МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

ФАКУЛЬТЕТ Информационных технологий и компьютерной безопасности

КАФЕДРА Систем управления и информационных технологий в строительстве

**КУРСОВОЙ ПРОЕКТ**

**по дисциплине**: «Обьектно-ориентированное программирование»

**на тему**: «Разработка приложения «Сбор сведений о писателях и их литературных произведениях» с применением языка программирования Java»

**Выполнил (-а):** Малова В. А.

**студент(ка)** 4 **курса**, **группы** збПИЭ211 .

**направления подготовки /специальность**

прикладная информатика .

**направленность (профиль) /специализация**

прикладная информатика в экономике цифрового общества .

заочная **форма обучения**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Руководитель работы:**

Минакова О.Е., к.т.н., доцент каф. СУИТС .

(Ф.И.О., должность, кафедра)

**Работа допущена к защите** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(подпись руководителя) (дата)

**Работа выполнена и защищена с оценкой** \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(оценка) (дата)

**Члены комиссии:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность) (подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность) (подпись) (Ф.И.О.)

Воронеж 2025

**УТВЕРЖДАЮ**

И.о. заведующего кафедрой

СУИТС

(наименование кафедры)

Аснина Н.Г.

(Ф.И.О.)

|  |  |  |
| --- | --- | --- |
| **Факультет** | Информационных технологий и компьютерной безопасности | |
| **Кафедра** | Систем управления и информационных технологий в | |
| строительстве | | |
| **Направление подготовки (специальность)** | | Прикладная информатика |
| **направленность (профиль) /специализация** | | Прикладная информатика в |
| экономике цифрового общества | | |

**Задание**

**на курсовую работу**

|  |  |
| --- | --- |
| **Студента** |  |

(Ф.И.О.)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **по дисциплине:** | | | | Обьектно-ориентированное программирование | | | | | | |
|  |  | | | | | | | | | |
| **1. Тема работы:** | | | | Разработка приложения «Сбор сведений о писателях и их | | | | | | |
|  | литературных произведениях» с применением языка программирования | | | | | | | | | |
|  | Java | | | | | | | | | |
| **2. Цель** | | Разработать приложение согласно выбранной предметной области | | | | | | | | |
|  |  | | | | | | | | | |
| **3. Задачи** | | | 1. Произвести моделирование классов предметной области. | | | | | | | |
|  | 1. Спроектировать приложение. | | | | | | | | | |
|  | 1. Реализовать приложение. | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
| **4. Перечень подлежащих разработке вопросов:** | | | | | | | | | |  |
| а) по теоретической части | | | | |  | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
| б) по аналитической части | | | | |  | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
| **5. Исходные данные:** | | | | | | | | | |  |
| а) по литературным источникам | | | | | |  | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
| б) по вариантам, разработанным преподавателем | | | | | | | |  | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
| в) иное | |  | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
| **6. Список рекомендуемых источников** | | | | | | |  | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
|  |  | | | | | | | | | |
| **7. Контрольные сроки представления** отдельных **разделов курсовой работы:** | | | | | | | | | |  |
| 25% − | | анализ и постановка задачи | | | | | | | « » 202 г. | |
| 50% − | | изучение теоретического обоснования | | | | | | | « » 202 г. | |
| работы | | | | | | |
| 75% − | | реализация программного решения | | | | | | | « » 202 г. | |
| 100% − | | оформление пояснительной записки | | | | | | | « » 202 г. | |

**8. Срок защиты студентом курсовой работы** « » 202 г.

**Дата выдачи задания** « » 202 г.

**Руководитель курсовой работы**

к.т.н., доцент Минакова О.В. .

(ученая степень, звание) (подпись) (Ф.И.О.)

**Задание принял(а) к исполнению студент(ка)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

формы обучения заочная курса 4 группы збПИЭ-211

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (Ф.И.О.)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ………………………………………………………………………..5

1. Моделирование классов предметной области………………………………..7
2. Проектирование приложения...........................................................................12
   1. Разработка графического пользовательского интерфейса (GUI)..............12
   2. Построение диаграмм деятельности приложения.......................................15
   3. Построение диаграмм последовательности приложения...........................17
   4. Диаграмма классов всего приложения.........................................................19
3. Реализация приложения...................................................................................22
   1. Структура программного проекта…………………………………………22

3.2.Описание классов приложения……………………………………………..23

ЗАКЛЮЧЕНИЕ………………………………………………………………….39

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ……………………………40

ПРИЛОЖЕНИЕ………………………………………………………………….41

ВВЕДЕНИЕ

Целью данного курсового проекта является получение навыков самостоятельной разработки приложения на основе методов, изученных рамках курса "Объектно-ориентированное программирование", являющихся основой для структурирования, проектирования и реализации программного обеспечения с применением принципов ООП для выбранной предметной области. В качестве темы для выполнения курсового проекта была выбрана разработка приложения «Сбор сведений о писателях и их литературных произведениях» с применением языка программирования Java.

Тема «Сбор сведений о писателях и их литературных произведениях» актуальна благодаря своей значимости для сохранения культуры, поддержки образования, удовлетворения информационных потребностей общества, а также для развития технологических навыков. Приложение, созданное на основе этой темы, может стать полезным инструментом для широкой аудитории и внести вклад в сохранение и популяризацию литературного наследия.

Основные преимущества использования приложения перечислены ниже:

* Управление данными: в настоящее время существует огромное количество книг, авторов и изданий, и потребность в эффективных инструментах для управления этими данными постоянно растёт.
* Быстрый и точный поиск: пользователям нужен удобный способ быстро находить нужные произведения, авторов и издания.
* Систематизация и организация: подобное приложение позволяет упорядочить данные и создать единую базу знаний, что повышает их ценность.
* Персонализация: возможность создания персональных коллекций и списков для чтения делает приложение более удобным для индивидуального использования.

Основными задачами, при разработке приложения по данной теме будут следующие:

1. Произвести моделирование классов предметной области.
2. Спроектировать приложение.
3. Реализовать приложение.

1. Моделирование классов предметной области

Приложение для сбора сведений о писателях и их литературных произведениях должно упрощать создание хранение обработку и поиск информации о писателях и созданных ими произведениях поэтому для его эффективного использования нужно корректно и качественно прописать классы которые будут отвечать за те или иные характеристики произведений и данных о писателе.

Составим вербальное описание классов которые будет включать наше приложение.

Класс *Author* (Автор) будет представлять информацию о писателе или авторе произведения.

Атрибуты:

* *author\_id* будет автоинкрементным полем с типом данных UUID позволяющим создавать уникальные идентификационные ключи для объектов класса.
* *name* (Имя автора). Атрибут будет иметь тип данных  String, поскольку имя — это текстовая информация.
* c*ountry* (Страна): страна, в которой проживает автор. Тип Country выбран для установления связи с объектами класса Country.

Класс *Work* (Произведение) представляет информацию о литературном произведении.

Атрибуты:

* *work\_id* будет автоинкрементным полем с типом данных UUID позволяющим создавать уникальные идентификационные ключи для объектов класса.
* *title* (Название произведения): Атрибут будет иметь тип данных String, поскольку название — это текстовая информация.
* *author* (Автор произведения): Тип Author выбран для установления связи с объектами класса Author.
* *genre* (Жанр произведения):Тип Genre выбран для установления связи с объектами класса Genre.
* *language* (язык произведения): Тип Language выбран для установления связи с объектами класса Language.

Класс *Genre* (Жанр) представляет информацию о жанре литературного произведения.

Атрибуты:

* *genre\_id* будет автоинкрементным полем с типом данных UUID позволяющим создавать уникальные идентификационные ключи для объектов класса.
* *genreName* (Название жанра):  Атрибут будет иметь тип данных String, поскольку название жанра — это текстовая информация.

Класс *Country* (Страна)будет представлять информацию о стране, в которой проживает автор.

Атрибуты:

* *country\_id* будет автоинкрементным полем с типом данных UUID позволяющим создавать уникальные идентификационные ключи для объектов класса.
* *countryName* (Название страны): Атрибут будет иметь тип данных String, поскольку название страны — это текстовая информация.

Класс *Language* (Язык) предоставляет информацию о языке, на котором было написано или переведено произведение

Атрибуты:

* *language\_id* будет автоинкрементным полем с типом данных UUID позволяющим создавать уникальные идентификационные ключи для объектов класса.
* *languageName* (Название языка): Атрибут будет иметь тип данных String, поскольку название языка — это текстовая информация.

Класс *Edition* (Издание) предоставляет информацию об отдельном издании произведения.

Атрибуты:

* *edition\_id* будет автоинкрементным полем с типом данных UUID позволяющим создавать уникальные идентификационные ключи для объектов класса.
* *work* (Work): Ссылка на произведение, к которому относится данное издание. Тип Work выбран для установления связи с объектами класса Work.
* *publisher* (Издатель): Издательство, выпустившее данное издание. Тип Publisher выбран для установления связи с объектами класса Publisher.
* *printingQuantity* (int): Тираж издания (количество экземпляров). Тип int выбран, поскольку тираж — это целое число.

Класс *Publisher* (Издатель) представляет информацию об издательстве, выпустившем произведение.

Атрибуты:

* *publisher\_id* будет автоинкрементным полем с типом данных UUID позволяющим создавать уникальные идентификационные ключи для объектов класса.
* *publisherName* (Название издательства): Атрибут будет иметь тип данных String, поскольку название издательства — это текстовая информация.

Связи между классами будут представлены в виде отношений.

Отношения между классами *Counrty* и *Author* будут представлены в виде отношений агрегации.

Отношения между классами *Author* и *Work* будут представлены в виде отношений композиции.

Отношения между классами *Language* и *Work* будут представлены в виде отношений ассоциации.

Отношения между классами *Work* и *Genre* будут представлены в виде отношений агрегации.

Отношения между классами *Work* и *Edition* будут представлены в виде отношений композиции.

Отношения между классами *Publisher* и *Edition* будут представлены в виде отношений композиции.

Изобразим диаграмму классов предметной области используя заданные нами отношения и классы(рис.1).

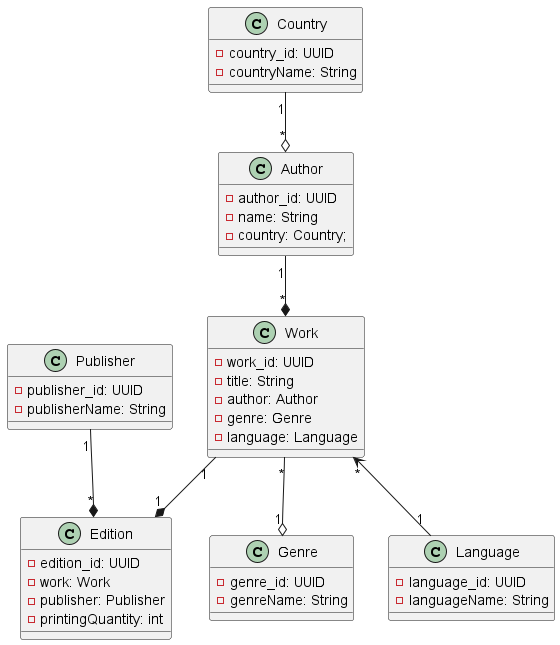


Рисунок 1 - Диаграмма классов предметной области.

**2**  Проектирование приложения

2.1 Разработка графического пользовательского интерфейса (GUI)

Существует множество способов взаимодействия пользователя с программным продуктом, таких как графический интерфейс пользователя (GUI), командная строка, голосовые команды, сенсорные экраны и другие[1].

Для приложения на тему "Сбор сведений о писателях и их литературных произведениях" наилучшим выбором будет графический интерфейс пользователя (GUI).

GUI обеспечивает пользователю интуитивно понятное и удобное взаимодействие с программой с помощью графических элементов, таких как кнопки, текстовые поля, меню и др. Это позволяет пользователям легко ориентироваться в приложении, быстро освоить его функционал и удобно вводить необходимые данные[2].

Для реализации приложения GUI идеально подходит, так как позволяет отображать информацию с помощью текста, изображений, списков и т.д., что позволяет пользователям легко просматривать и редактировать информацию о писателях и их произведениях.

Кроме того, графический интерфейс пользователя обладает широкой популярностью и распространённостью среди пользователей, что упрощает привлечение и обучение новых пользователей к работе с приложением.

Обозначим какие задачи должен решать графический интерфейс программы:

* Удобный и эффективный ввод данных о писателях и их произведениях.
* Простой и наглядный способ просмотра добавленных записей.
* Возможность добавления новых авторов, жанров, языков и издательств.
* Управление основными функциями приложения (добавление, сохранение, выход).
* Предоставление обратной связи пользователю.

Пользователь должен будет иметь возможность добавлять запись об авторе и его произведении, а также просматривать уже созданные им ранее записи.

Чтобы пользователь мог взаимодействовать с приложением нужно окно приложения включающее в себя различные элементы управления.Чтобы его создать нужен контейнер(панель компоновки) в котором будут содержаться эти самые элементы управления. Выберем контейнер типа GridPane. Переходя в это окно пользователь может добавить новую запись о писателе и его произведении или просмотреть созданные им записи раннее.Внутри этого контейнера чтобы разместить вложенные элементы управления у одной из сторон контейнера: по верхнему, нижнему, левому или правому краю или расположить по центру создадим контейнер BorderPane. Внутри этого контейнера чтобы элементы управления компоновались в виде вертикального ряда создадим контейнер VBox.

В контейнере VBox воспользуемся панелью меню MenuBar в которую поместим метку MenuItem при нажатии на которую будет открываться отдельное окно для вывода созданных записей. Для создания окна для вывода созданных записей воспользуемся контейнером Stage. В табличном виде пользователю более удобно воспринимать информацию о созданных записях, поэтому внутри контейнера Stage создадим таблицу с помощью TableView с столбцами «Автор», «Страна автора», «Название произведения», «Жанр», «Язык», «Издательство», «Тираж».

Изобразим раскадровку окна для вывода созданных записей о писателях и их произведениях(рис.2).

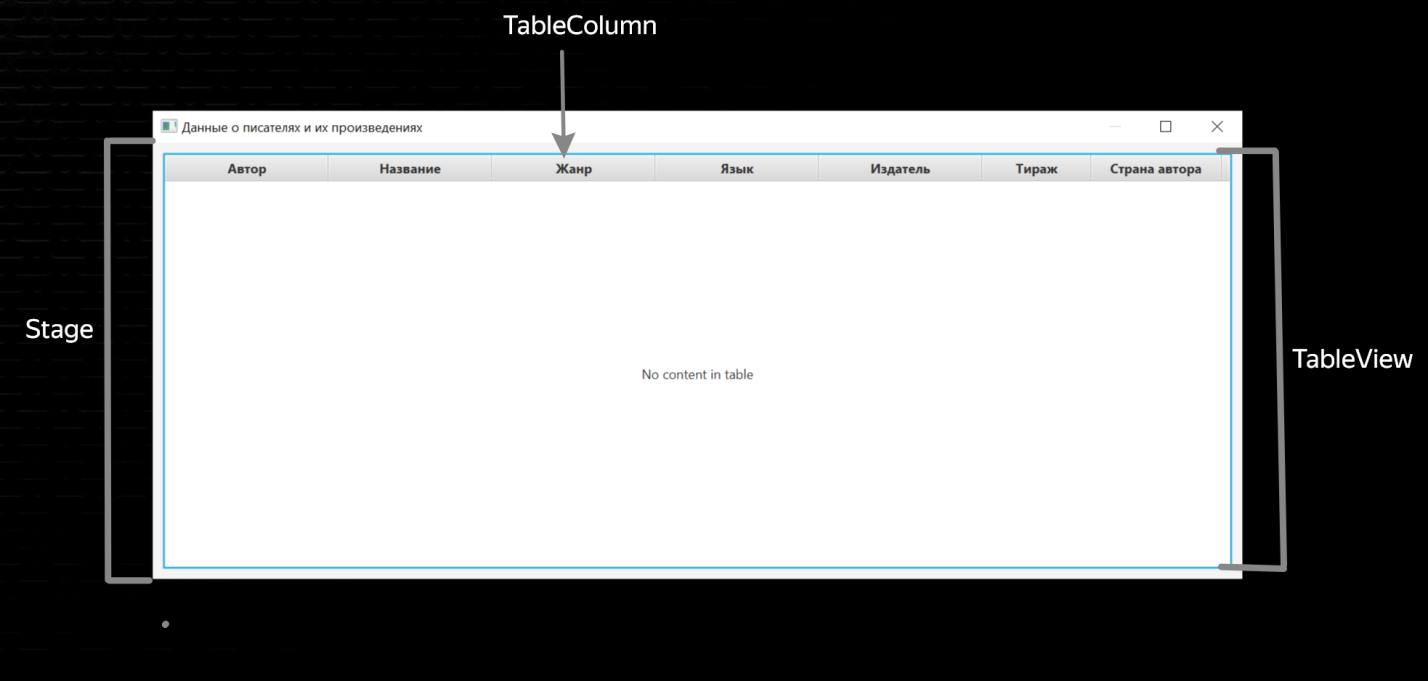


Рисунок 2 - Раскадровка GUI окна для вывода созданных записей о писателях и их произведениях.

Пользователь должен будет иметь возможность добавлять новые жанры, языки, авторов, издательства, издания, страну автора. Для этого создадим отдельный контейнер типа HBox в который поместим 3 кнопки Button при нажатии на которые будут выводится окна TextInputDialog, куда можно будет ввести текстовую информацию такую как имя автора, его страну, название жанра, язык.

Чтобы пользователь имел возможность выбирать из уже созданных записей авторов, жанры, страну автора, язык создадим элементы управления ComboBox позволяющие выбирать из выпадающего списка созданные элементы. У каждого произведения есть название издательство и тираж изданий этого произведения в этом издательстве. Чтобы пользователь мог вводить эти данные о произведении создадим три элемента управления типа TextField. Для того чтобы пользователь вносил данные в верные поля ввода, для каждого ComboBox и TextField создадим элемент управления для вывода текстовой информации Label.

Также для того чтобы пользователь при заполнении всех данных о произведении мог дать команду приложению записать эти данные, в отдельный контейнер типа HBox поместим кпопку Button “Добавить произведение”.

Изобразим раскадровку главного окна приложения(рис.3).

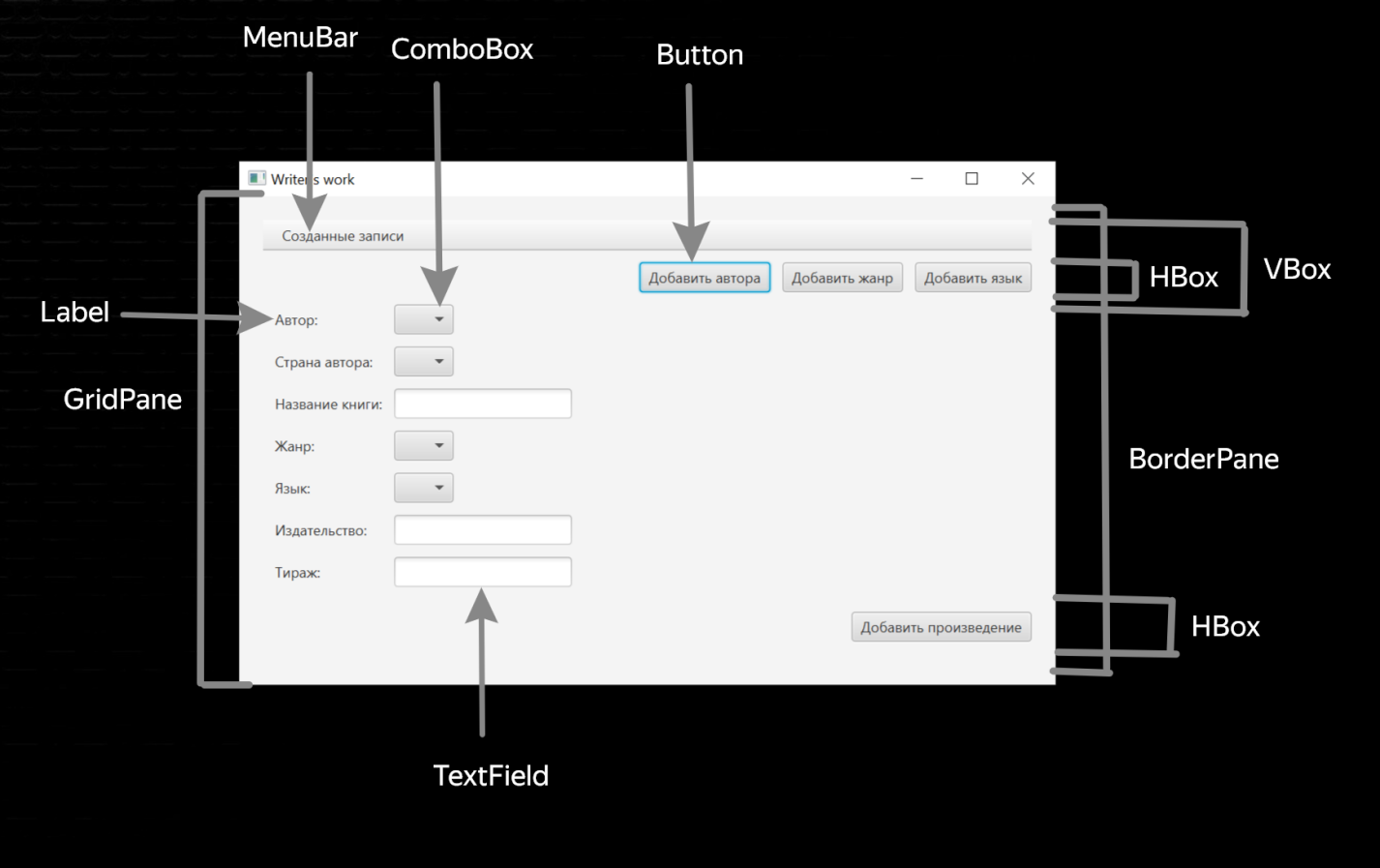


Рисунок 3 - Раскадровка GUI главного окна приложения.

2.2.Построение диаграммы деятельности приложения

Пользователь при открытии окна приложения может реализовывать два сценария использования приложения: создание новой записи о произведении и просмотр созданных записей. Для того чтобы изобразить как приложение будет действовать при реализации того или иного сценария воспользуемся диаграммой деятельности.

Диаграмма деятельности - это блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Результат может привести к изменению состояния системы или возвращению некоторого значения.

Диаграммы деятельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений[3].

Изобразим примерную диаграмму деятельности которая покажет два сценария использования создаваемого приложения приложения: создание новой записи о произведении и просмотр созданных записей(Рис.4).

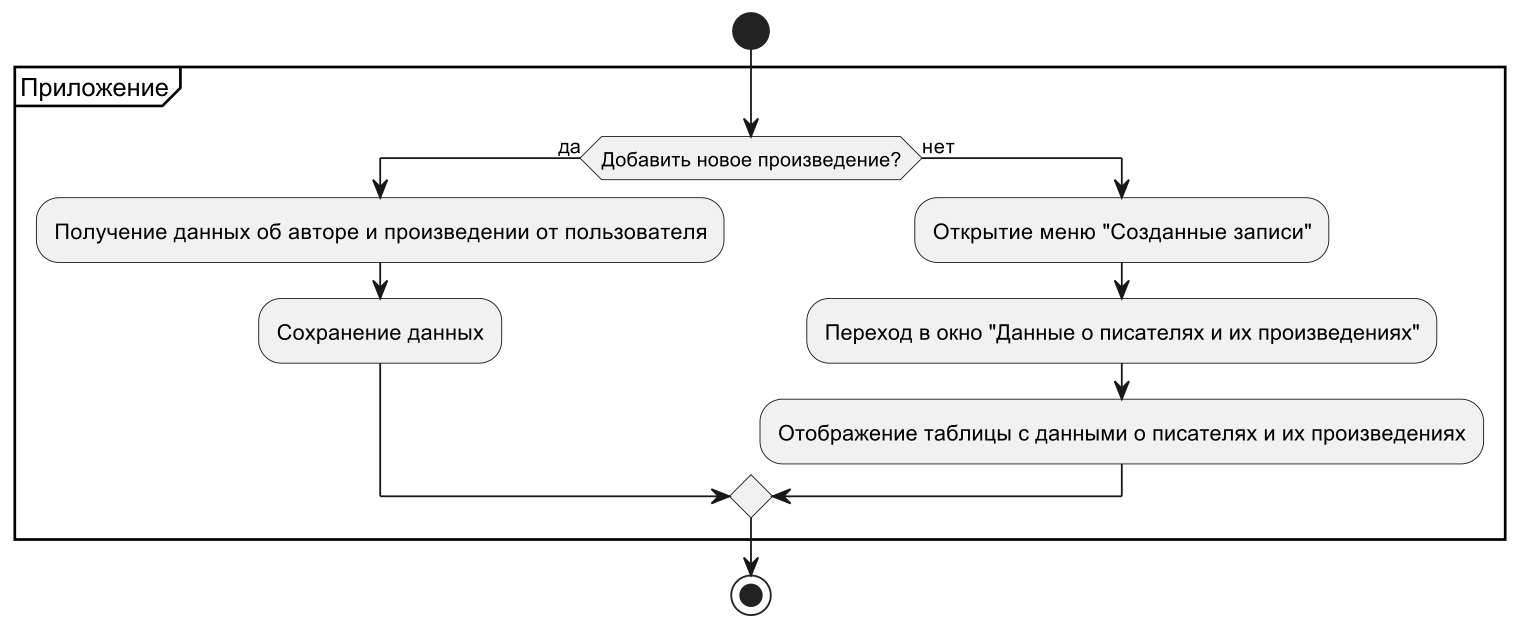


Рисунок 4 - Диаграмма деятельности приложения.

Опишем изображённую диаграмму деятельности:

* Начало: Процесс начинается с открытия пользователем приложения.
* Выбор действия: Пользователь делает выбор, какое действие он хочет совершить.
* Добавление произведения: если пользователь хочет добавить новое произведение, то выполняется блок «Получение данных об авторе и произведении от пользователя»  с последующим действием «Сохранение данных».
* Просмотр записей: если пользователь хочет просмотреть существующие записи, то выполняется блок  «Открытие меню "Созданные записи"», с последующим действием «Переход в окно "Данные о писателях и их произведениях"» и завершающим действием «Отображение таблицы с данными о писателях и их произведениях».
* Конец: Процесс завершается при закрытии приложения.

2.3. Построение диаграмм последовательности приложения

Диаграмма последовательности ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) sequence diagram) — [UML-диаграмма](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_(UML)" \o "Диаграмма (UML)), на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие акторов (действующих лиц) информационной системы в рамках [прецедента](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82_(UML)" \o "Прецедент (UML))[4].

Для чего нужна диаграмма последовательности:

Визуализация процесса: диаграмма последовательностей наглядно показывает, как различные компоненты приложения (графический интерфейс, контроллер, хранилище данных) взаимодействуют друг с другом во время добавления книги. Это упрощает понимание потока данных и логики работы приложения.

Понимание взаимодействий: она позволяет увидеть, какие методы и объекты вызываются в какой последовательности при выполнении определённого действия. Это помогает понять, как происходит обмен информацией между разными частями приложения.

Выявление проблем: диаграмма может помочь выявить потенциальные проблемы или «узкие места» в логике приложения. Например, если какой-то объект используется слишком часто, это может указывать на необходимость пересмотра архитектуры.

Упрощение разработки: понимание потока данных и взаимодействий делает процесс разработки и отладки более простым и эффективным. Зная, как вызываются методы, можно точнее определить, где находится проблема при отладке.

Определение ответственности: диаграмма наглядно показывает, какой объект (или компонент) отвечает за выполнение определённых действий.

Для того чтобы визуализировать взаимодействие между объектами приложения и пользователем составим диаграммы последовательностей.

Пользователь может проигрывать два сценария при обращении к приложению (Добавление новой записи и просмотр созданных записей).

Изобразим диаграмму последовательности для первого сценария использования приложения(рис.5).

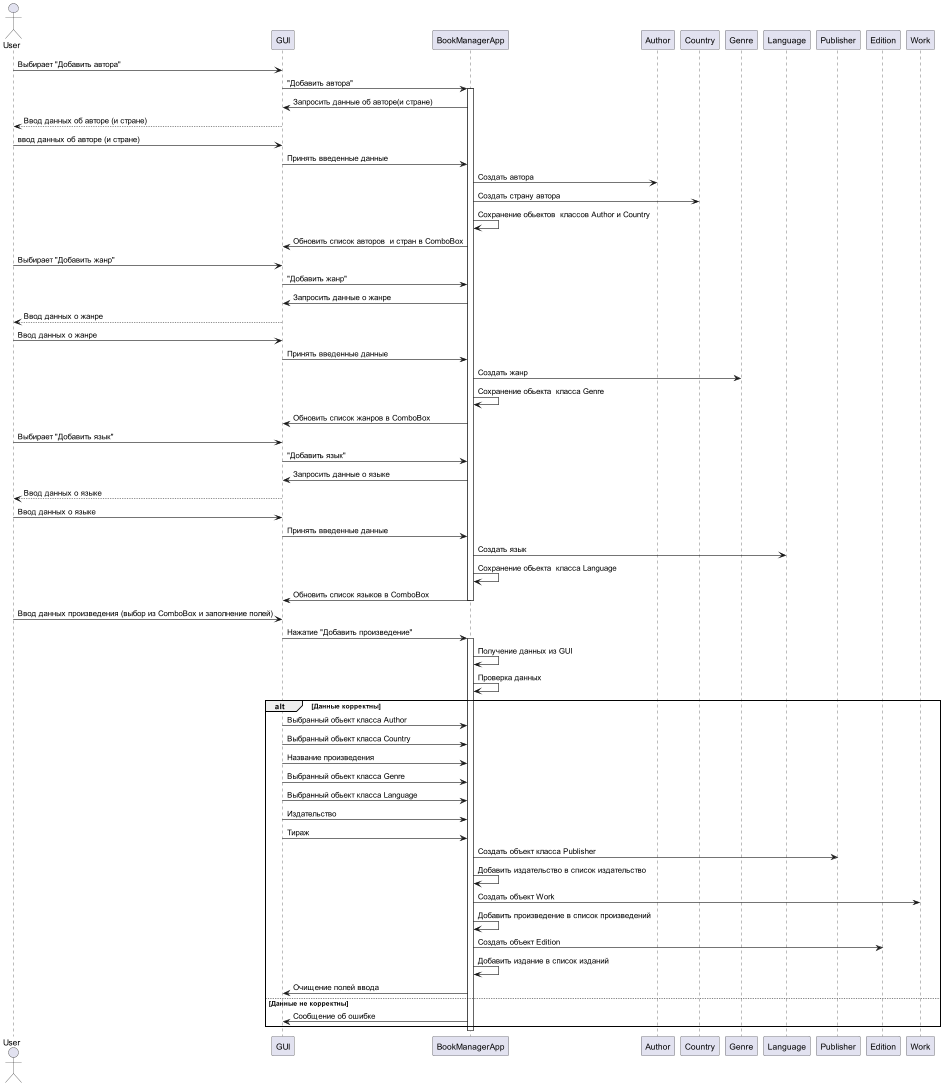


Рисунок 5 - Диаграмма последовательностей

Изобразим диаграмму последовательности для второго сценария, когда пользователь открывает приложение и хочет посмотреть созданные записи(рис.6).

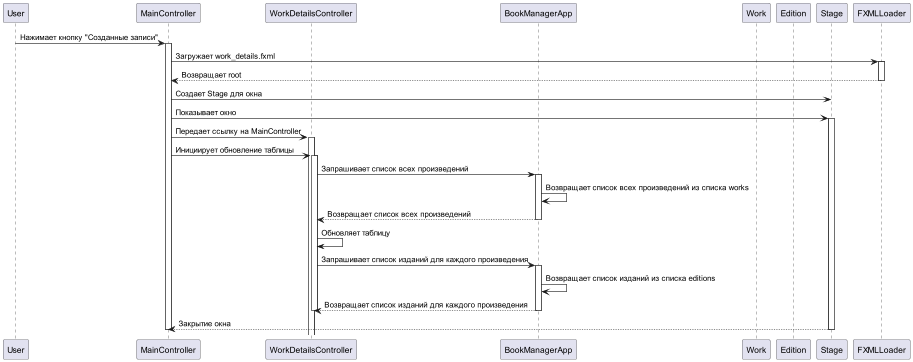


Рисунок 6 - Диаграмма последовательности для второго сценария.

2.4.Диаграмма классов всего приложения

Диаграмма классов (class diagram) всего приложения — это фундаментальный инструмент в разработке программного обеспечения, который выполняет ряд важных функций[5]:

1. Визуализация структуры приложения:

Диаграмма классов даёт наглядное представление о структуре приложения, показывая все основные классы, их атрибуты (поля) и методы. Она показывает, как классы связаны между собой, включая наследование, ассоциации, агрегацию и композицию. Это помогает понять, как различные части приложения взаимодействуют друг с другом. Диаграмма помогает понять масштаб приложения, особенно когда оно становится сложным и разветвлённым.

2. Проектирование и планирование:

Диаграмма классов помогает принимать архитектурные решения, показывая сильные и слабые стороны текущей структуры. Она помогает планировать разработку, распределять задачи между членами команды и устанавливать зависимости между различными частями проекта, также позволяет проверить правильность структуры до начала кодирования и помогает избежать ошибок, которые могут возникнуть в процессе разработки.

3. Поиск и исправление ошибок:

Диаграмма помогает анализировать код в случае возникновения ошибок, так как на ней можно увидеть связи между классами и потоки управления. Упрощает рефакторинг кода, предоставляя чёткое представление о том, какие классы нужно изменить и как это повлияет на другие части приложения.

Изобразим диаграмму классов реализуемого приложения(рис. 7).

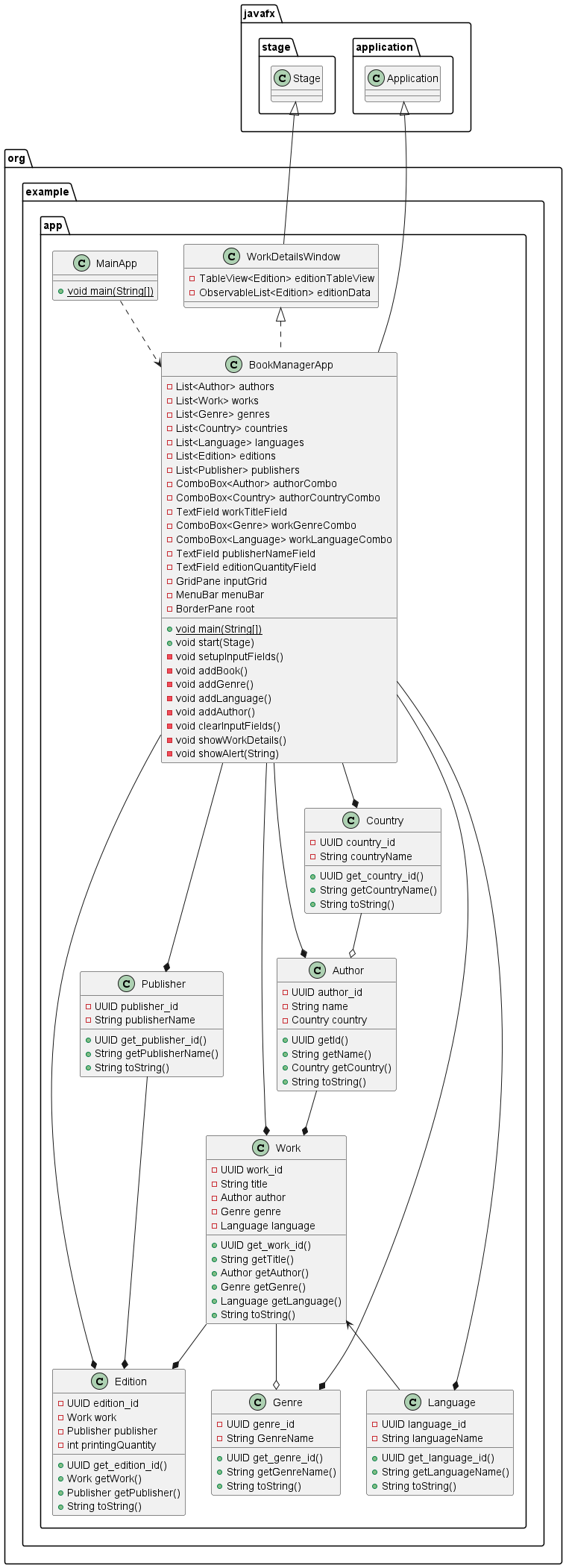
.

Рисунок 7- Диаграмма классов всего приложения.

3.Реализация приложения

3.1.Структура программного проекта

Рассмотрим структуру реализуемого программного проекта(рис.8):

* App-W :Корневая директория проекта.
* .gigaide : Конфигурационные файлы для IDE Giga IDE.
* .idea: Конфигурационные файлы для IDE IntelliJ IDEA.
* .mvn: Конфигурационные файлы для инструмента сборки Maven.
* src: Директория с исходным кодом проекта.
* target: Каталог для сборки (содержит скомпилированные классы и другие артефакты).
* .gitignore: Файл, указывающий, какие файлы и директории Git должен игнорировать.
* mvnw, mvnw.cmd: Скрипты для запуска Maven Wrapper (для сборки проекта без установки Maven на компьютер).
* pom.xml: Файл конфигурации проекта Maven (содержит зависимости проекта, настройки сборки и т. д.).

Рассмотрим директорию src с исходным кодом проекта:

* main: Содержит основной код приложения.
* java: Директория для Java кода.
* org.example.app: Это базовый пакет (package) для всего приложения. Он содержит все классы, которые относятся к программному проекту.
* Author.java, Country.java, Edition.java, Genre.java, Language.java, Publisher.java, Work.java: Классы модели, представляющие сущности предметной области.
* BookManagerApp.java: Основной класс JavaFX приложения (GUI).
* WorkDetailsWindow.java: Класс JavaFX окна с таблицей содержащей созданные записи о писателях и их произведениях.
* MainApp.java: Класс для запуска приложения
* module-info.java: Файл для определения модулей.
* resources: Каталог для ресурсов (файлов конфигурации, изображений и т. д.)

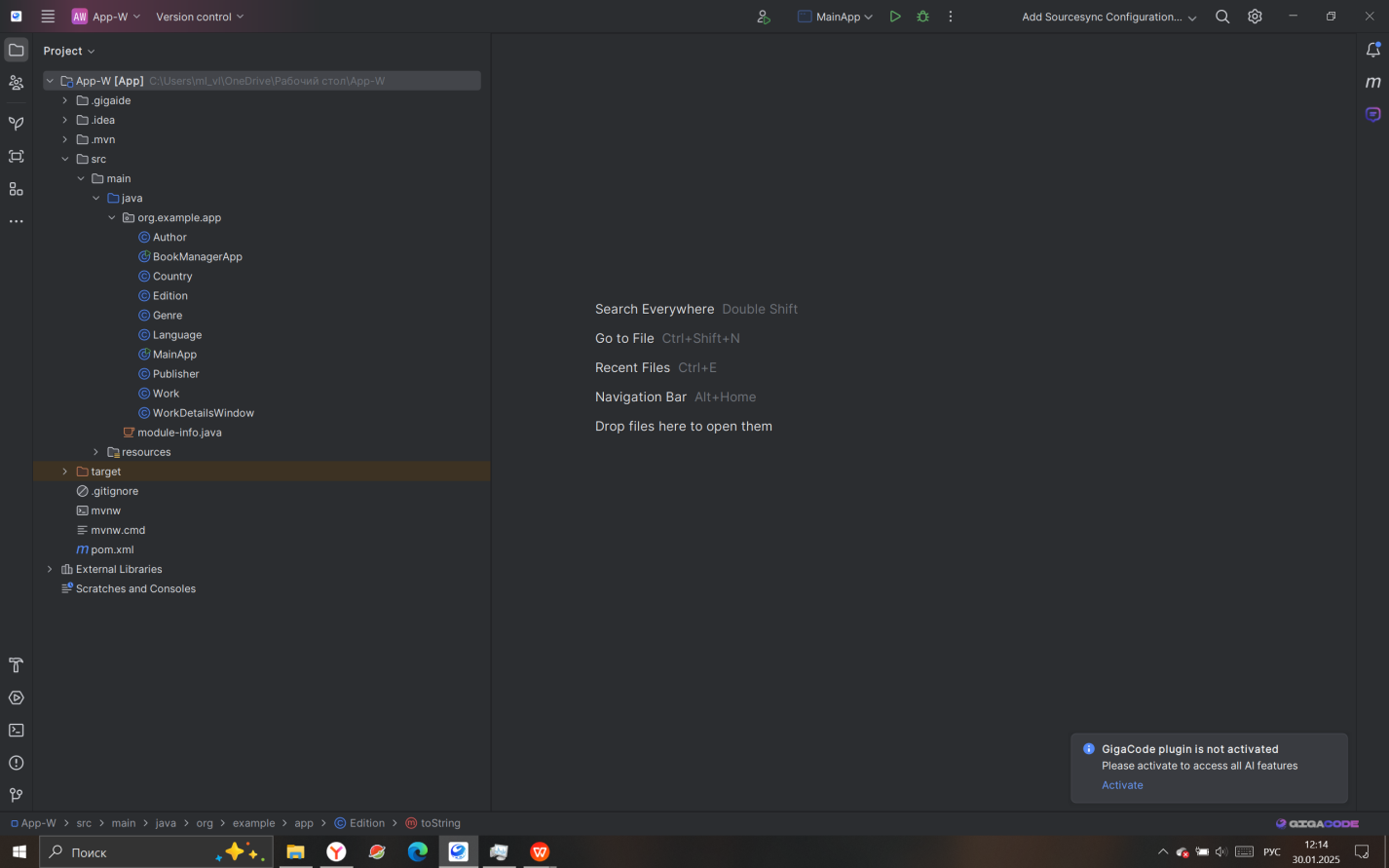


Рисунок 8 - Структура проекта.

* 1. Описание классов приложения

Опишем поля и методы реализуемых классов приложения.

Класс Author

public class Author

Этот класс используется для создания автора произведения.

Атрибуты класса Author:

-author\_id

private UUID author\_id

Уникальный идентификатор объекта, генерируемый как UUID при его создании.

-name

private String name

Имя автора записываемое строкой.

-Country

private Country country

Ссылка на объект класса Country, представляющий страну автора.

Конструктор класса Author :

public Author(String name, Country country)

Создает объект класса Author, генерируя UUID для author\_id и присваивая атрибутам принимаемые значения.

Параметры:

String name - значение типа String определяющее имя автора

Country country - значение типа Country определяющее ссылку на нужный объект класса Country.

Методы класса Author :

-getId

public UUID getId()

Возвращает уникальный UUID идентификатор объекта класса Author.

Возврат(return): author\_id

-getName

Метод возвращает значение типа String с именем автора текущего экземпляра класса Author

Возврат(return): name

-getCountry

public Country getCountry()

Возвращает объект класса Country с информацией о стране автора.

Возврат(return): country

-toString

@Override  
public String toString()

Переопределен для возвращения имени автора в виде строки.

Возврат(return):name

Класс Country

public class Country

Класс используется для создания страны автора.

Атрибуты класса Country:

-country\_id

private UUID country\_id

Уникальный идентификатор объекта класса Country, генерируемый как UUID его создании.

-countryName

private String countryName

Название страны записываемое строкой.

Конструктор класса Country

public Country(String name)

Создает экземпляр класса, генерируя UUID для country\_id и присваивая атрибуту countryName принимаемое значение.

Параметр: String name - значение типа String определяющее название страны автора.

Метода класса Country:

-get\_country\_id

public UUID get\_country\_id()

Возвращает уникальный UUID идентификатор экземпляра класса Country.

Возврат (return): country\_id

-getCountryName

public String getCountryName()

Метод возвращает значение типа String с названием страны текущего экземпляра класса Country.

Возврат (return):сountryName

-toString

@Override  
public String toString()

Переопределен для возвращения названия страны в виде строки.

Возврат (return):сountryName

Класс Genre

public class Genre

Класс используется для создания жанра произведения.

Атрибуты класса Genre:

-genre\_id

private UUID genre\_id

Уникальный идентификатор экземпляра класса Genre, генерируемый как UUID его создании.

-genreName

private String GenreName

Название жанра записываемое строкой.

Конструктор класса Genre

public Genre(String name)

Создает экземпляр класса, генерируя UUID для genre\_id и присваивая атрибуту genreName принимаемое значение.

Параметр: String name - значение типа String определяющее название жанра произведения.

Методы класса Genre

-get\_genre\_id

public UUID get\_genre\_id()

Возвращает уникальный UUID идентификатор экземпляра класса Genre.

Возврат (return): genre\_id

-getGenreName

Метод возвращает значение типа String с названием жанра текущего экземпляра класса Genre.

Возврат (return): genreName

-toString

@Override  
public String toString()

Переопределен для возвращения названия жанра в виде строки.

Возврат (return):genreName

Класс Language

public class Language

Класс используется для создания языка произведения.

Атрибуты класса Language

-language\_id

private UUID language\_id

Уникальный идентификатор экземпляра класса Language, генерируемый как UUID его создании.

-languageName

private String languageName

Название языка записываемое строкой.

Конструктор класса Language

public Language(String name)

Создает экземпляр класса, генерируя UUID для language\_id и присваивая атрибуту languageName принимаемое значение.

Параметр: String name - значение типа String определяющее название языка произведения.

Методы класса Language

-get\_language\_id

public UUID get\_language\_id()

Возвращает уникальный UUID идентификатор экземпляра класса Language

Возврат (return):language\_id

-getLanguageName

Метод возвращает значение типа String с названием языка текущего экземпляра класса Language.

Возврат (return): languageName

-toString

@Override  
public String toString()

Переопределен для возвращения названия языка в виде строки.

Возврат (return): languageName

Класс Publisher

public class Publisher

Класс используется для создания издательства произведения.

Атрибуты класса Publisher:

-publisher\_id

private UUID publisher\_id

Уникальный идентификатор экземпляра класса Publisher, генерируемый как UUID его создании.

-publisherName

private String publisherName

Название издательства записываемое строкой.

Конструктор класса Publisher

public Publisher(String name)

Создает экземпляр класса, генерируя UUID для publisher\_id и присваивая атрибуту publisherName принимаемое значение.

Параметр: String name - значение типа String определяющее название издательства.

Методы класса Publisher:

-get\_ publisher\_id

public UUID get\_publisher\_id()

Возвращает уникальный UUID идентификатор экземпляра класса Publisher.

Возврат (return): publisher\_id

-getPublisherName

public String getPublisherName()

Метод возвращает значение типа String с названием издательства текущего экземпляра класса Publisher.

Возврат (return): publisherName

-toString

@Override  
public String toString()

Переопределен для возвращения названия издательства в виде строки.

Возврат (return): publisherName

Класс Work

public class Work

Класс используется для создания произведения.

Атрибуты класса Work

-work\_id:

private UUID work\_id

Уникальный идентификатор экземпляра класса Work, генерируемый как UUID его создании.

-title

private String title

Название издательства записываемое строкой.

-author

private Author author

Ссылка на объект класса Author, представляющий автора произведения.

-genre

private Genre genre

Ссылка на объект класса Genre, представляющий жанр произведения.

-language

private Language language

Ссылка на объект класса Language, представляющий язык.

Конструктор класса Work

public Work(String title, Author author, Genre genre, Language language)

Создает экземпляр класса Work, генерируя UUID для work\_id и присваивая атрибутам принимаемые значения.

Параметры:

String title- - значение типа String, определяющее название произведения.

Author author- значение типа Author, определяющее ссылку на обьект представляющий автора произведения.

Genre genre - значение типа Genre, определяющее ссылку на обьект представляющий жанр произведения.

Language language - значение типа Language, определяющее ссылку на обьект представляющий язык произведения.

Методы класса Work:

-get\_ work\_id

public UUID get\_work\_id()

Возвращает уникальный UUID идентификатор экземпляра класса Work.

Возврат (return): publisherName.

-getTitle

public String getTitle()

Метод возвращает строку с названием страны текущего экземпляра класса Work.

Возврат (return): title.

-getAuthor

public Author getAuthor()

Возвращает автора произведения (объект Author)

Возврат (return): author.

-getGenre

public Genre getGenre()

Возвращает жанр произведения (объект Genre).

Возврат (return):genre.

-getLanguage

public Language getLanguage()

Возвращает язык произведения (объект Language).

Возврат (return):language.

toString

public String toString()

Переопределён для возврата строки, содержащей название произведения, имя автора, название жанра и название языка.

Возврат (return): title + " by " + author.getName() + ", Genre: " + genre.getGenreName() + ", Language: " + language.getLanguageName().

Класс Edition

public class Edition

Класс используется для создания издания произведения.

Атрибуты класса Edition:

-edition\_id

private UUID edition\_id

Уникальный идентификатор экземпляра класса Edition, генерируемый как UUID его создании.

-work

private Work work

Объект класса Work, представляющий произведение которое издаётся этим изданием.

-publisher

private Publisher publisher

Объект класса Publisher, представляющий издательство издания.

printingQuantity

private int printingQuantity

Целое число, представляющее тираж издания.

Конструктор класса Edition

public Edition(Work work, Publisher publisher, int printingQuantity)

Принимает значения Work work, Publisher publisher, int printingQuantity

Создает экземпляр класса Edition, генерируя UUID для edition\_id и присваивая атрибутам принимаемые значения.

Параметры:

Work work - значение типа Work, определяющее ссылку на обьект представляющий произведение.

Publisher publisher - значение типа Publisher, определяющее ссылку на обьект представляющий издательство.

int printingQuantity - значение типа Int, определяющее тираж обьекта Edition.

Методы класса Edition

- get\_edition\_id

public UUID get\_edition\_id()

Возвращает уникальный UUID идентификатор экземпляра класса Edition.

Возврат (return): edition\_id.

-getWork

public Work getWork()

Возвращает произведение, к которому относится издание (объект Work).

Возврат (return):work.

-getPublisher

public Publisher getPublisher()

Возвращает издателя издания (объект Publisher).

Возврат (return):publisher.

-toString

@Override  
public String toString()

Переопределён для возврата строки, содержащей название произведения, имя издателя и тираж.

Возврат (return):"Edition info: " + work.getTitle() + ", Publisher: "+ publisher.getPublisherName()+ ", Printing Quantity "+ printingQuantity.

Класс WorkDetailsWindow

public class WorkDetailsWindow extends Stage

Создает окно для вывода таблицы с созданными записями о писателях и их произвелениях.

Атрибуты класса WorkDetailsWindow:

-editionTableView

private TableView<Edition> editionTableView

Атрибут типа TableView используется для отображения данных в табличном виде.

-editionData

private ObservableList<Edition> editionData

Атрибут типа данных ObservableList, используется для хранения данных для отображения в таблице editionTableView.

Конструктор класса WorkDetailsWindow

public WorkDetailsWindow(ObservableList<Edition> editions)

Cоздает обьект класса WorkDetailsWindow.

Инициализирует модальное окно, устанавливает заголовок окна, создает таблицу TableView и добавляет в неё данные из принятого списка editions.

Создает столбцы таблицы, устанавливая для каждого столбца соответствующие ячейки значений (setCellValueFactory) для отображения данных.Для столбцов с текстовыми значениями используется SimpleStringProperty. Для столбца printingQuantity используется PropertyValueFactory. Добавляет все столбцы в таблицу. Создает корневой макет VBox, добавляет в него таблицу и устанавливает отступы. Устанавливает scene для окна.

Параметры:ObservableList<Edition> editions

Класс BookManagerApp

public class BookManagerApp extends Application

Главный класс приложения, который управляет логикой и пользовательским интерфейсом.

Реализует метод start для инициализации приложения JavaFX, а также другие методы для обработки действий пользователя (добавление произведений, стран, авторов, жанров, языков и просмотр созданных записей).

Атрибуты класса BookManagerApp

-authors

private List<Author> authors

Коллекция для хранения объектов класса Authors.

-works

private List<Work> works

Коллекция для хранения объектов класса Work.

- genres

private List<Genre> genres

Коллекция для хранения объектов класса Genre.

-countries

private List<Country> countries

Коллекция для хранения объектов класса Country.

-languages

private List<Language> languages

Коллекция для хранения объектов класса Language.

-editions

private List<Edition> editions

Коллекция для хранения объектов класса Edition.

-publishers

private List<Publisher> publishers

Коллекция для хранения объектов класса Publisher.

-authorCombo

private ComboBox<Author> authorCombo

Элемент управления JavaFX для выбора автора произведения.

-authorCountryCombo

private ComboBox<Country> authorCountryCombo;

Элемент управления JavaFX для выбора страны автора произведения.

-workTitleField

private TextField workTitleField

Элемент управления JavaFX для ввода названия произведения.

-workGenreCombo

private ComboBox<Genre> workGenreCombo

Элемент управления JavaFX для выбора жанра произведения.

-workLanguageCombo

private ComboBox<Language> workLanguageCombo;

Элемент управления JavaFX для выбора языка произведения.

-publisherNameField

private TextField publisherNameField

Элемент управления JavaFX для ввода названия издательства произведения.

-editionQuantityField

private TextField editionQuantityField;  
Элемент управления JavaFX для ввода тиража издания произведения.

-inputGrid

private GridPane inputGrid;

Контейнер типа GridPane для создания окна приложения.

-menuBar

private MenuBar menuBar;

Графический элемент пользовательского интерфейса «панель меню» .

-root

private BorderPane root

Контейнер типа BorderPane для размещения элементов управления.

Методы класса BookManagerApp

-main

public static void main(String[] args)

Запускает JavaFX приложение с помощью метода launch(args)

Параметры:String[] args

-start

@Override  
public void start(Stage primaryStage)

Инициализирует основное окно приложения, задает заголовок окна, настраивает элементы пользовательского интерфейса (меню, поля ввода, кнопки), создает контейнер BorderPane, устанавливает Scene и отображает Stage.

Параметры:Stage primaryStage

-setupInputFields

private void setupInputFields()

Инициализирует элементы ввода (ComboBox и TextField)

-addBook

private void addBook()

Получает данные из полей ввода, создаёт объекты классов Work, Publisher и Edition, добавляет их в соответствующие списки и очищает поля ввода.

-addGenre

private void addGenre()

Открывает диалоговое окно для ввода названия жанра, создаёт объект Genre, добавляет его в список жанров и обновляет ComboBox для выбора жанра.

-addLanguage

private void addLanguage()

Открывает диалоговое окно для ввода названия языка, создаёт объект Language, добавляет его в список языков и обновляет ComboBox для выбора языка.

-addAuthor

private void addAuthor()

Открывает два диалоговых окна для ввода имени автора и страны автора, создаёт объекты Author и Country, добавляет их в соответствующие списки и обновляет ComboBox для выбора автора и страны.

-clearInputFields

private void clearInputFields()

Очищает все поля ввода в окне.

-showWorkDetails

private void showWorkDetails()

Создает и отображает окно с таблицей содержащей записи о писателях и их произведениях (WorkDetailsWindow), передавая ему список всех изданий.

-showAlert

private void showAlert(String message)

Отображает окно с сообщением об ошибке.

Класс MainApp

public class MainApp

Основной класс, который содержит метод main для запуска приложения BookManagerApp. Класс MainApp служит точкой входа в приложение. Он выступает в роли лаунчера, вызывая основной метод приложения BookManagerApp, который уже инициализирует и отображает графический интерфейс пользователя. Он нужен для того, чтобы отделить запуск приложения от логики построения пользовательского интерфейса.

Метод main

public static void main(String[] args)

Вызывает метод main класса BookManagerApp. Это позволяет запускать приложение, содержащее графический интерфейс, из класса, у которого нет графического интерфейса.

Параметры:String[] args

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта практически отработали навыки самостоятельной разработки приложения на основе методов, изученных рамках курса "Объектно-ориентированное программирование", являющихся основой для структурирования, проектирования и реализации программного обеспечения с применением принципов ООП для выбранной предметной области.

Также были решены следующие задачи:

1. Произведено моделирование классов предметной области.
2. Произвели проектирование приложение.
3. Реализовали приложение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Соловьев, С. В. Технология разработки прикладного программного обеспечения / С. В. Соловьев, Р. И. Цой, Л. С. Гринкруг. – Москва : Акад. естествознания, 2011. – 407 с.
2. Графические интерфейсы пользователя Java/ Мишнин Т. - 686 с.
3. Фаулер M. UML. Основы, 3 е издание. – Пер. с англ. – СПб: Символ Плюс, 2004 – 192 с.
4. Моисеев А.Н., Литовченко М.И. Основы языка UML : учеб. пособие. – Томск : ИздательствоТомского государственного университета, 2023 – 96 с.
5. Использование диаграммы классов UML при проектировании и документировании программного обеспечения/Хабр https://habr.com/ru/articles/572234/

ПРИЛОЖЕНИЕ

Author.java

package org.example.app;  
import java.util.UUID;  
public class Author {  
 private UUID author\_id;  
 private String name;  
 private Country country;  
 public Author(String name, Country country) {  
 this.author\_id = UUID.randomUUID();  
 this.name = name;  
 this.country = country;  
 }  
 public UUID getId() {return author\_id;}  
 public String getName() {return name;}  
 public Country getCountry() {return country;}  
 @Override  
 public String toString() {return name;}  
}

Country.java

package org.example.app;  
import java.util.UUID;  
public class Country {  
 private UUID country\_id;  
 private String countryName;  
 public Country(String name) {  
 this.country\_id = UUID.randomUUID();  
 this.countryName = name;  
 }  
 public UUID get\_country\_id() {  
 return country\_id;  
 }  
 public String getCountryName() {return countryName;}  
 @Override  
 public String toString(){return countryName;}  
}

Genre.java

package org.example.app;  
import java.util.UUID;  
public class Genre {  
 private UUID genre\_id;  
 private String GenreName;  
 public Genre(String name) {  
 this.genre\_id = UUID.randomUUID();  
 this.GenreName = name;  
 }  
 public UUID get\_genre\_id() {  
 return genre\_id;  
 }  
 public String getGenreName() {  
 return GenreName;  
 }  
 @Override  
 public String toString(){return GenreName;}  
}

Edition.java

package org.example.app;  
import java.util.UUID;  
public class Edition {  
 private UUID edition\_id;  
 private Work work;  
 private Publisher publisher;  
 private int printingQuantity;  
 public Edition(Work work, Publisher publisher, int printingQuantity) {  
 this.edition\_id = UUID.randomUUID();  
 this.work = work;  
 this.publisher = publisher;  
 this.printingQuantity = printingQuantity;  
 }  
 public UUID get\_edition\_id() {  
 return edition\_id;  
 }  
 public Work getWork() {  
 return work;  
 }  
 public Publisher getPublisher() {  
 return publisher;  
 }  
 @Override  
 public String toString()  
 {return "Edition info: " + work.getTitle() + ", Publisher: "+ publisher.getPublisherName()+ ", Printing Quantity "+ printingQuantity;  
 }  
}

Language.java

package org.example.app;  
import java.util.UUID;  
public class Language {  
 private UUID language\_id;  
 private String languageName;  
 public Language(String name) {  
 this.language\_id = UUID.randomUUID();  
 this.languageName = name;  
 }  
 public UUID get\_language\_id() {  
 return language\_id;  
 }  
 public String getLanguageName() {  
 return languageName;  
 }  
 @Override  
 public String toString(){return languageName;}  
}

Publisher.java

package org.example.app;  
import java.util.UUID;  
public class Publisher {  
 private UUID publisher\_id;  
 private String publisherName;  
 public Publisher(String name) {  
 this.publisher\_id = UUID.randomUUID();  
 this.publisherName = name;  
 }  
  
 public UUID get\_publisher\_id() {  
 return publisher\_id;  
 }  
 public String getPublisherName() {  
 return publisherName;  
 }  
 @Override  
 public String toString()  
 {  
 return publisherName;  
 }  
}

Work.java

package org.example.app;  
import java.util.UUID;  
public class Work {  
 private UUID work\_id;  
 private String title;  
 private Author author;  
 private Genre genre;  
 private Language language;  
 public Work(String title, Author author, Genre genre, Language language) {  
 this.work\_id = UUID.randomUUID();  
 this.title = title;  
 this.author = author;  
 this.genre = genre;  
 this.language = language;  
 }  
 public UUID get\_work\_id(){return work\_id;}  
 public String getTitle() {  
 return title;  
 }  
 public Author getAuthor() {  
 return author;  
 }  
 public Genre getGenre() {  
 return genre;  
 }  
 public Language getLanguage() {  
 return language;  
 }  
 @Override  
 public String toString() {  
 return title + " by " + author.getName() + ", Genre: " + genre.getGenreName() + ", Language: " + language.getLanguageName();  
 }  
}

BookManagerApp.java

package org.example.app;  
import javafx.application.Application;  
import javafx.collections.FXCollections;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.layout.BorderPane;  
import javafx.scene.layout.GridPane;  
import javafx.scene.layout.HBox;  
import javafx.scene.layout.VBox;  
import javafx.stage.Stage;  
import java.util.ArrayList;  
import java.util.List;  
public class BookManagerApp extends Application {  
 private List<Author> authors = new ArrayList<>();  
 private List<Work> works = new ArrayList<>();  
 private List<Genre> genres = new ArrayList<>();  
 private List<Country> countries = new ArrayList<>();  
 private List<Language> languages = new ArrayList<>();  
 private List<Edition> editions = new ArrayList<>();  
 private List<Publisher> publishers = new ArrayList<>();  
 // Теперь authorNameField ComboBox  
 private ComboBox<Author> authorCombo;  
 private ComboBox<Country> authorCountryCombo;  
 private TextField workTitleField;  
 private ComboBox<Genre> workGenreCombo;  
 private ComboBox<Language> workLanguageCombo;  
 private TextField publisherNameField;  
 private TextField editionQuantityField;  
 // Объявляем inputGrid, menuBar, root как поля класса  
 private GridPane inputGrid;  
 private MenuBar menuBar;  
 private BorderPane root;  
 public static void main(String[] args) {  
 launch(args);  
 }  
 @Override  
 public void start(Stage primaryStage) {  
 primaryStage.setTitle("Writer`s work");  
 // Инициализация элементов ввода  
 setupInputFields();  
 Button addButton = new Button("Добавить произведение");  
 addButton.setOnAction(e -> addBook());  
 Button addGenreButton = new Button("Добавить жанр");  
 addGenreButton.setOnAction(e -> addGenre());  
 Button addLanguageButton = new Button("Добавить язык");  
 addLanguageButton.setOnAction(e -> addLanguage());  
 Button addAuthorButton = new Button("Добавить автора");  
 addAuthorButton.setOnAction(e -> addAuthor());  
 inputGrid = new GridPane();  
 inputGrid.setPadding(new Insets(10));  
 inputGrid.setHgap(10);  
 inputGrid.setVgap(10);  
 inputGrid.add(new Label("Автор:"), 0, 0);  
 inputGrid.add(authorCombo, 1, 0);  
 inputGrid.add(new Label("Страна автора:"), 0, 1);  
 inputGrid.add(authorCountryCombo, 1, 1);  
 inputGrid.add(new Label("Название книги:"), 0, 2);  
 inputGrid.add(workTitleField, 1, 2);  
 inputGrid.add(new Label("Жанр:"), 0, 3);  
 inputGrid.add(workGenreCombo, 1, 3);  
 inputGrid.add(new Label("Язык:"), 0, 4);  
 inputGrid.add(workLanguageCombo, 1, 4);  
 inputGrid.add(new Label("Издательство:"), 0, 5);  
 inputGrid.add(publisherNameField, 1, 5);  
 inputGrid.add(new Label("Тираж:"), 0, 6);  
 inputGrid.add(editionQuantityField, 1, 6);  
 // Создание меню  
 menuBar = new MenuBar();  
 Menu fileMenu = new Menu("Созданные записи");  
 MenuItem showDetailsItem = new MenuItem("Данные о писателях и их книгах");  
 showDetailsItem.setOnAction(e -> showWorkDetails());  
 fileMenu.getItems().add(showDetailsItem);  
 menuBar.getMenus().add(fileMenu);  
 // Создание HBox для кнопок в верхнем правом углу  
 HBox topRightButtons = new HBox(10);  
 topRightButtons.setAlignment(Pos.TOP\_RIGHT);  
 topRightButtons.getChildren().addAll(addAuthorButton, addGenreButton, addLanguageButton);  
 //Создаем VBox для меню и кнопок  
 VBox menuButtonsBox = new VBox(10);  
 menuButtonsBox.getChildren().addAll(menuBar, topRightButtons);  
 //Создание HBox для размещения кнопки добавить книгу  
 HBox buttonBox = new HBox(10);  
 buttonBox.setAlignment(Pos.BOTTOM\_RIGHT);  
 buttonBox.getChildren().add(addButton);  
 VBox content = new VBox(10);  
 content.getChildren().addAll(inputGrid, buttonBox);  
 // Создаем корневой BorderPane и добавляем в него элементы  
 root = new BorderPane();  
 root.setTop(menuButtonsBox);  
 root.setCenter(content);  
 root.setPadding(new Insets(20));  
 Scene scene = new Scene(root, 900, 600);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
 private void setupInputFields() {  
 authorCombo = new ComboBox<>(FXCollections.observableArrayList(authors));  
 authorCountryCombo = new ComboBox<>(FXCollections.observableArrayList(countries));  
 workTitleField = new TextField();  
 workGenreCombo = new ComboBox<>(FXCollections.observableArrayList(genres));  
 workLanguageCombo = new ComboBox<>(FXCollections.observableArrayList(languages));  
 publisherNameField = new TextField();  
 editionQuantityField = new TextField();  
 }  
 private void addBook() {  
 Author author = authorCombo.getValue();  
 Country country = authorCountryCombo.getValue();  
 String workTitle = workTitleField.getText();  
 Genre genre = workGenreCombo.getValue();  
 Language language = workLanguageCombo.getValue();  
 String publisherName = publisherNameField.getText();  
 String editionQuantityText = editionQuantityField.getText();  
 if (author == null) {  
 showAlert("Имя автора не может быть пустым");  
 return;  
 }  
 if (country == null) {  
 showAlert("Страна автора не может быть пустой");  
 return;  
 }  
 if (workTitle == null || workTitle.trim().isEmpty()) {  
 showAlert("Название книги не может быть пустым");  
 return;  
 }  
 if (genre == null) {  
 showAlert("Жанр не может быть пустым");  
 return;  
 }  
 if (language == null) {  
 showAlert("Язык не может быть пустым");  
 return;  
 }  
 if (publisherName == null || publisherName.trim().isEmpty()) {  
 showAlert("Поле 'Издатель' не может быть пустым");  
 return;  
 }  
 if (editionQuantityText == null || editionQuantityText.trim().isEmpty()) {  
 showAlert("Поле 'Тираж' не может быть пустым");  
 return;  
 }  
 int printingQuantity = 0;  
 try {  
 printingQuantity = Integer.parseInt(editionQuantityText);  
 } catch (NumberFormatException e) {  
 showAlert("Тираж должен быть числом");  
 return;  
 }  
  
  
 if (!authors.contains(author)) {  
 authors.add(author);  
 }  
 Work work = new Work(workTitle, author, genre, language);  
 works.add(work);  
 Publisher publisher = new Publisher(publisherName);  
 publishers.add(publisher);  
 Edition edition = new Edition(work, publisher, printingQuantity);  
 editions.add(edition);  
 clearInputFields();  
 }  
 private void addGenre() {  
 TextInputDialog dialog = new TextInputDialog();  
 dialog.setTitle("Добавить жанр");  
 dialog.setHeaderText("Введите название жанра");  
 dialog.setContentText("Жанр:");  
 dialog.showAndWait().ifPresent(name -> {  
 Genre genre = new Genre(name);  
 genres.add(genre);  
 workGenreCombo.getItems().add(genre);  
 });  
 }  
 private void addLanguage() {  
 TextInputDialog dialog = new TextInputDialog();  
 dialog.setTitle("Добавить язык");  
 dialog.setHeaderText("Введите название языка");  
 dialog.setContentText("Язык:");  
  
 dialog.showAndWait().ifPresent(name -> {  
 Language language = new Language(name);  
 languages.add(language);  
 workLanguageCombo.getItems().add(language);  
 });  
 }  
 private void addAuthor() {  
 TextInputDialog dialog = new TextInputDialog();  
 dialog.setTitle("Добавить автора");  
 dialog.setHeaderText("Введите имя автора");  
 dialog.setContentText("Имя:");  
 dialog.showAndWait().ifPresent(name -> {  
 TextInputDialog countryDialog = new TextInputDialog();  
 countryDialog.setTitle("Добавить страну");  
 countryDialog.setHeaderText("Введите страну автора");  
 countryDialog.setContentText("Страна:");  
 countryDialog.showAndWait().ifPresent(countryName -> {  
 Country country = new Country(countryName);  
 if (!countries.contains(country)) {  
 countries.add(country);  
 authorCountryCombo.getItems().add(country);  
 }  
 Author author = new Author(name, country);  
 if (!authors.contains(author)) {  
 authors.add(author);  
 authorCombo.getItems().add(author);}});});}  
 private void clearInputFields() {  
 authorCombo.setValue(null);  
 authorCountryCombo.setValue(null);  
 workTitleField.clear();  
 workGenreCombo.setValue(null);  
 workLanguageCombo.setValue(null);  
 publisherNameField.clear();  
 editionQuantityField.clear();  
 }  
 private void showWorkDetails() {  
 WorkDetailsWindow workDetailsWindow = new WorkDetailsWindow(FXCollections.observableArrayList(editions));  
 workDetailsWindow.show();  
 }  
 private void showAlert(String message) {  
 Alert alert = new Alert(Alert.AlertType.ERROR);  
 alert.setTitle("Ошибка ввода");  
 alert.setHeaderText("Некорректный ввод");  
 alert.setContentText(message);  
 alert.showAndWait();}}

WorkDetailsWindow.java

package org.example.app;  
import javafx.beans.property.SimpleStringProperty; // Добавлена строка импорта  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.geometry.Insets;  
import javafx.scene.Scene;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.layout.VBox;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
public class WorkDetailsWindow extends Stage {  
 private TableView<Edition> editionTableView;  
 private ObservableList<Edition> editionData;  
 public WorkDetailsWindow(ObservableList<Edition> editions) {  
 initModality(Modality.APPLICATION\_MODAL);  
 setTitle("Данные о писателях и их произведениях");  
 editionData = FXCollections.observableArrayList();  
 editionData.addAll(editions);  
 editionTableView = new TableView<>();  
 editionTableView.setItems(editionData)  
 TableColumn<Edition, String> authorColumn = new TableColumn<>("Автор");  
 authorColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getWork().getAuthor().getName()));  
 authorColumn.setPrefWidth(150);  
 TableColumn<Edition, String> titleColumn = new TableColumn<>("Название");  
 titleColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getWork().getTitle()));  
 titleColumn.setPrefWidth(150);  
 TableColumn<Edition, String> genreColumn = new TableColumn<>("Жанр");  
 genreColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getWork().getGenre().getGenreName()));  
 genreColumn.setPrefWidth(150);  
 TableColumn<Edition, String> languageColumn = new TableColumn<>("Язык");  
 languageColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getWork().getLanguage().getLanguageName()));  
 languageColumn.setPrefWidth(150);  
 TableColumn<Edition, String> publisherColumn = new TableColumn<>("Издатель");  
 publisherColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getPublisher().getPublisherName()));  
 publisherColumn.setPrefWidth(150);  
 TableColumn<Edition, Integer> quantityColumn = new TableColumn<>("Тираж");  
 quantityColumn.setCellValueFactory(new PropertyValueFactory<>("printingQuantity"));  
 quantityColumn.setPrefWidth(100);  
 TableColumn<Edition, String> countryColumn = new TableColumn<>("Страна автора");  
 countryColumn.setCellValueFactory(cellData -> new SimpleStringProperty(cellData.getValue().getWork().getAuthor().getCountry().getCountryName()));  
 countryColumn.setPrefWidth(120);  
 editionTableView.getColumns().addAll(authorColumn, titleColumn, genreColumn, languageColumn, publisherColumn, quantityColumn, countryColumn);  
  
 VBox root = new VBox(10, editionTableView);  
 root.setPadding(new Insets(10));  
 setScene(new Scene(root, 1000, 400));  
 }  
}