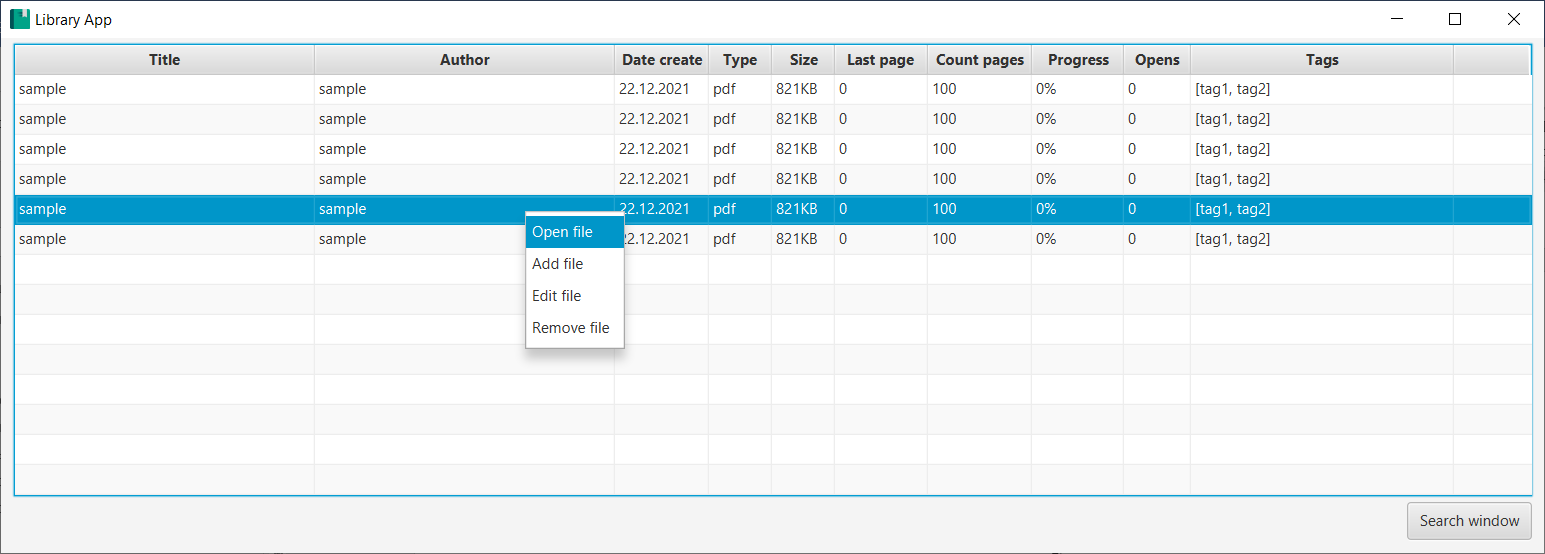


Рис. 8 Контекстное меню

Выбор опции «Открыть файл»

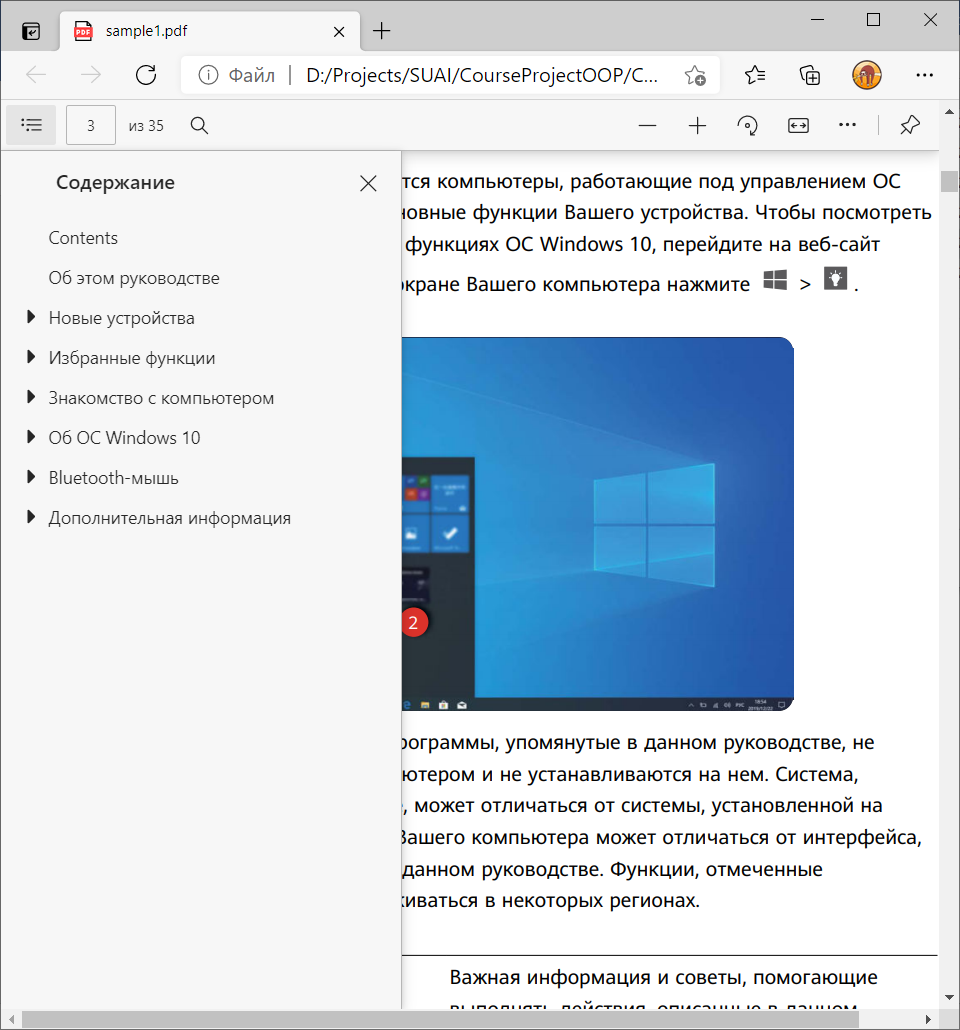


Рис. 9 Окно браузера

Количество открытий файла изменилось

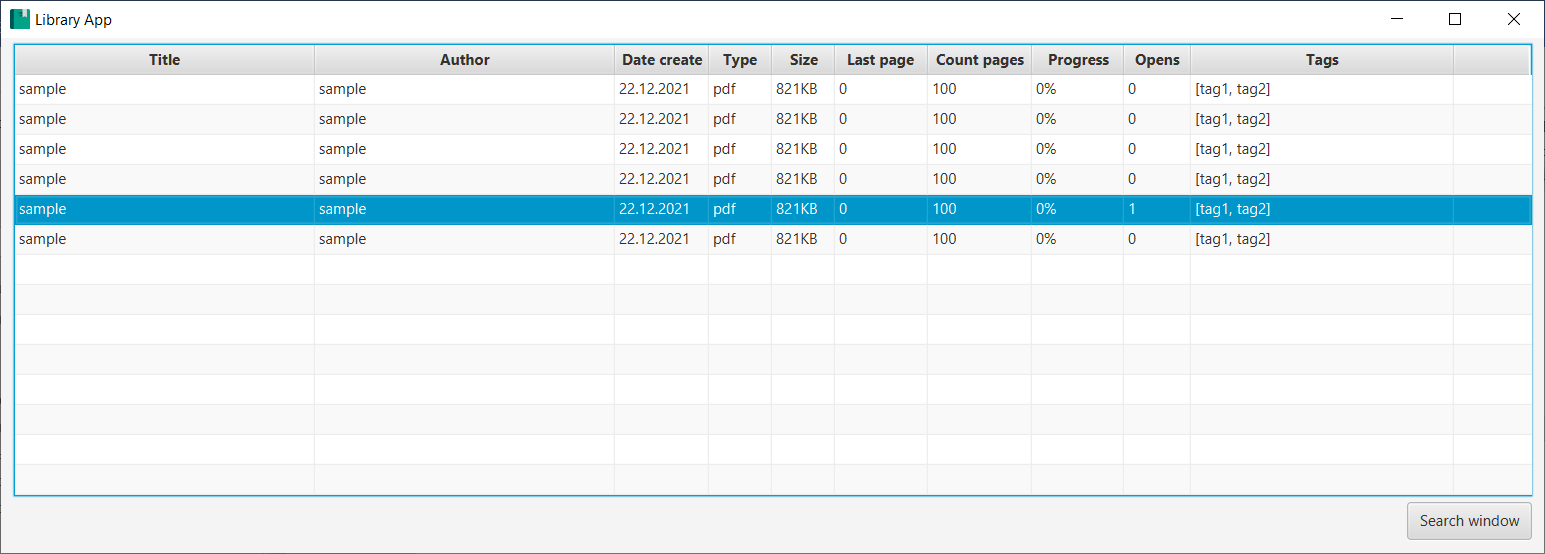


Рис. 10 Список кник

Выбор опции «Добавить файл»

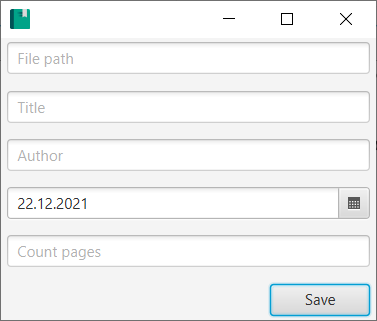


Рис. 11 Окно добавления файла

Заполнение полей

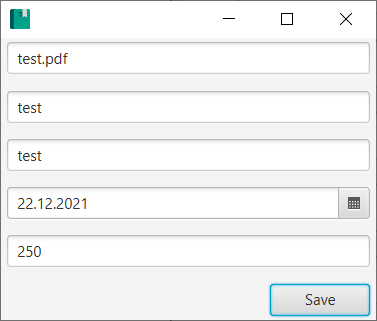


Рис. 12 Заполненные поля в окне добавления файла

При сохранении окно закрывается, проверяем список книг после добавления

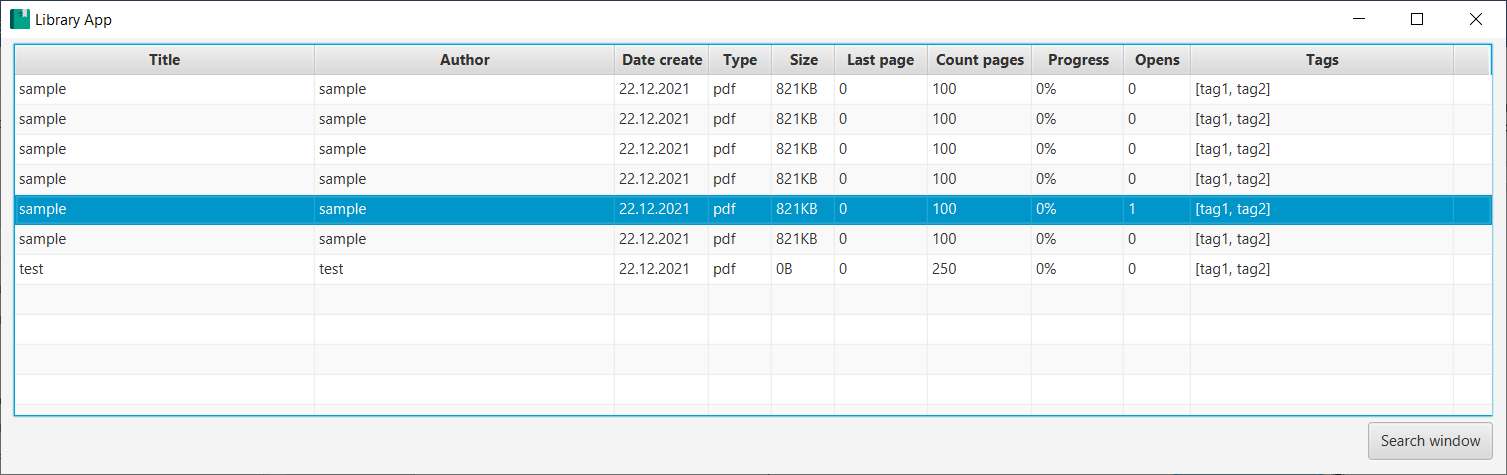


Рис. 13 Список книг после добавления

Выбор опции «Редактирование книги»

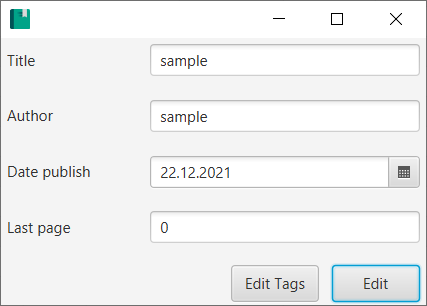


Рис. 14 Окно редактирования файла

Изменение данных и сохранение

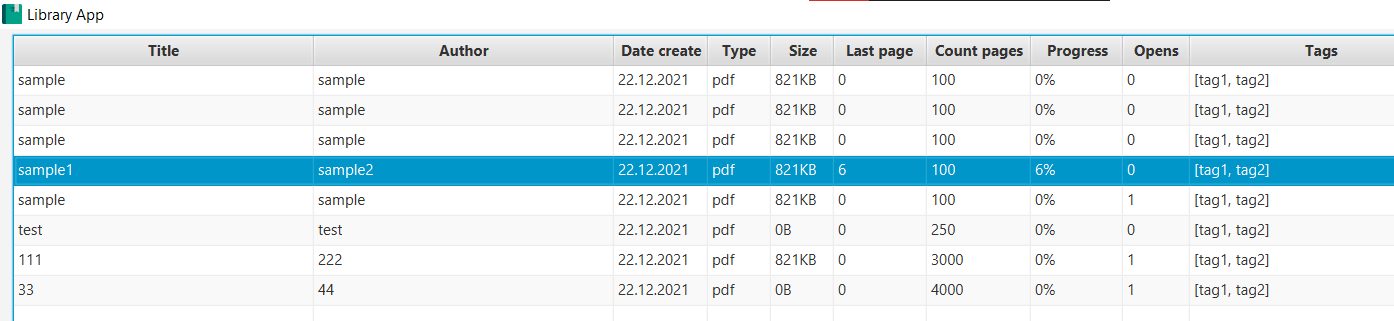


Рис 15. Список всех книг

Открытие окна редактирования меток (в меню редактирования книги)

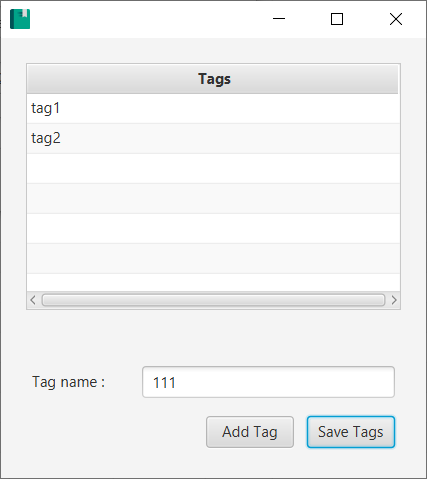


Рис. 16 Окно редактирования меток

Проверка списка книг

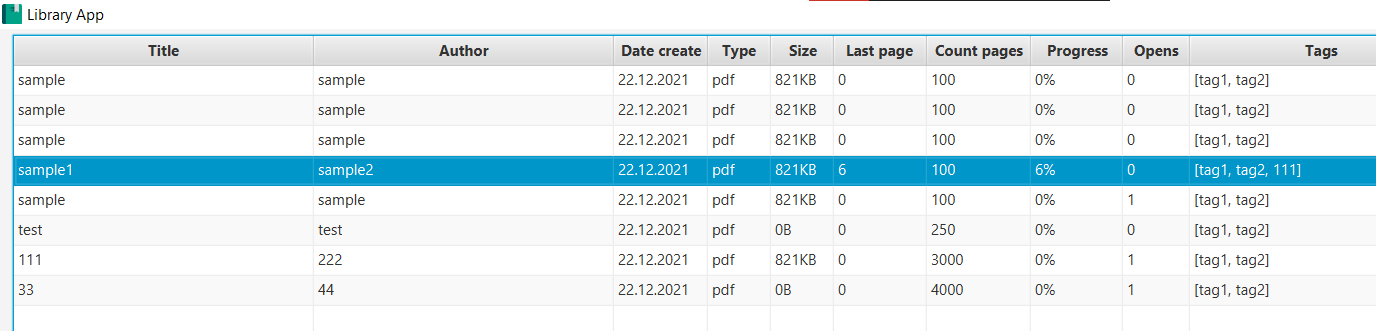


Рис. 17 Список книг с новой меткой

Выбор опции «Удаление» для книги с новым тегом

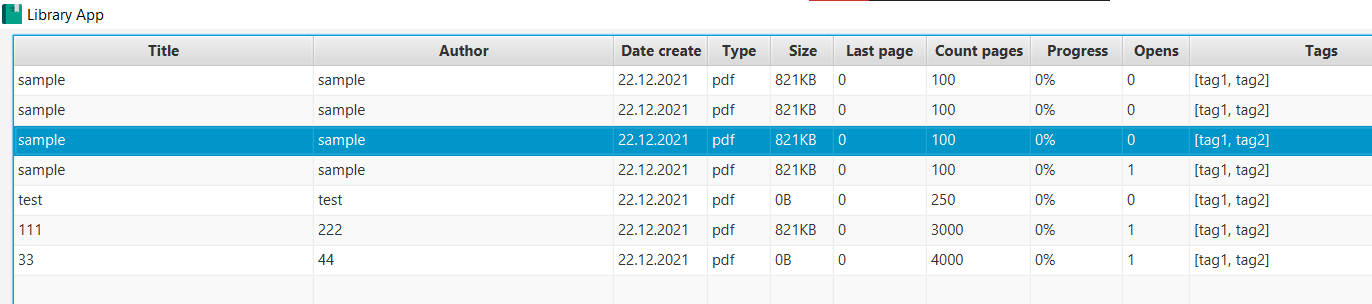


Рис. 18 Список книг после удаления

Поиск информации в библиотеке

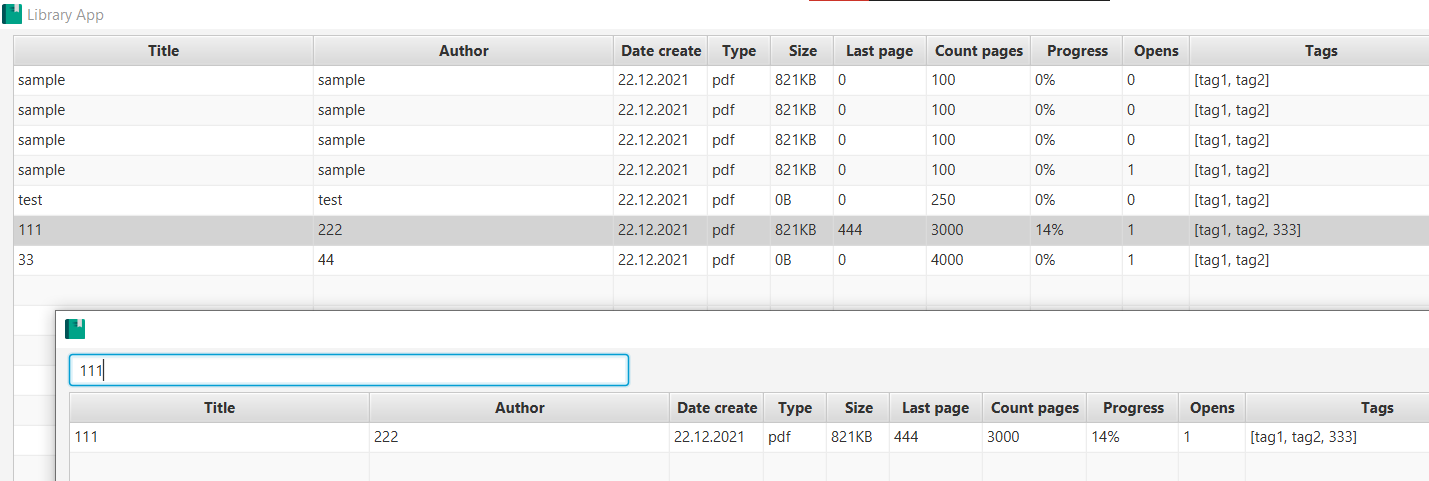


Рис. 19 Результат поиска запроса

Поиск информации в библиотеке

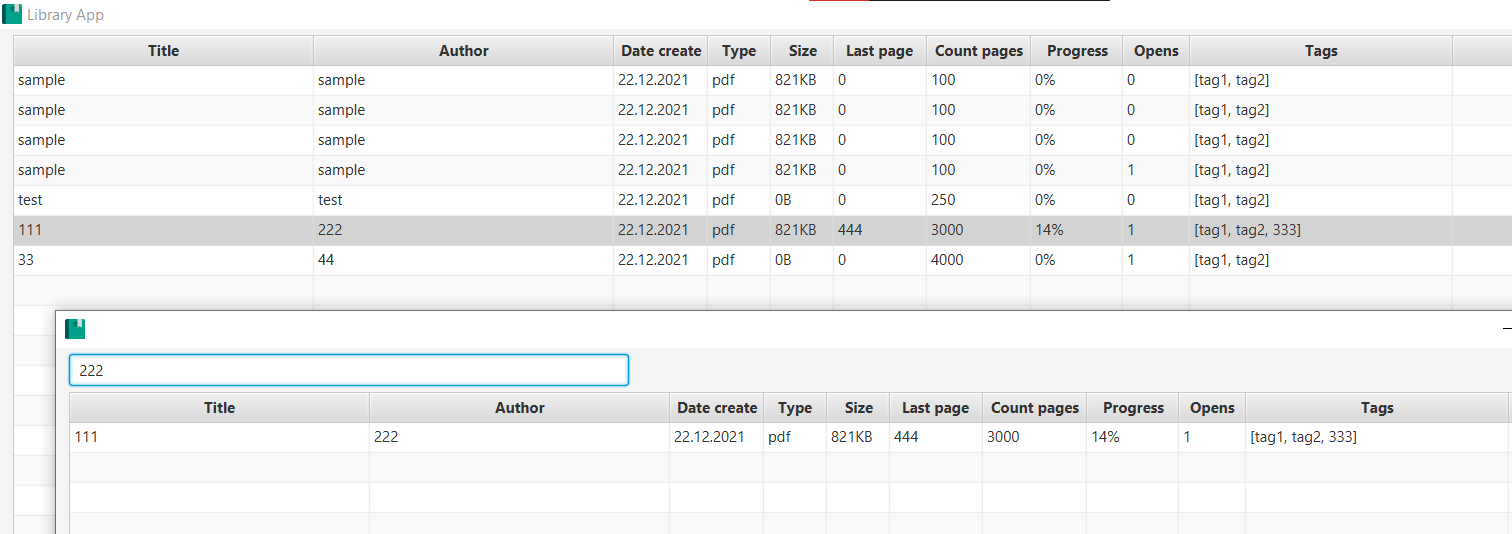


Рис. 20 Результат поиска запроса

Поиск информации в библиотеке

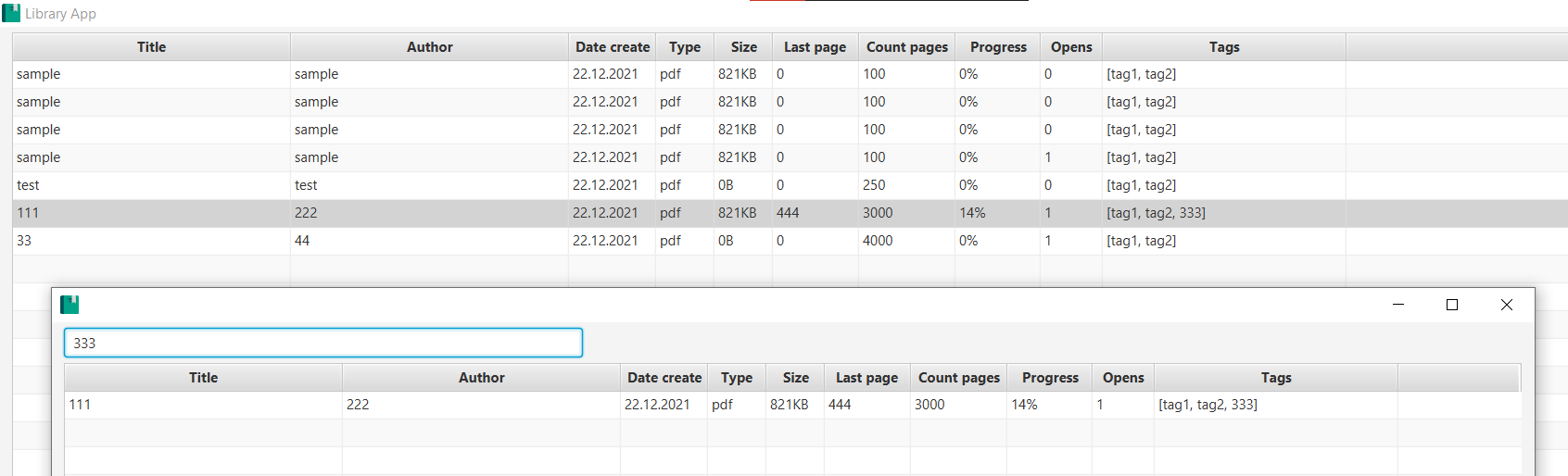


Рис. 21 Результат поиска запроса

Удаление книги из окна поиска

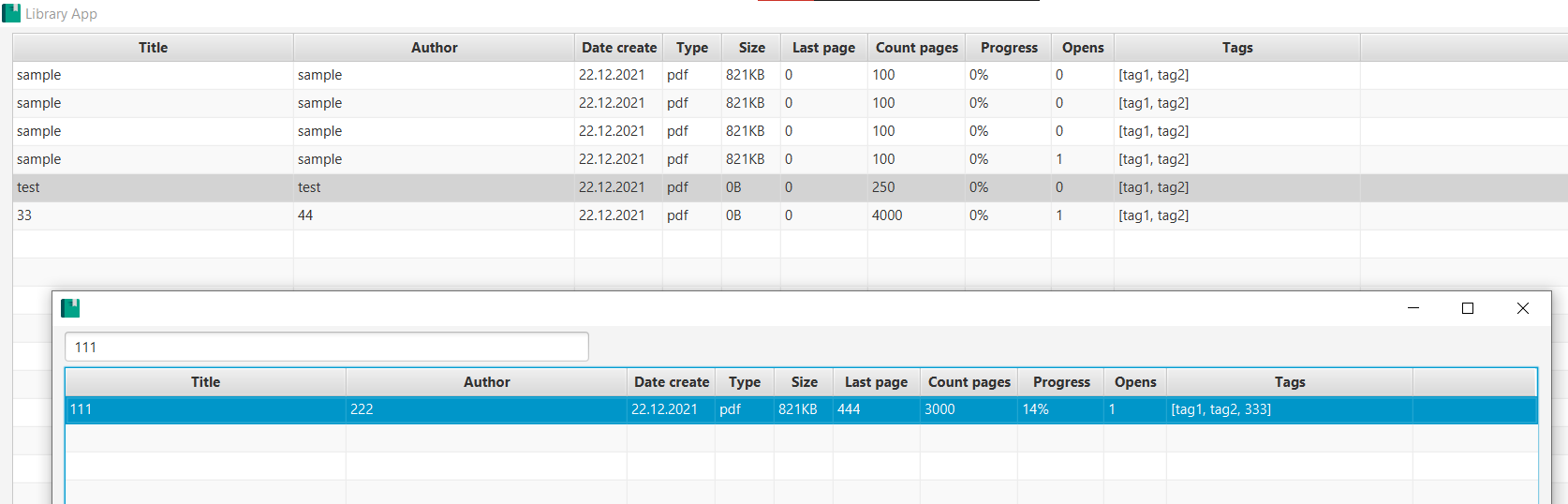


Рис. 22 Результат удаления

# ЗАКЛЮЧЕНИЕ

В ходе курсового проекта было разработано приложение, позволяющее оптимизировать работу пользователя с файлами, осуществлять добавление, удаление файлов, меток, поиск хранящейся информации.

Приложение осуществляет поставленную цель оптимизации разоты с файлами на основе объектно-ориентированного подхода. Я получил навыки реализации и проектирования схемы данных.

У приложения можно вывести следующие недостатки:

* Функциональность могла бы быть больше

Достоинства:

* Хороший поиск
* Не тормозящий интерфейс
* Кроссплатформенность

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Е.О.Шумова / Объектно-ориентированное программирование: Методические указания к выполнению курсового проекта - ГУАП. -СПб., 2021.
2. <https://stackoverflow.com/>
3. <https://docs.oracle.com/en/java/>
4. <https://metanit.com/java/>

# ПРИЛОЖЕНИЕ A

// @filename \src\main\java\module-info.java

module app.library {

requires javafx.controls;

requires javafx.fxml;

requires javafx.web;

requires org.controlsfx.controls;

requires com.dlsc.formsfx;

requires validatorfx;

requires org.kordamp.ikonli.javafx;

requires org.kordamp.bootstrapfx.core;

requires eu.hansolo.tilesfx;

requires org.apache.logging.log4j;

requires lombok;

opens app.library to javafx.fxml;

exports app.library;

exports app.exceptions;

opens app.exceptions to javafx.fxml;

exports app.library.book;

opens app.library.book to javafx.fxml;

exports app.library.controllers;

opens app.library.controllers to javafx.fxml;

exports app.library.factory;

opens app.library.factory to javafx.fxml;

exports app.library.util;

opens app.library.util to javafx.fxml;

exports app.library.util.controllers;

opens app.library.util.controllers to javafx.fxml;

exports app.library.util.windows;

opens app.library.util.windows to javafx.fxml;

exports app.exceptions.custom;

opens app.exceptions.custom to javafx.fxml;

}

// @filename \src\main\java\app\exceptions\ExceptionDisplayAlert.java

package app.exceptions;

public interface ExceptionDisplayAlert {

void showAlert(String message);

}

// @filename \src\main\java\app\exceptions\LibraryException.java

package app.exceptions;

public abstract class LibraryException

extends Exception

implements ExceptionDisplayAlert {

public LibraryException(String message) {

super(message);

showAlert(message);

}

}

// @filename \src\main\java\app\exceptions\LibraryExceptionError.java

package app.exceptions;

import javafx.scene.control.Alert;

import javafx.scene.control.ButtonType;

public abstract class LibraryExceptionError extends LibraryException {

public LibraryExceptionError(String message) {

super(message);

}

@Override

public void showAlert(String message) {

Alert alert = new Alert(

Alert.AlertType.ERROR,

this.getClass().getSimpleName() + " : " + message,

ButtonType.OK);

alert.showAndWait();

}

}

// @filename \src\main\java\app\exceptions\LibraryExceptionInfo.java

package app.exceptions;

import javafx.scene.control.Alert;

import javafx.scene.control.ButtonType;

public abstract class LibraryExceptionInfo extends LibraryException {

public LibraryExceptionInfo(String message) {

super(message);

}

@Override

public void showAlert(String message) {

Alert alert = new Alert(

Alert.AlertType.INFORMATION,

message,

ButtonType.OK);

alert.showAndWait();

}

}

// @filename \src\main\java\app\exceptions\LibraryExceptionWarning.java

package app.exceptions;

import javafx.scene.control.Alert;

import javafx.scene.control.ButtonType;

public abstract class LibraryExceptionWarning extends LibraryException {

public LibraryExceptionWarning(String message) {

super(message);

}

@Override

public void showAlert(String message) {

Alert alert = new Alert(

Alert.AlertType.WARNING,

message,

ButtonType.OK);

alert.showAndWait();

}

}

// @filename \src\main\java\app\exceptions\custom\CountPagesException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionError;

public class CountPagesException extends LibraryExceptionError {

public CountPagesException() {

super("Invalid count pages: count less 0, must be over or equal 0");

}

}

// @filename \src\main\java\app\exceptions\custom\FileNameException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionError;

public class FileNameException extends LibraryExceptionError {

public FileNameException() {

super("Invalid file name");

}

}

// @filename \src\main\java\app\exceptions\custom\FilePathException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionError;

public class FilePathException extends LibraryExceptionError {

public FilePathException() {

super("Invalid file path");

}

}

// @filename \src\main\java\app\exceptions\custom\FileTypeException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionError;

public class FileTypeException extends LibraryExceptionError {

public FileTypeException() {

super("Invalid file type");

}

}

// @filename \src\main\java\app\exceptions\custom\NoImplementedException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionInfo;

public class NoImplementedException extends LibraryExceptionInfo {

public NoImplementedException() {

super("Function don't implemented yet");

}

}

// @filename \src\main\java\app\exceptions\custom\NotFoundException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionError;

public class NotFoundException extends LibraryExceptionError {

public NotFoundException() {

super("Element not found");

}

}

// @filename \src\main\java\app\exceptions\custom\SelectBookException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionError;

public class SelectBookException extends LibraryExceptionError {

public SelectBookException() { super("Book don't selected"); }

}

// @filename \src\main\java\app\exceptions\custom\SelectProgramException.java

package app.exceptions.custom;

import app.exceptions.LibraryExceptionError;

public class SelectProgramException extends LibraryExceptionError {

public SelectProgramException() {

super("The program for this file (in settings) is selected incorrectly");

}

}

// @filename \src\main\java\app\library\LibraryApp.java

package app.library;

import app.library.util.UIName;

import app.library.util.windows.Window;

import app.library.util.windows.WindowsList;

import javafx.scene.control.Alert;

import javafx.scene.control.ButtonType;

import lombok.extern.log4j.Log4j2;

import org.apache.logging.log4j.Logger;

import org.apache.logging.log4j.LogManager;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.scene.image.Image;

import javafx.stage.Stage;

import java.io.IOException;

import java.util.List;

import java.util.Objects;

@Log4j2

public class LibraryApp extends Application {

private final WindowsList windowsList = WindowsList.getInstance();

public static void main(String[] args) {

log.debug("App run");

launch();

}

@Override

public void start(Stage mainWindow) throws Exception {

this.addWindows(mainWindow);

LibraryApp.setStyles(windowsList);

LibraryApp.setAppIcon(windowsList);

// set window properties

mainWindow.setMinHeight(300);

mainWindow.setMinWidth(500);

// mainWindow.setHeight(windowsList.getWindow(UIName.MainWindow).getPrefWinHeight());

// mainWindow.setWidth(windowsList.getWindow(UIName.MainWindow).getPrefWinWidth());

// mainWindow.setMaximized(windowsList.getWindow(UIName.MainWindow).getMaximized());

mainWindow.setTitle("Library App");

mainWindow.show();

log.debug("App display");

}

private void addWindows(Stage mainWindow) throws IOException {

Scene mainScene = loadFXML("ui/main-window.fxml");

mainWindow.setScene(mainScene);

Stage searchWindow = new Stage();

Scene searchWindowScene = loadFXML("ui/search-window.fxml");

searchWindow.setScene(searchWindowScene);

Stage bookAddForm = new Stage();

Scene bookAddScene = loadFXML("ui/book-add-form.fxml");

bookAddForm.setScene(bookAddScene);

Stage bookEditForm = new Stage();

Scene bookEditScene = loadFXML("ui/book-edit-form.fxml");

bookEditForm.setScene(bookEditScene);

Stage bookTagForm = new Stage();

Scene bookTagScene = loadFXML("ui/book-tag-form.fxml");

bookTagForm.setScene(bookTagScene);

windowsList.addWindow(new Window(mainWindow, UIName.MainWindow));

windowsList.addWindow(new Window(searchWindow, UIName.SearchWindow));

windowsList.addWindow(new Window(bookAddForm, UIName.BookAddForm));

windowsList.addWindow(new Window(bookEditForm, UIName.BookEditForm));

windowsList.addWindow(new Window(bookTagForm, UIName.BookTagForm));

}

private static void setStyles(WindowsList windows){

List<Scene> scenes = windows.getWindows().stream()

.filter(Objects::nonNull)

.map(Window::getScene).toList();

for(var scene : scenes){

scene.getStylesheets().add(

Objects.requireNonNull(LibraryApp.class.

getResource("styles/all-stylesheets-import.css")).toExternalForm());

}

}

private static void setAppIcon(WindowsList windows){

for(var window : windows.getWindows()) {

window.getStage().getIcons().add(new Image(

Objects.requireNonNull(LibraryApp.class.

getResourceAsStream("icon.png"))));

}

}

private static Scene loadFXML(String file) throws IOException {

return new Scene(new FXMLLoader(LibraryApp.class.getResource(file)).load());

}

}

// @filename \src\main\java\app\library\LibraryFacade.java

package app.library;

import app.exceptions.custom.\*;

import app.library.book.Book;

import app.library.book.BookList;

import app.library.factory.FactoryDjvu;

import app.library.factory.FactoryFile;

import app.library.factory.FactoryPdf;

import app.library.util.LibrarySettings;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.scene.control.Alert;

import javafx.scene.control.ButtonType;

import lombok.extern.log4j.Log4j2;

import java.io.IOException;

import java.util.Arrays;

@Log4j2

public class LibraryFacade {

private static volatile LibraryFacade instance;

private final LibrarySettings settings = LibrarySettings.getInstance();

private final ObservableList<Book> books = FXCollections.observableArrayList();

private final ObservableList<BookList> bookLists = FXCollections.observableArrayList();

private LibraryFacade(){}

public static LibraryFacade getInstance(){

LibraryFacade result = instance;

if (result != null) {

return result;

}

synchronized(LibraryFacade.class) {

if (instance == null) {

instance = new LibraryFacade();

}

return instance;

}

}

public void addBookToLibrary(String path) {

try{

FactoryFile factory;

switch (Book.determineBookType(path)){

case PDF -> factory = new FactoryPdf();

case DJVU -> factory = new FactoryDjvu();

default -> throw new FileTypeException();

}

Book book = factory.createBook();

book.setPath(path);

books.add(book);

}catch (Exception e){

log.error(e.getMessage());

}

}

public void addBookToLibrary(Book book) {

books.add(book);

}

public void rmBookFromLibrary(Book book){

books.remove(book);

}

public void createBookList(String name){

bookLists.add(new BookList(name));

}

public void deleteBookList(BookList bookList){

bookLists.remove(bookList);

}

public void addBookToList(BookList list, Book book){

list.addBook(book);

}

public void rmBookFromList(BookList list, Book book){

list.rmBook(book);

}

public ObservableList<Book> getBooks() {

return books;

}

public ObservableList<BookList> getBookList(String filter){

return bookLists;

}

public void openDocument(Book book){

try{

String program = settings.getProgramPathReader(book);

String filepath = book.getPath();

Runtime.getRuntime().exec(program + " --open " + filepath);

} catch (IOException | SelectProgramException e) {

Alert alert = new Alert(Alert.AlertType.ERROR, e.getMessage(), ButtonType.OK); alert.showAndWait();

}

}

}

// @filename \src\main\java\app\library\book\Book.java

package app.library.book;

import app.exceptions.custom.CountPagesException;

import app.exceptions.custom.FileNameException;

import app.exceptions.custom.FilePathException;

import app.exceptions.custom.FileTypeException;

import lombok.Getter;

import lombok.Setter;

import java.io.File;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.util.Arrays;

import java.util.List;

import java.util.Locale;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public abstract class Book extends BookSystemInfo {

@Setter @Getter protected String author;

@Setter @Getter protected String title;

@Setter @Getter protected LocalDate date;

@Setter @Getter protected BookType type;

@Getter protected BookStatistic statistic = new BookStatistic();

protected BookTagList tagsContainer = new BookTagList();

public String getDate\_str() {

return date.format(DateTimeFormatter.ofPattern("dd.MM.yyyy"));

}

public String getType\_str() {

return type.toString().toLowerCase();

}

public void addTag(BookTag tag){

tagsContainer.addTag(tag);

}

public void rmTag(BookTag tag){

tagsContainer.rmTag(tag);

}

public BookTagList getTags(){

return tagsContainer;

}

public String getTags\_str(){

return tagsContainer.toString();

}

public void setCountPages(Integer countPages) throws CountPagesException {

if(countPages > 0){

this.statistic.setCountPagesTotal(countPages);

}

else {

throw new CountPagesException();

}

}

public Integer getCountPages() throws CountPagesException {

if(this.statistic.getCountPagesTotal() >= 0){

return this.statistic.getCountPagesTotal();

}

else {

throw new CountPagesException();

}

}

public String getProgress() throws CountPagesException {

return ( (statistic.getCountPagesRead() \* 100) / this.getCountPages() ) + "%";

}

public Integer getCountOpen(){

return statistic.getOpenTotal();

}

public Integer getLastPage(){

return statistic.getCountPagesRead();

}

public String getSize() {

long bytes = 0;

long kilobytes = 0;

long megabytes = 0;

long gigabytes = 0;

File file = new File(this.getPath());

if(file.exists()) {

bytes = file.length();

kilobytes = bytes / 1024;

megabytes = kilobytes / 1024;

gigabytes = megabytes / 1024;

}

List<Long> size = Arrays.asList(gigabytes, megabytes, kilobytes, bytes);

List<String> sign = Arrays.asList("GB", "MB", "KB", "B");

for (var s : size) {

if(s > 0){

return s + sign.get(size.indexOf(s));

}

}

return "0B";

}

public static void checkFileName(String filename) throws FileNameException {

// bad symbols: \/:\*?"<>|

Pattern pattern = Pattern.compile("[\\\\\\/\\:\\\*\\?\\\"\\<\\>/|]");

Matcher matcher = pattern.matcher(filename);

if(matcher.find()){

throw new FileNameException();

}

}

public static void checkPath(String path) throws FilePathException {

Matcher rightSlash = Pattern.compile("\\/").matcher(path);

Matcher leftSlash = Pattern.compile("\\\\").matcher(path);

Matcher startPointer = Pattern.compile("^\\.").matcher(path);

if(rightSlash.find() && leftSlash.find() || startPointer.find()){

throw new FilePathException();

}

}

public static String getFileName(String path) throws FilePathException, FileNameException, FileTypeException {

checkPath(path);

int from = path.lastIndexOf("/");

if (from == -1) { from = path.lastIndexOf("\\"); }

from += 1;

int to = path.lastIndexOf(".");

if(to == -1){ throw new FileTypeException(); }

String filename = path.substring(from, to);

checkFileName(filename);

return filename;

}

public static String getFileExtension(String path) throws FilePathException, FileNameException, FileTypeException {

checkPath(path);

checkFileName(getFileName(path));

Pattern pattern = Pattern.compile("\\.\\w+");

Matcher matcher = pattern.matcher(path);

if(matcher.find()){

return matcher.group(0);

}

throw new FileTypeException();

}

public static BookType determineBookType(String path) throws FilePathException, FileNameException, FileTypeException {

String fileExtension = getFileExtension(path).toLowerCase(Locale.ROOT);

if(fileExtension.equals(".pdf")){

return BookType.PDF;

}

else if(fileExtension.equals(".djvu")){

return BookType.DJVU;

}

else {

throw new FileTypeException();

}

}

}

// @filename \src\main\java\app\library\book\BookDatesAndCounts.java

package app.library.book;

import java.time.LocalDate;

import java.util.HashMap;

public class BookDatesAndCounts {

public final HashMap<LocalDate, Integer> container = new HashMap<>();

public void addOpen(LocalDate date){

addOpen(date, 1);

}

public void addOpen(LocalDate date, Integer count){

if(container.containsKey(date)){

container.put(date, container.get(date)+count);

}

else{

container.put(date, count);

}

}

public void clearOpens(LocalDate date){

container.remove(date);

}

public void clearOpens(LocalDate from, LocalDate to){

while (from.isBefore(to)){

container.remove(from);

from = from.plusDays(1);

}

}

public Integer getDailyOpens(LocalDate date){

return container.getOrDefault(date, 0);

}

}

// @filename \src\main\java\app\library\book\BookDjvu.java

package app.library.book;

import app.exceptions.custom.NoImplementedException;

public class BookDjvu extends Book {

public BookDjvu() throws NoImplementedException {

throw new NoImplementedException();

}

}

// @filename \src\main\java\app\library\book\BookList.java

package app.library.book;

import java.util.ArrayList;

public class BookList {

private final ArrayList<Book> booksContainer = new ArrayList<>();

private String bookListName;

public BookList(String name){

bookListName = name;

}

public void addBook(Book book){

booksContainer.add(book);

}

public void rmBook(Book book){

booksContainer.remove(book);

}

public void rename(String name){

bookListName = name;

}

public String getName(){

return bookListName;

}

public ArrayList<Book> getBooksContainer() {

return booksContainer;

}

}

// @filename \src\main\java\app\library\book\BookPdf.java

package app.library.book;

import java.time.LocalDate;

public class BookPdf extends Book {

public BookPdf(){

tagsContainer = new BookTagList();

statistic = new BookStatistic();

this.setPath("undefined");

author = "sample";

title = "sample";

date = LocalDate.now();

type = BookType.PDF;

statistic.setCountPagesTotal(100);

tagsContainer.addTag(new BookTag("tag1"));

tagsContainer.addTag(new BookTag("tag2"));

}

public BookPdf(String path){

tagsContainer = new BookTagList();

statistic = new BookStatistic();

this.setPath(path);

author = "default";

title = "default";

date = LocalDate.now();

type = BookType.PDF;

statistic.setCountPagesTotal(100);

tagsContainer.addTag(new BookTag("tag1"));

tagsContainer.addTag(new BookTag("tag2"));

tagsContainer.addTag(new BookTag("tag3"));

}

}

// @filename \src\main\java\app\library\book\BookStatistic.java

package app.library.book;

import lombok.Getter;

import lombok.Setter;

import java.time.LocalDate;

import java.util.TreeSet;

public class BookStatistic {

@Getter private final TreeSet<Integer> bookmarks = new TreeSet<>();

private final BookDatesAndCounts openStatistic;

@Getter @Setter private Integer countPagesTotal;

@Getter @Setter private Integer countPagesRead;

@Getter private Integer openTotal;

public BookStatistic() {

this.openStatistic = new BookDatesAndCounts();

this.countPagesTotal = 1;

this.countPagesRead = 0;

this.openTotal = 0;

}

public void addBookmark(Integer page){

bookmarks.add(page);

}

public void rmBookmark(Integer page){

bookmarks.remove(page);

}

public void addOpenTotal(){

openTotal += 1;

openStatistic.addOpen(LocalDate.now());

}

}

// @filename \src\main\java\app\library\book\BookSystemInfo.java

package app.library.book;

import lombok.Getter;

import lombok.Setter;

public class BookSystemInfo {

@Getter @Setter private String path;

}

// @filename \src\main\java\app\library\book\BookTag.java

package app.library.book;

import lombok.Getter;

import lombok.Setter;

public class BookTag {

@Getter @Setter private String name;

@Getter @Setter private String color;

public BookTag(){

this.name = "undefined";

this.color = "#a1a1a1";

}

public BookTag(String name){

this.name = name;

this.color = "#a1a1a1";

}

public BookTag(String name, String color){

this.name = name;

this.color = color;

}

}

// @filename \src\main\java\app\library\book\BookTagList.java

package app.library.book;

import lombok.Getter;

import lombok.Setter;

import java.util.LinkedList;

import java.util.List;

public class BookTagList {

@Getter @Setter private List<BookTag> tagsContainer = new LinkedList<>();

public void addTag(BookTag tag){

tagsContainer.add(tag);

}

public void rmTag(BookTag tag){

tagsContainer.remove(tag);

}

@Override

public String toString() {

return tagsContainer

.stream()

.map(BookTag::getName)

.toList()

.toString();

}

}

// @filename \src\main\java\app\library\book\BookType.java

package app.library.book;

public enum BookType { PDF, DJVU }

// @filename \src\main\java\app\library\controllers\ControllerBookAddForm.java

package app.library.controllers;

import app.exceptions.custom.\*;

import app.library.LibraryFacade;

import app.library.book.Book;

import app.library.factory.FactoryDjvu;

import app.library.factory.FactoryFile;

import app.library.factory.FactoryPdf;

import app.library.util.UIName;

import app.library.util.controllers.Controller;

import app.library.util.controllers.ControllerList;

import app.library.util.windows.WindowsList;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.DatePicker;

import javafx.scene.control.TableView;

import javafx.scene.control.TextField;

import lombok.extern.log4j.Log4j2;

import java.net.URL;

import java.time.LocalDate;

import java.util.Arrays;

import java.util.ResourceBundle;

@Log4j2

public class ControllerBookAddForm implements Initializable, Controller {

protected final LibraryFacade lib = LibraryFacade.getInstance();

protected final WindowsList windowsList = WindowsList.getInstance();

protected final ControllerList controllersList = ControllerList.getInstance();

@FXML protected TextField fieldFilepath;

@FXML protected TextField fieldTitle;

@FXML protected TextField fieldAuthor;

@FXML protected TextField fieldCountPages;

@FXML protected DatePicker fieldDate;

@FXML protected Button buttonSave;

@FXML protected TableView<Book> tableBooks;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

controllersList.addController(this);

}

public void setFields(){

fieldFilepath.setText("");

fieldTitle.setText("");

fieldAuthor.setText("");

fieldCountPages.setText("");

fieldDate.setValue(LocalDate.now());

}

public void setTableBooks(TableView<Book> table){

tableBooks = table;

}

public void refreshTableBooks(){

if(tableBooks != null){

tableBooks.refresh();

}

}

public void saveBook() {

try {

Book book;

FactoryFile factory;

switch (Book.determineBookType(fieldFilepath.getText())){

case PDF -> factory = new FactoryPdf();

case DJVU -> factory = new FactoryDjvu();

default -> throw new FileTypeException();

}

book = factory.createBook();

book.setPath(fieldFilepath.getText());

book.setTitle(fieldTitle.getText());

book.setAuthor(fieldAuthor.getText());

book.setCountPages(Integer.parseInt(fieldCountPages.getText()));

book.setDate(fieldDate.getValue());

lib.addBookToLibrary(book);

refreshTableBooks();

windowsList.getWindowStage(UIName.BookAddForm).close();

} catch (Exception e) {

log.error(e.getMessage());

}

}

@Override

public void openWindow(){

this.setFields();

try {

windowsList.getWindowStage(UIName.BookAddForm).show();

} catch (NotFoundException e) {

e.printStackTrace();

}

}

@Override

public UIName getUIName() {

return UIName.BookAddForm;

}

}

// @filename \src\main\java\app\library\controllers\ControllerBookEditForm.java

package app.library.controllers;

import app.exceptions.custom.NotFoundException;

import app.exceptions.custom.SelectBookException;

import app.library.LibraryFacade;

import app.library.util.UIName;

import app.library.util.controllers.Controller;

import app.library.util.controllers.ControllerList;

import app.library.util.windows.WindowsList;

import app.library.book.Book;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import lombok.extern.log4j.Log4j2;

import java.net.URL;

import java.util.ResourceBundle;

@Log4j2

public class ControllerBookEditForm implements Initializable, Controller {

protected final LibraryFacade lib = LibraryFacade.getInstance();

protected final WindowsList windowsList = WindowsList.getInstance();

protected final ControllerList controllersList = ControllerList.getInstance();

@FXML protected TextField fieldAuthor;

@FXML protected TextField fieldTitle;

@FXML protected DatePicker fieldDate;

@FXML protected TextField fieldLastPage;

@FXML protected Button buttonSave;

@FXML protected Button buttonEditTags;

@FXML protected TableView<Book> tableBooks;

protected Book editableBook;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

controllersList.addController(this);

}

public void setFields(){

fieldTitle.setText(editableBook.getTitle());

fieldAuthor.setText(editableBook.getAuthor());

fieldDate.setValue(editableBook.getDate());

fieldLastPage.setText(editableBook.getStatistic().getCountPagesRead().toString());

}

public void setEditableBook(Book book){

this.editableBook = book;

}

public void setTableBooks(TableView<Book> table){

this.tableBooks = table;

}

public void refreshTableBooks(){

if(tableBooks != null){

tableBooks.refresh();

}

}

public void editTags(){

try{

if(tableBooks.getSelectionModel().getSelectedItem() != null){

Book selectedBook = this.editableBook;

ControllerBookTagForm editController = (ControllerBookTagForm)

controllersList.getController(UIName.BookTagForm);

editController.setEditableBook(selectedBook);

editController.setTableBooks(tableBooks);

editController.openWindow();

}

else {

log.error("Book don't selected");

throw new SelectBookException();

}

}catch (Exception e){

log.error(e.getMessage());

}

}

public void saveBook() {

editableBook.setTitle(fieldTitle.getText());

editableBook.setAuthor(fieldAuthor.getText());

editableBook.setDate(fieldDate.getValue());

editableBook.getStatistic().setCountPagesRead(Integer.parseInt(fieldLastPage.getText()));

refreshTableBooks();

try {

windowsList.getWindowStage(UIName.BookEditForm).close();

} catch (NotFoundException e) {

e.printStackTrace();

}

}

@Override

public void openWindow(){

this.setFields();

try {

windowsList.getWindowStage(UIName.BookEditForm).show();

} catch (NotFoundException e) {

e.printStackTrace();

}

}

@Override

public UIName getUIName() {

return UIName.BookEditForm;

}

}

// @filename \src\main\java\app\library\controllers\ControllerBookTagForm.java

package app.library.controllers;

import app.exceptions.custom.NotFoundException;

import app.library.book.Book;

import app.library.book.BookTag;

import app.library.util.UIName;

import app.library.util.controllers.Controller;

import app.library.util.controllers.ControllerList;

import app.library.util.windows.WindowsList;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import java.net.URL;

import java.util.ResourceBundle;

public class ControllerBookTagForm implements Initializable, Controller {

protected final WindowsList windowsList = WindowsList.getInstance();

protected final ControllerList controllersList = ControllerList.getInstance();

@FXML protected MenuItem menuItemDeleteTag;

@FXML protected MenuItem menuItemEditTag;

@FXML protected TextField fieldTagName;

@FXML protected Button buttonAddTag;

@FXML protected Button buttonSaveTags;

@FXML protected TableView<BookTag> tagTable;

@FXML protected TableColumn<BookTag, String> tableColumnTags;

protected ObservableList<BookTag> tagObservableList = FXCollections.observableArrayList();

@FXML protected TableView<Book> tableBooks;

protected Book editableBook;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

controllersList.addController(this);

tableColumnTags.setCellValueFactory(new PropertyValueFactory<>("name"));

tagTable.setItems(tagObservableList);

}

public void setFields() {

tagObservableList.clear();

tagObservableList.addAll(editableBook.getTags().getTagsContainer());

tagTable.refresh();

}

public void setEditableBook(Book book){

this.editableBook = book;

}

public void setTableBooks(TableView<Book> table){

this.tableBooks = table;

}

public void refreshTableBooks(){

if(tableBooks != null){

tableBooks.refresh();

}

}

public void addNewTag(){

BookTag tag = new BookTag();

tag.setName(fieldTagName.getText());

tagObservableList.add(tag);

tagTable.refresh();

}

public void deleteTag(){

if(tableBooks.getSelectionModel().getSelectedItem() != null) {

BookTag selectedTag = tagTable.getSelectionModel().getSelectedItem();

tagObservableList.remove(selectedTag);

tagTable.refresh();

}

}

public void editTag(){

if(tableBooks.getSelectionModel().getSelectedItem() != null) {

BookTag selectedTag = tagTable.getSelectionModel().getSelectedItem();

selectedTag.setName(fieldTagName.getText());

tagTable.refresh();

}

}

public void saveTags() {

editableBook.getTags().setTagsContainer(

tagObservableList.stream().toList()

);

try {

refreshTableBooks();

windowsList.getWindowStage(UIName.BookTagForm).close();

} catch (NotFoundException e) {

e.printStackTrace();

}

}

@Override

public void openWindow(){

this.setFields();

try {

windowsList.getWindowStage(UIName.BookTagForm).show();

} catch (NotFoundException e) {

e.printStackTrace();

}

}

@Override

public UIName getUIName() {

return UIName.BookTagForm;

}

}

// @filename \src\main\java\app\library\controllers\ControllerMainWindow.java

package app.library.controllers;

import app.exceptions.custom.\*;

import app.library.\*;

import app.library.book.Book;

import app.library.book.BookTagList;

import app.library.book.BookType;

import app.library.util.UIName;

import app.library.util.controllers.Controller;

import app.library.util.controllers.ControllerList;

import app.library.util.windows.WindowsList;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import lombok.extern.log4j.Log4j2;

import java.net.URL;

import java.time.LocalDate;

import java.util.Arrays;

import java.util.ResourceBundle;

@Log4j2

public class ControllerMainWindow implements Controller, Initializable {

private final LibraryFacade lib = LibraryFacade.getInstance();

private final WindowsList windowsList = WindowsList.getInstance();

private final ControllerList controllersList = ControllerList.getInstance();

@FXML protected MenuItem menuItemOpenFileInLib;

@FXML protected MenuItem menuItemAddFileToLib;

@FXML protected MenuItem menuItemEditFileInLib;

@FXML protected MenuItem menuItemRmFileFromLib;

@FXML protected TableView<Book> tableBooks;

@FXML protected TableColumn<Book, String> titleColumn;

@FXML protected TableColumn<Book, String> authorColumn;

@FXML protected TableColumn<Book, LocalDate> dateCreateColumn;

@FXML protected TableColumn<Book, BookType> typeColumn;

@FXML protected TableColumn<Book, Integer> sizeColumn;

@FXML protected TableColumn<Book, BookTagList> tagsColumn;

@FXML protected TableColumn<Book, String> progressColumn;

@FXML protected TableColumn<Book, Integer> countOpenColumn;

@FXML protected TableColumn<Book, Integer> countPagesColumn;

@FXML protected TableColumn<Book, Integer> lastPageColumn;

@FXML protected Button buttonSearchWindow;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

controllersList.addController(this);

initTableBooks();

}

private void initTableBooks(){

titleColumn.setCellValueFactory(new PropertyValueFactory<>("title"));

authorColumn.setCellValueFactory(new PropertyValueFactory<>("author"));

dateCreateColumn.setCellValueFactory(new PropertyValueFactory<>("date\_str"));

typeColumn.setCellValueFactory(new PropertyValueFactory<>("type\_str"));

sizeColumn.setCellValueFactory(new PropertyValueFactory<>("size"));

tagsColumn.setCellValueFactory(new PropertyValueFactory<>("tags\_str"));

progressColumn.setCellValueFactory(new PropertyValueFactory<>("progress"));

countOpenColumn.setCellValueFactory(new PropertyValueFactory<>("countOpen"));

countPagesColumn.setCellValueFactory(new PropertyValueFactory<>("countPages"));

lastPageColumn.setCellValueFactory(new PropertyValueFactory<>("lastPage"));

tableBooks.setItems(lib.getBooks());

for(int i = 0; i < 6;i++){

lib.addBookToLibrary(

"D:\\Projects\\SUAI\\CourseProjectOOP\\CourseProjectOOP-hot-release" +

"\\src\\main\\resources\\app\\library\\samples\\sample1.pdf");

}

}

public void addFileToLib() {

try{

ControllerBookAddForm addController = (ControllerBookAddForm)

controllersList.getController(UIName.BookAddForm);

addController.setTableBooks(tableBooks);

addController.openWindow();

}catch (Exception e){

log.error(e.getMessage());

}

}

public void editFileInLib() {

try{

if(tableBooks.getSelectionModel().getSelectedItem() != null){

Book selectedBook = tableBooks.getSelectionModel().getSelectedItem();

ControllerBookEditForm editController = (ControllerBookEditForm)

controllersList.getController(UIName.BookEditForm);

editController.setEditableBook(selectedBook);

editController.setTableBooks(tableBooks);

editController.openWindow();

}

else {

log.error("Book don't selected");

throw new SelectBookException();

}

}catch (Exception e){

log.error(e.getMessage());

}

}

public void rmFileFromLib() {

try {

if(tableBooks.getSelectionModel().getSelectedItem() != null){

Book selectedBook = tableBooks.getSelectionModel().getSelectedItem();

lib.rmBookFromLibrary(selectedBook);

}

else {

log.error("Book don't selected");

throw new SelectBookException();

}

}catch (Exception e){

log.error(e.getMessage());

}

}

public void openFileInLib() {

try{

if(tableBooks.getSelectionModel().getSelectedItem() != null){

Book selectedBook = tableBooks.getSelectionModel().getSelectedItem();

selectedBook.getStatistic().addOpenTotal();

tableBooks.refresh();

lib.openDocument(selectedBook);

}

else {

log.error("not selected book");

throw new SelectBookException();

}

}catch (Exception e){

log.error(e.getMessage());

}

}

public void openSearchWindow() throws NotFoundException {

ControllerSearchWindow editController = (ControllerSearchWindow)

controllersList.getController(UIName.SearchWindow);

editController.openWindow();

}

@Override

public void openWindow() {

try {

windowsList.getWindowStage(UIName.MainWindow).show();

} catch (NotFoundException e) {

e.printStackTrace();

}

}

@Override

public UIName getUIName() {

return UIName.MainWindow;

}

}

// @filename \src\main\java\app\library\controllers\ControllerSearchWindow.java

package app.library.controllers;

import app.exceptions.custom.NotFoundException;

import app.exceptions.custom.SelectBookException;

import app.library.LibraryFacade;

import app.library.book.Book;

import app.library.book.BookTagList;

import app.library.book.BookType;

import app.library.util.UIName;

import app.library.util.controllers.Controller;

import app.library.util.controllers.ControllerList;

import app.library.util.windows.WindowsList;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import lombok.extern.log4j.Log4j2;

import java.net.URL;

import java.time.LocalDate;

import java.util.Arrays;

import java.util.ResourceBundle;

@Log4j2

public class ControllerSearchWindow implements Initializable, Controller {

private final LibraryFacade lib = LibraryFacade.getInstance();

private final WindowsList windowsList = WindowsList.getInstance();

private final ControllerList controllersList = ControllerList.getInstance();

@FXML protected MenuItem menuItemOpenFileInLib;

@FXML protected MenuItem menuItemEditFileInLib;

@FXML protected MenuItem menuItemRmFileFromLib;

@FXML protected TableView<Book> tableBooks;

@FXML protected TableColumn<Book, String> titleColumn;

@FXML protected TableColumn<Book, String> authorColumn;

@FXML protected TableColumn<Book, LocalDate> dateCreateColumn;

@FXML protected TableColumn<Book, BookType> typeColumn;

@FXML protected TableColumn<Book, Integer> sizeColumn;

@FXML protected TableColumn<Book, BookTagList> tagsColumn;

@FXML protected TableColumn<Book, String> progressColumn;

@FXML protected TableColumn<Book, Integer> countOpenColumn;

@FXML protected TableColumn<Book, Integer> countPagesColumn;

@FXML protected TableColumn<Book, Integer> lastPageColumn;

@FXML protected TextField fieldSearch;

protected ObservableList<Book> globalList = FXCollections.observableArrayList();

protected ObservableList<Book> searchList = FXCollections.observableArrayList();

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

controllersList.addController(this);

initTableBooks();

globalList.clear();

searchList.clear();

globalList = lib.getBooks();

searchList = globalList;

this.tableBooks.setItems(searchList);

}

private void initTableBooks(){

titleColumn.setCellValueFactory(new PropertyValueFactory<>("title"));

authorColumn.setCellValueFactory(new PropertyValueFactory<>("author"));

dateCreateColumn.setCellValueFactory(new PropertyValueFactory<>("date\_str"));

typeColumn.setCellValueFactory(new PropertyValueFactory<>("type\_str"));

sizeColumn.setCellValueFactory(new PropertyValueFactory<>("size"));

tagsColumn.setCellValueFactory(new PropertyValueFactory<>("tags\_str"));

progressColumn.setCellValueFactory(new PropertyValueFactory<>("progress"));

countOpenColumn.setCellValueFactory(new PropertyValueFactory<>("countOpen"));

countPagesColumn.setCellValueFactory(new PropertyValueFactory<>("countPages"));

lastPageColumn.setCellValueFactory(new PropertyValueFactory<>("lastPage"));

}

public void addFileToLib() {

try{

ControllerBookAddForm addController = (ControllerBookAddForm)

controllersList.getController(UIName.BookAddForm);

addController.setTableBooks(tableBooks);

addController.openWindow();

}catch (Exception e){

log.error(e.getMessage());

}

}

public void editFileInLib() {

try{

if(tableBooks.getSelectionModel().getSelectedItem() != null){

Book selectedBook = tableBooks.getSelectionModel().getSelectedItem();

ControllerBookEditForm editController = (ControllerBookEditForm)

controllersList.getController(UIName.BookEditForm);

editController.setEditableBook(selectedBook);

editController.setTableBooks(tableBooks);

editController.openWindow();

}

else {

log.error("Book don't selected");

throw new SelectBookException();

}

}catch (Exception e){

log.error(e.getMessage());

}

}

public void rmFileFromLib() {

try {

if(tableBooks.getSelectionModel().getSelectedItem() != null){

Book selectedBook = tableBooks.getSelectionModel().getSelectedItem();

lib.rmBookFromLibrary(selectedBook);

}

else {

log.error("Book don't selected");

throw new SelectBookException();

}

}catch (Exception e){

log.error(e.getMessage());

}

}

public void openFileInLib() {

try{

if(tableBooks.getSelectionModel().getSelectedItem() != null){

Book selectedBook = tableBooks.getSelectionModel().getSelectedItem();

selectedBook.getStatistic().addOpenTotal();

tableBooks.refresh();

lib.openDocument(selectedBook);

}

else {

log.error("not selected book");

throw new SelectBookException();

}

}catch (Exception e){

log.error(e.getMessage());

}

}

public void filterSearchList(){

String request = fieldSearch.getText();

if (request == null

|| request.equals("")

|| request.equals(" ")) {

searchList = globalList;

return;

}

searchList = FXCollections.observableArrayList(

globalList

.stream()

.filter(book -> {

return book.getTitle().contains(request)

|| book.getAuthor().contains(request)

|| book.getDate().toString().contains(request)

|| book.getType().toString().contains(request)

|| book.getTags().toString().contains(request);

})

.toList()

);

this.tableBooks.refresh();

this.tableBooks.setItems(searchList);

this.tableBooks.refresh();

log.debug("filter searched list");

log.debug("size request = " + searchList.size());

}

@Override

public void openWindow() {

try {

windowsList.getWindowStage(UIName.SearchWindow).show();

} catch (NotFoundException e) {

e.printStackTrace();

}

}

@Override

public UIName getUIName() {

return UIName.SearchWindow;

}

}

// @filename \src\main\java\app\library\factory\FactoryDjvu.java

package app.library.factory;

import app.exceptions.custom.NoImplementedException;

import app.library.book.Book;

public class FactoryDjvu implements FactoryFile {

public FactoryDjvu() throws NoImplementedException {

throw new NoImplementedException();

}

@Override

public Book createBook() { return null; }

}

// @filename \src\main\java\app\library\factory\FactoryFile.java

package app.library.factory;

import app.library.book.Book;

public interface FactoryFile {

Book createBook();

}

// @filename \src\main\java\app\library\factory\FactoryPdf.java

package app.library.factory;

import app.library.book.Book;

import app.library.book.BookPdf;

public class FactoryPdf implements FactoryFile {

@Override

public Book createBook() {

return new BookPdf();

}

}

// @filename \src\main\java\app\library\util\LibrarySettings.java

package app.library.util;

import app.exceptions.custom.FilePathException;

import app.exceptions.custom.SelectProgramException;

import app.library.book.Book;

import app.library.book.BookType;

import app.library.util.windows.WindowsList;

import java.util.HashMap;

public class LibrarySettings {

private static volatile LibrarySettings instance;

private String sourceRoot = "D:\\Programming\\Projects\\SUAI\\CourseProjectOOP\\";

private final WindowsList windowsList = WindowsList.getInstance();

public final HashMap<BookType, String> programs = new HashMap<>();

// List displayedColumns

private LibrarySettings(){

programs.put(BookType.PDF, "C:\\Program Files (x86)\\Microsoft\\Edge\\Application\\msedge.exe");

}

public static LibrarySettings getInstance(){

LibrarySettings result = instance;

if (result != null) {

return result;

}

synchronized(LibrarySettings.class) {

if (instance == null) {

instance = new LibrarySettings();

}

return instance;

}

}

public String getSourceRoot() {

return sourceRoot;

}

public void setSourceRoot(String sourceRoot) throws FilePathException {

Book.checkPath(sourceRoot);

this.sourceRoot = sourceRoot;

}

public WindowsList getWindowsList() {

return windowsList;

}

public String getProgramPathReader(Book book) throws SelectProgramException {

String programName = programs.get(book.getType());

if (programName == null){

throw new SelectProgramException();

}

return programName;

}

public void setProgramPathReader(Book book, String programPath){

programs.put(book.getType(), programPath);

}

public void setProgramPathReader(BookType bookType, String programPath){

programs.put(bookType, programPath);

}

}

// @filename \src\main\java\app\library\util\UIName.java

package app.library.util;

public enum UIName {

MainWindow,

SearchWindow,

BookEditForm,

BookAddForm,

BookTagForm

}

// @filename \src\main\java\app\library\util\controllers\Controller.java

package app.library.util.controllers;

import app.library.util.UIName;

public interface Controller {

abstract public void openWindow();

abstract public UIName getUIName();

}

// @filename \src\main\java\app\library\util\controllers\ControllerList.java

package app.library.util.controllers;

import app.exceptions.custom.NotFoundException;

import app.library.util.LibrarySettings;

import app.library.util.UIName;

import lombok.Getter;

import java.util.LinkedList;

import java.util.List;

import java.util.Objects;

public class ControllerList {

// <Singleton code>

private static volatile ControllerList instance;

private ControllerList(){}

public static ControllerList getInstance(){

ControllerList result = instance;

if (result != null) {

return result;

}

synchronized(LibrarySettings.class) {

if (instance == null) {

instance = new ControllerList();

}

return instance;

}

}

// </Singleton code>

@Getter private final List<Controller> controllers = new LinkedList<>();

public void addController(Controller c){

controllers.add(c);

}

public Controller getController(UIName name) throws NotFoundException {

for (Controller controller : controllers) {

if (Objects.equals(controller.getUIName(), name)) {

return controller;

}

}

throw new NotFoundException();

}

}

// @filename \src\main\java\app\library\util\windows\Window.java

package app.library.util.windows;

import app.library.util.UIName;

import javafx.scene.Scene;

import javafx.stage.Stage;

import lombok.Getter;

import lombok.Setter;

public class Window {

@Getter private final UIName UIName;

@Getter private final Stage window;

@Getter @Setter private Boolean maximized;

@Getter @Setter private Double prefWinHeight;

@Getter @Setter private Double prefWinWidth;

public Window(Stage window, UIName name) {

this.UIName = name;

this.window = window;

this.maximized = false;

this.prefWinHeight = 720.0;

this.prefWinWidth = 1200.0;

}

public Stage getStage() {

return window;

}

public Scene getScene() {

return window.getScene();

}

}

// @filename \src\main\java\app\library\util\windows\WindowsList.java

package app.library.util.windows;

import app.exceptions.custom.NotFoundException;

import app.library.util.LibrarySettings;

import app.library.util.UIName;

import javafx.stage.Stage;

import lombok.Getter;

import java.util.LinkedList;

import java.util.List;

import java.util.Objects;

public class WindowsList {

// <Singleton code>

private static volatile WindowsList instance;

private WindowsList(){}

public static WindowsList getInstance(){

WindowsList result = instance;

if (result != null) {

return result;

}

synchronized(LibrarySettings.class) {

if (instance == null) {

instance = new WindowsList();

}

return instance;

}

}

// </Singleton code>

@Getter private final List<Window> windows = new LinkedList<>();

public void addWindow(Window win){

windows.add(win);

}

public Window getWindow(UIName name) throws NotFoundException {

for(var window : windows){

if(Objects.equals(window.getUIName(), name)){

return window;

}

}

throw new NotFoundException();

}

public Stage getWindowStage(UIName name) throws NotFoundException {

return this.getWindow(name).getStage();

}

}

// @filename \src\test\java\app\tests\MyTests.java

package app.tests;

import java.time.LocalDate;

import app.exceptions.custom.FileNameException;

import app.exceptions.custom.FilePathException;

import app.exceptions.custom.FileTypeException;

import app.exceptions.custom.NoImplementedException;

import app.library.book.Book;

import app.library.book.BookType;

import app.library.factory.FactoryDjvu;

import app.library.factory.FactoryFile;

import app.library.factory.FactoryPdf;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

public class MyTests {

private static final Logger log = LogManager.getLogger();

Book book;

@BeforeEach

@DisplayName("Init book with factory")

void init() throws FilePathException, FileNameException, FileTypeException, NoImplementedException {

FactoryFile factory;

switch (Book.determineBookType("app/library/samples/sample1.pdf")){

case PDF -> factory = new FactoryPdf();

case DJVU -> factory = new FactoryDjvu();

default -> throw new FileTypeException();

}

book = factory.createBook();

assertEquals(book.getType(), BookType.PDF);

}

@Test

@DisplayName("Date Comparing")

public void dateCompare() {

LocalDate from = LocalDate.now();

LocalDate to = LocalDate.now().plusDays(1);

assertTrue(from.isBefore(to));

}

@Test

@DisplayName("Determine type Book")

public void determineBookType() throws Exception {

// examples for correct paths

assertEquals(BookType.PDF, Book.determineBookType("path\\to\\file.pdf\n"));

assertEquals(BookType.PDF, Book.determineBookType("path/to/file.pdf\n"));

assertEquals(BookType.PDF, Book.determineBookType("file.pdf\n"));

assertEquals(BookType.PDF, Book.determineBookType("file.pdf \n"));

assertEquals(BookType.DJVU, Book.determineBookType("file.djvu\n"));

// remove "\n"

assertEquals(BookType.PDF, Book.determineBookType("path\\to\\file.pdf"));

assertEquals(BookType.PDF, Book.determineBookType("path/to/file.pdf"));

assertEquals(BookType.PDF, Book.determineBookType("file.pdf"));

assertEquals(BookType.PDF, Book.determineBookType("file.pdf "));

assertEquals(BookType.DJVU, Book.determineBookType("file.djvu"));

}

@Test

@DisplayName("Invalid Book file path : empty string")

public void bookPathEmptyString() {

assertThrows(FileTypeException.class, () -> Book.determineBookType(""));

assertThrows(FileTypeException.class, () -> Book.determineBookType("\n"));

}

@Test

@DisplayName("Invalid Book file path : invalid slash")

public void bookPathSlash() {

assertThrows(FilePathException.class, () -> Book.determineBookType("path/to\\file.pdf"));

assertThrows(FilePathException.class, () -> Book.determineBookType("path\\to/file.pdf"));

assertThrows(FilePathException.class, () -> Book.determineBookType("path/to\\file.pdf\n"));

assertThrows(FilePathException.class, () -> Book.determineBookType("path\\to/file.pdf\n"));

}

@Test

@DisplayName("Invalid Book file path : start point")

public void bookPathPoint() {

assertThrows(FilePathException.class, () -> Book.determineBookType("."));

assertThrows(FilePathException.class, () -> Book.determineBookType(".p"));

assertThrows(FilePathException.class, () -> Book.determineBookType(".\\..\\folder\\file.pdf\n"));

assertThrows(FilePathException.class, () -> Book.determineBookType("./../folder/file.pdf\n"));

assertThrows(FilePathException.class, () -> Book.determineBookType(".\\file.pdf\n"));

assertThrows(FilePathException.class, () -> Book.determineBookType("./file.pdf\n"));

assertThrows(FilePathException.class, () -> Book.determineBookType(".\\..\\folder\\file.pdf"));

assertThrows(FilePathException.class, () -> Book.determineBookType("./../folder/file.pdf"));

assertThrows(FilePathException.class, () -> Book.determineBookType(".\\file.pdf"));

assertThrows(FilePathException.class, () -> Book.determineBookType("./file.pdf"));

}

@Test

@DisplayName("Invalid Book file name : without pointer")

public void bookTypeWithoutPointer() {

assertThrows(FileTypeException.class, () -> Book.determineBookType("path\\to\\file"));

}

@Test

@DisplayName("Invalid Book file type")

public void bookType() {

assertThrows(FileTypeException.class, () -> Book.determineBookType("file.p"));

assertThrows(FileTypeException.class, () -> Book.determineBookType("path\\to\\file.p"));

}

}