

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

КУРСОВАЯ РАБОТА
по дисциплине «Объектно-ориентированное программирование»
ТЕМА: «СОЗДАНИЕ ПРОГРАММНОГО КОМПЛЕКСА
СРЕДСТВАМИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО
ПРОГРАММИРОВАНИЯ»

Студент гр. 3311

Баймухамедов Р. Р.

Преподаватель

Павловский М. Г.

Санкт-Петербург

2024

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Введение

ПК администрирования кинотеатра предназначен для автоматизации процессов управления данными о фильмах, репертуаре, сеансах и проданных билетах. Это приложение создано для курсового проекта по дисциплине "Объектно-ориентированное программирование".

Основание для разработки

Основанием для разработки ПК "Управление репертуаром кинотеатра" является курсовой проект, требующий реализации объектно-ориентированного подхода

Назначение разработки

ПК предназначен для автоматизации следующих задач:

- Добавление, изменение, удаление данных о фильмах, сеансах и проданных билетах.
- Просмотр репертуара на месяц.
- Получение информации о количестве проданных билетов

Требования к программе

Перечень функций

ПК должен предоставлять:

- Добавление, удаление, редактирование данных.
- Выдачу справочной информации по запросам пользователя.

Организация данных

Данные должны быть связаны через таблицу "многие ко многим". Основные таблицы:

- Фильмы
- Репертуар
- Сеансы
- Билеты

Надежность

Программа должна стабильно работать в рамках ОС Windows, предотвращать сбои и проверять корректность входных данных.

Условия эксплуатации

Программа ориентирована на однопользовательский режим работы с монопольным доступом к базе данных.

Требования к информационной и программной совместимости

ПК должен быть выполнен на языке программирования Java и работать под управлением ОС Windows. Интерфейс программы должен быть интуитивно

понятным и удобным для восприятия пользователем. Выходные данные должны быть представлены в табличной форме, обеспечивая быстрый доступ к информации о репертуаре, сеансах и проданных билетах.

Обязательные требования при реализации кода:

- Закрытые и открытые члены классов.
- Использование наследования.
- Реализация конструкторов с параметрами и конструктора копирования.
- Абстрактные базовые классы.
- Виртуальные функции.
- Обработка исключительных ситуаций.
- Динамическое создание объектов.

Рекомендуется включить:

- Дружественные функции.
- Переопределение функций или операторов.
- Шаблонные классы.

Требования к программной документации

Программная документация должна соответствовать требованиям стандартов ЕСПД и включать:

1. Описание процесса проектирования ПК.
2. Руководство пользователя для работы с приложением.
3. Исходные тексты программы с комментариями.

Стадии и этапы разработки

1. Разработка технического задания
2. Описание вариантов использования ПК
3. Создание прототипа интерфейса пользователя
4. Разработка объектной модели ПК
5. Построение диаграмм программных классов
6. Описание поведения ПК
7. Построение диаграмм действий

Порядок контроля и приёмки

В процессе приема работы устанавливается соответствие ПК и прилагаемой документации требованиям, обозначенным в техническом задании

ПРОЕКТИРОВАНИЕ ПК

Описание вариантов использования ПК

Развернутое описание функциональных требований осуществляется на этапе проектирования комплекса. Для того чтобы детализировать требования, необходимо выделить процессы, происходящие в заданной предметной области. Описание таких процессов на UML выполняется в виде прецедентов (use case). Прецеденты являются сценарием или вариантом использования ПК при взаимодействии с внешней средой. Они являются продолжением описаний требований и функциональных спецификаций, указанных в техническом задании. Прецедент изображается в виде эллипса, в котором содержится имя прецедента. Название прецедента обязательно включает в себя глагол, выражающий суть выполняемой функции. С помощью прецедентов описывается функционирование ПК с точки зрения внешнего пользователя, который называется в UML актором (actor). Актор представляет собой любую внешнюю по отношению к моделируемой системе сущность (человек, программная система, устройство), которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Актор на диаграмме изображается пиктограммой в виде человечка, под которым указано его имя. Совокупность функций, реализуемых ПК, изображается в виде диаграммы (use case diagram). Для построения диаграммы необходимо определить акторы, прецеденты (функции) и взаимоотношение между акторами и прецедентами, и между прецедентами, если один прецедент расширяет или использует другой. В языке UML для вариантов использования и действующих лиц поддерживается несколько типов связей. Это связи коммуникации (communication), использования (uses) и расширения (extends).

Связь коммуникации — это связь между прецедентом и актором. На языке UML связь коммуникации изображают в виде стрелки. Направление стрелки показывает, кто инициирует коммуникацию. При задании коммуникации необходимо указать данные, которые вводит или получает пользователь. Кроме данных на концах стрелки можно указать кратности отношения, которые характеризуют количество взаимодействующих между собой акторов и прецедентов. На диаграммах прецедентов наиболее распространенными являются две формы записи кратности 1 и 1..*. Первая форма записи означает, что один актор (прецедент) участвует во взаимодействии, а вторая форма записи, что один или несколько акторов (прецедентов) участвуют во взаимодействии.

Связь использования предполагает, что один прецедент всегда применяет функциональные возможности другого. С помощью таких связей структурируют прецеденты, показывая тем самым, какой прецедент является составной частью другого прецедента. Такой включаемый прецедент является абстрактным прецедентом в том смысле, что он не может исполняться независимо от других прецедентов, а лишь в их составе. Связь использования изображается с помощью стрелок и слова «uses» (использование).

Направление стрелки указывает, какой прецедент используется для реализации функциональности другого прецедента.

Связь расширения задается в том случае, если необходимо показать родственные отношения между двумя прецедентами. Один из них является базовым, а другой его расширением. Базовый прецедент не зависит от расширяющих прецедентов и способен функционировать без них. С другой стороны, расширяющие прецеденты без базового прецедента функционировать не могут. Связи расширения изображают в виде стрелки со словом «extends» (расширение), которая имеет направление от базового прецедента к расширяемому.

Прецеденты необходимо ранжировать, чтобы в начальных циклах разработки реализовать наиболее приоритетные из них. Разбиение функциональности системы на отдельные прецеденты служит примерно той же цели, что и разбиение сложного алгоритма на подпрограммы. Основная стратегия должна заключаться в том, чтобы сначала сконцентрировать внимание на тех прецедентах, которые в значительной мере определяют базовую архитектуру ПК.

Диаграмма прецедентов представлена на рис. 1

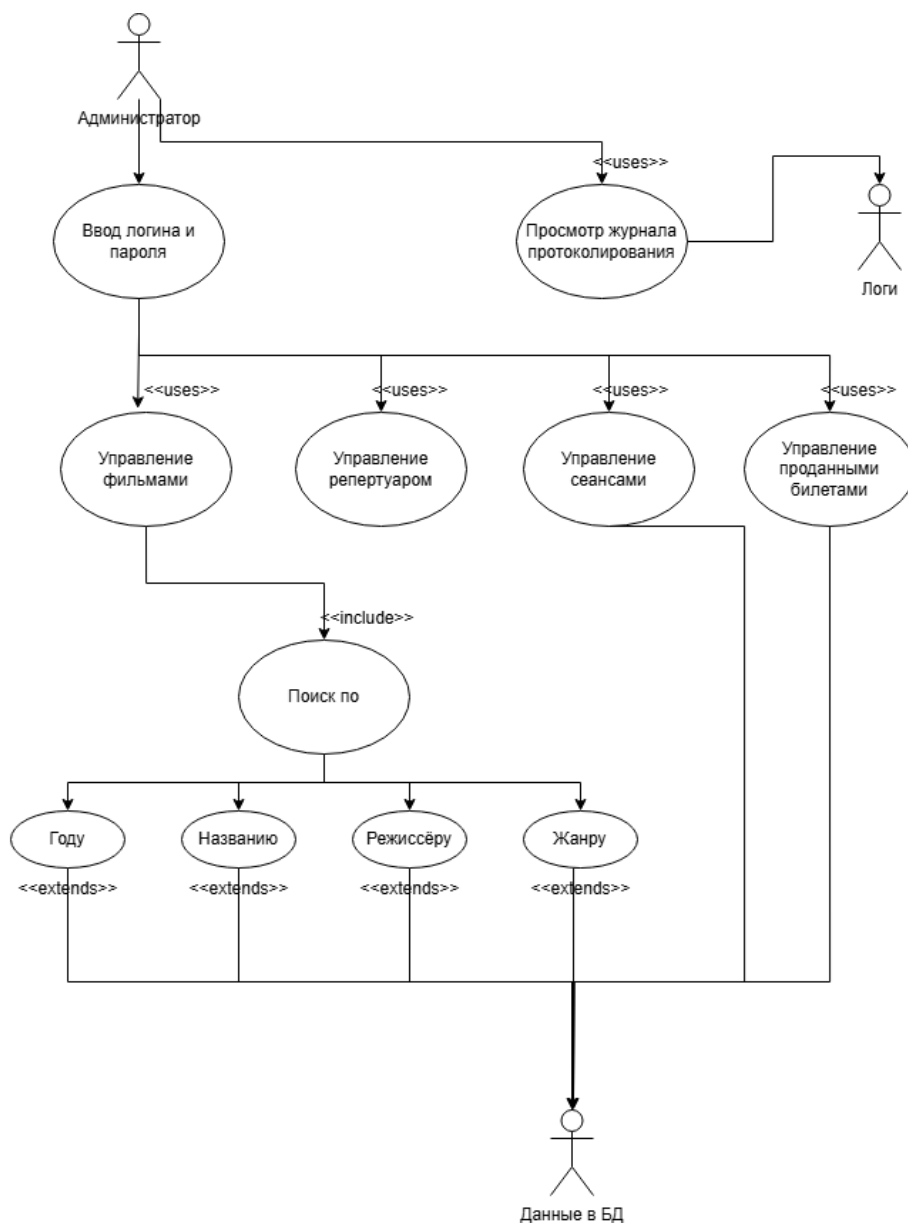


Рисунок 1

Под прецедентом управление разделом имеется в виду обобщенный прецедент, включающий в себя добавление, удаление, изменение данных раздела

Создание прототипа интерфейса пользователя

Описание прецедента выражает общую сущность процесса без детализации его реализации. Проектные решения, связанные с интерфейсом пользователя, при этом опускаются. Для разработки пользовательского интерфейса необходимо описать процесс в терминах реальных проектных решений, на основе конкретных технологий ввода-вывода информации. Когда речь идет об интерфейсе пользователя, прецеденты разбиваются на экранные формы, которые определяют содержимое диалоговых окон и описывают способы взаимодействия с конкретными устройствами. Для каждой экранной формы указываются поля ввода и перечень элементов управления, действия пользователя (нажать кнопку, выбрать пункт меню, ввести данные, нажать правую/левую кнопку мыши) и отклики системы (отобразить данные, вывести

подсказку, переместить курсор). Такое описание интерфейса представляется в виде таблицы экранных форм. Данная таблица будет отображена ниже, после определения архитектуры и макета экранной

Архитектура программы

На следующей диаграмме (рис. 2.1) отображена архитектура разделов и данных программы, а ниже приведена информация о разделе, его описание и функционал

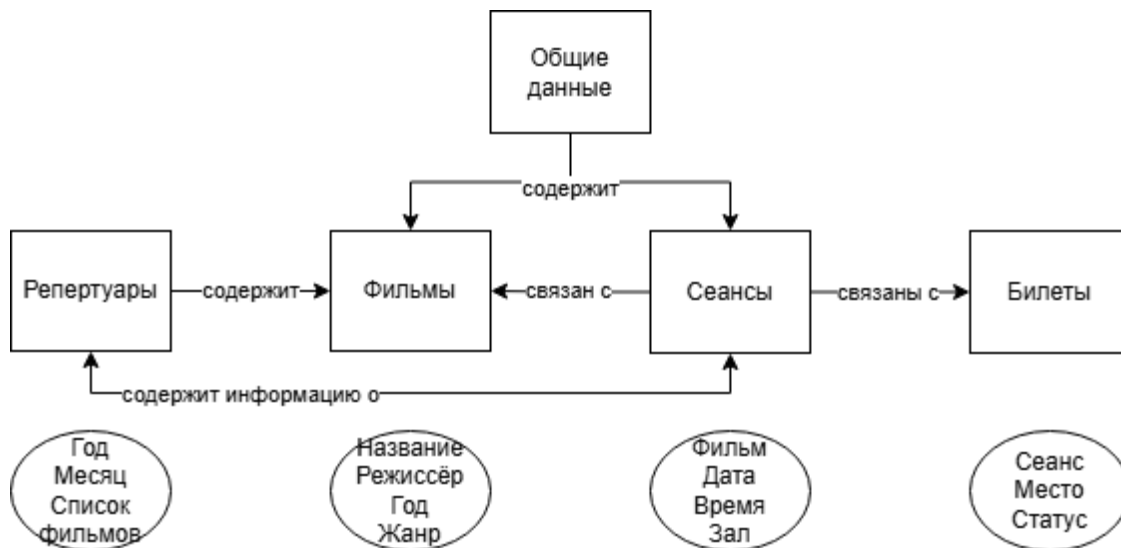


Рисунок 2.1

В данной модели данных сущности "Фильмы" и "Сеансы" связаны между собой через промежуточную сущность "Репертуары". Это позволяет реализовать связь "многое ко многим".

Сущность "Фильмы" содержит информацию о различных фильмах, таких как название фильма, год выпуска, режиссер и жанр. Каждый фильм может быть включен в несколько репертуаров, что означает связь "один ко многим" между "Фильмами" и "Репертуары".

Сущность "Репертуары" служит промежуточной таблицей, которая связывает фильмы с сеансами. Она содержит информацию о том, какие фильмы включены в каждый репертуар и в какое время они будут показаны. Один репертуар может включать несколько фильмов, и один фильм может быть включен в несколько репертуаров.

Сущность "Сеансы" содержит информацию о конкретных показах фильмов, таких как дата и время сеанса, количество мест и количество проданных билетов. Каждый сеанс связан с одним репертуаром, что означает связь "один ко многим" между "Репертуары" и "Сеансы".

Таким образом, через сущность "Репертуары" реализуется связь "многое ко многим" между "Фильмами" и "Сеансы". Это позволяет одному фильму участвовать в нескольких сеансах, и одному сеансу показывать один из нескольких фильмов, включенных в репертуар. Диаграмму описанной связи можно увидеть на рис. 2.2



Рисунок 2.2

Фильмы (Управление базой фильмов)

1. Просмотр списка фильмов:
 - Отображение списка всех фильмов в базе данных. Для каждого фильма указываются:
 - Название.
 - Режиссёр.
 - Год выпуска.
 - Жанр.
2. Добавление нового фильма:
 - Ввод информации о фильме:
 - Название, режиссёр, год, жанр.
 - После сохранения запись становится доступной для добавления сеансов в разделе "Репертуар".
3. Удаление фильма:
 - При удалении фильма автоматически удаляются связанные записи в "Репертуаре" и "Сеансах".
 - Удаление возможно только если у фильма нет проданных билетов. В противном случае система выводит предупреждение.
4. Редактирование информации о фильме:
 - Возможность изменить название, режиссёра, год или жанр фильма.
 - Изменения немедленно отражаются в связанных разделах ("Репертуар" и "Сеансы").

Репертуар (Управление расписанием кинотеатра)

1. Просмотр репертуара на месяц:
 - Отображение списка фильмов и сеансов на текущий месяц с указанием:
 - Названия фильма.
 - Даты и времени сеанса.
 - Зала.
2. Добавление сеансов фильма:
 - Пользователь выбирает фильм из списка.
 - Вводит дату, время, зал.
 - Запись синхронизируется с разделом "Сеансы".
3. Удаление сеанса:

- Удаляется выбранный сеанс из расписания.
 - Сеанс и все связанные данные удаляются из раздела "Сеансы".
4. Редактирование информации о сеансе:
- Пользователь может изменить время, дату или зал для выбранного сеанса.
 - Все изменения автоматически обновляются в "Сеансах".

Сеансы (Управление билетами и деталями сеансов)

1. Просмотр информации о сеансах конкретного фильма:
 - После выбора фильма отображаются все его сеансы с указанием:
 - Даты, времени, зала.
 - Проданных и доступных билетов.
2. Продажа билетов:
 - Пользователь может увидеть количество проданных билетов и оставшиеся места.
3. Информация о зале:
 - Отображение количества мест в зале и текущего статуса мест (свободные/занятые).
4. Добавление новых сеансов:
 - Пользователь может добавить новый сеанс (с указанием фильма, даты, времени, зала).
 - Новый сеанс автоматически добавляется в "Репертуар".
5. Удаление и изменение текущих сеансов:
 - Удаление сеанса из "Сеансов" автоматически удаляет его из "Репертуара".
 - Изменение данных о сеансе (дата, время, зал) синхронизируется с "Репертуаром".

Билеты (Управление продажами и учётом билетов)

1. Просмотр проданных билетов на сеанс:
 - Отображение списка проданных билетов с указанием:
 - Номера места.
 - Статуса (продан/свободен).
 - Общая сумма продаж на выбранный сеанс.
2. Продажа билетов:
 - Выбор сеанса и номера места.
 - Ввод данных покупателя.
 - После подтверждения продажа сохраняется, место становится занятым.
3. Просмотр статистики продаж:
 - Отображение общего количества проданных билетов за месяц.
4. Информация о доступных местах:
 - Список свободных мест для выбранного сеанса.
 - Возможность быстро выбрать свободное место для продажи.

Логика взаимодействия разделов “Сеансы” и “Репертуар”

Основная идея

- Раздел "Репертуар" оперирует более высоким уровнем абстракции, работая с расписанием на месяц. Он предназначен для управления общим списком фильмов и их сеансов.
- Раздел "Сеансы" работает с деталями каждого отдельного сеанса, предоставляя функционал для управления билетами и информации о зале.

Связь и функциональные зависимости

1. Добавление и удаление сеансов:
 - Когда администратор добавляет сеанс через "Репертуар", создаётся соответствующая запись в "Сеансах", включающая дату, время и зал.
 - Удаление сеанса в "Репертуаре" автоматически удаляет связанные записи в "Сеансах" и все связанные билеты.
2. Редактирование информации о сеансе:
 - Изменение информации о сеансе в "Репертуаре" синхронизируется с соответствующей записью в "Сеансах". Например, изменение времени или зала в "Репертуаре" обновляет эти данные в "Сеансах".
3. Просмотр информации:
 - В "Репертуаре" доступен общий просмотр сеансов по месяцам, без детальной информации о зале и билетах.
 - Для более подробной информации (проданные билеты, зал) пользователь переходит в "Сеансы", где данные фильтруются по выбранному сеансу.

Пример логики взаимодействия между "Сеансами" и "Билетами":

- При создании нового сеанса система автоматически генерирует записи о всех доступных местах для зала.
- При продаже билета запись о месте обновляется, и его статус меняется на "продан".
- Раздел "Сеансы" предоставляет быстрый доступ к данным о проданных и свободных билетах.

Структура базы данных

Таблица раздела “Фильмы”

Поле	Тип данных	Описание
movie_id	INT (PK)	Уникальный идентификатор фильма
movie_name	TEXT	Название фильма
director	TEXT	Имя режиссёра
movie_year	INT	Год выпуска фильма
genre	TEXT	Жанр фильма

Таблица раздела “Репертуар”

Поле	Тип данных	Описание
rep_id	INT (PK)	Уникальный идентификатор записи
movie_id	INT (FK)	Внешний ключ, указывает на фильм
date	DATE	Дата показа фильма
time	TIME	Время показа фильма
host	INT	Номер зала

Таблица раздела “Сеансы”

Поле	Тип данных	Описание
session_id	INT (PK)	Уникальный идентификатор сеанса
rep_id	INT (FK)	Внешний ключ, указывает на запись в репертуаре
numb_seats	INT	Общее количество мест в зале
numb_sold_tickets	INT	Количество проданных билетов

Таблица раздела “Билеты”

Поле	Тип данных	Описание
ticket_id	INT (PK)	Уникальный идентификатор билета
session_id	INT (FK)	Внешний ключ, указывает на сеанс
numb_of_seat	INT	Номер места в зале
status	TEXT	Статус билета (свободен/продан)
date_bought	DATETIME	Дата и время покупки билета

Первичный ключ (Primary Key, PK) — это уникальный идентификатор для каждой записи в таблице базы данных. Он обеспечивает уникальность каждой записи и не может содержать значения NULL.

Внешний ключ (Foreign Key, FK) — это столбец или набор столбцов в одной таблице, который ссылается на первичный ключ в другой таблице. Он используется для установления и поддержания связей между таблицами. Внешний ключ обеспечивает ссылочную целостность, то есть значения в столбце внешнего ключа должны соответствовать значениям первичного ключа в другой таблице.

Представление базы данных в виде диаграммы изображено на рис. 3.1

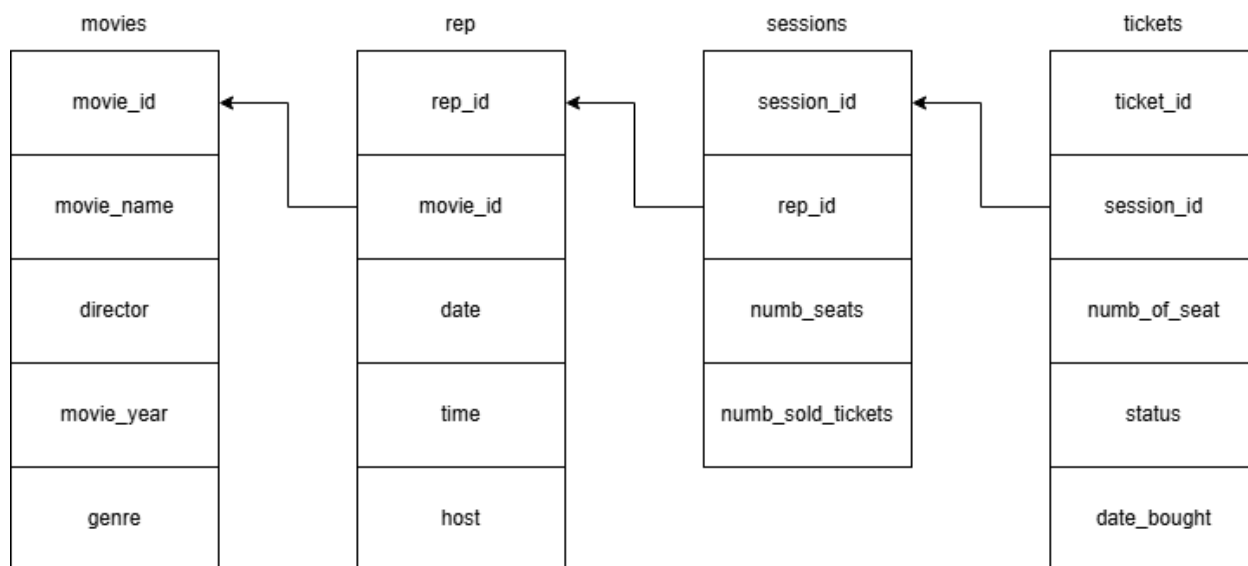


Рисунок 3.1

Реализация базы данных

В ходе разработки приложения для администратора кинотеатра было принято решение использовать SQLite в качестве системы управления базами данных. Данное решение обусловлено несколькими ключевыми факторами, которые соответствуют требованиям проекта и общей концепции программной реализации.

SQLite представляет собой компактную, высокоэффективную и кроссплатформенную реляционную базу данных, которая идеально подходит для встроенных приложений, таких как наше. Она предоставляет все необходимые возможности реляционной модели данных и соответствует стандарту SQL, что делает её отличным выбором для создания и управления структурированными данными в приложении. Более того SQLite входит в тройку самых популярных реляционных баз данных, что представлено на рис. 3.2 ([Источник Stack Overflow Developer Survey 2021](#))

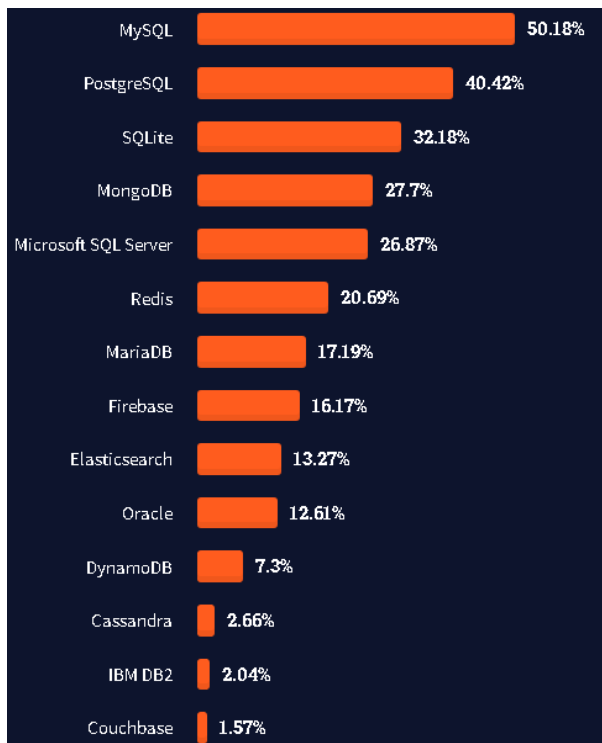


Рисунок 3.2

Преимущества выбора SQLite

1. Лёгкость и удобство использования

SQLite — это библиотека, которая интегрируется непосредственно в приложение без необходимости установки отдельного серверного ПО. Это упрощает процесс настройки и минимизирует затраты на администрирование. Для запуска достаточно лишь одного файла базы данных, что делает SQLite особенно удобной для небольших и средних проектов.

2. Эффективность и производительность

SQLite разработана таким образом, чтобы быть максимально быстрой и эффективной в обработке запросов. В рамках приложения администратора кинотеатра это особенно важно, так как функционал предусматривает работу с репертуаром, сеансами, билетами и фильмами.

3. Отсутствие зависимостей

В отличие от других СУБД, SQLite не требует наличия серверной части. Это позволяет исключить потенциальные проблемы, связанные с совместимостью программного обеспечения, а также снизить требования к аппаратному обеспечению.

4. Поддержка SQL-стандарта

SQLite поддерживает стандарт SQL, что упрощает написание запросов для работы с таблицами, связи между данными и выполнение сложных операций.

Интеграция и инструменты взаимодействия

SQLite была интегрирована в приложение посредством JDBC (Java Database Connectivity). Это позволяет напрямую взаимодействовать с базой данных, реализуя функционал просмотра, добавления, редактирования и удаления данных через интерфейс приложения.

В качестве версии SQLite было выбрано SQLite version 3.47.2. Также для работы с SQLite было установлен инструмент для удобного взаимодействия с БД — SQLite Browser. Интерфейс можно увидеть на рис. 3.3

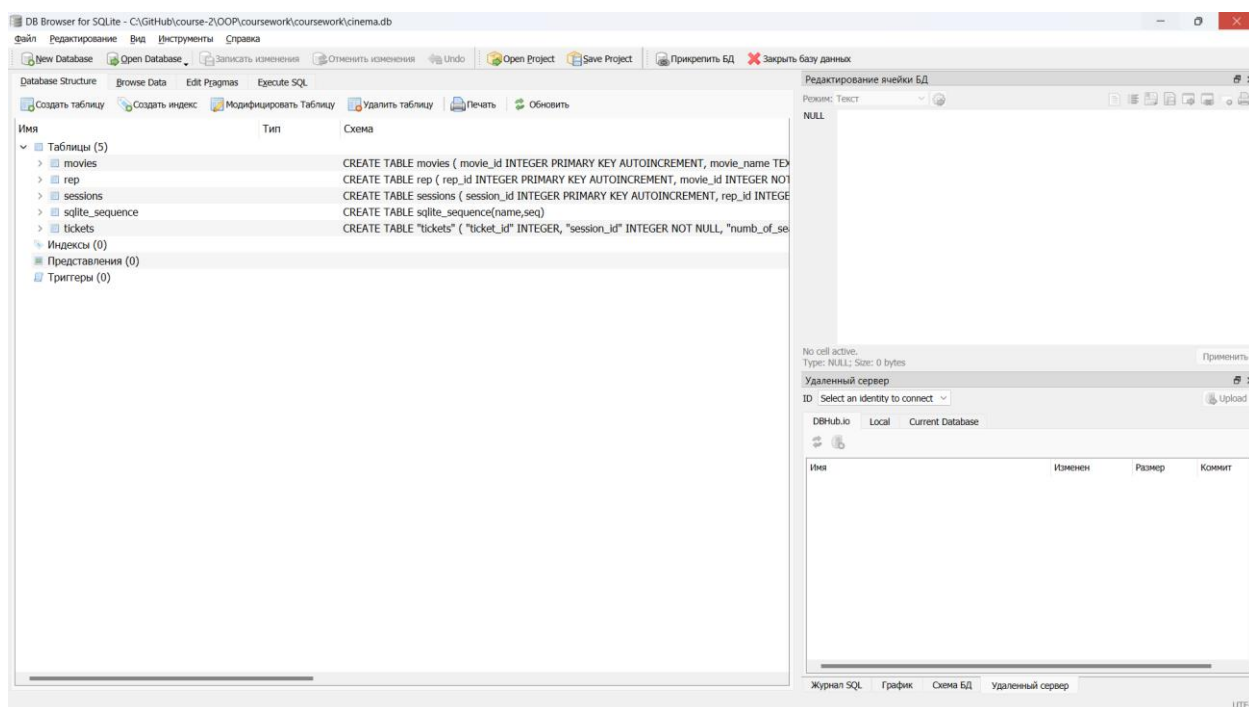


Рисунок 3.3

Журнал протоколирования (Логи)

В рамках текущего проекта терминология "логи" и "журнал протоколирования" будет использоваться как синонимы, подразумевая одно и то же — запись различных событий и действий, происходящих в приложении. В качестве инструмента для логирования был выбран встроенный инструментарий Java, а не Log4j или Logback. Это решение было принято по нескольким причинам. Во-первых, встроенный инструментарий Java предоставляет все необходимые функции для логирования, такие как различные уровни логирования, форматирование сообщений и настройка вывода в файлы или консоль. Это позволяет удовлетворить потребности проекта без необходимости внедрения дополнительных библиотек.

Во-вторых, использование встроенного инструментария Java упрощает процесс разработки и поддержки приложения. Встроенный инструментарий не требует дополнительных зависимостей и конфигураций, что делает его более удобным для использования в небольших и средних проектах. Это особенно важно для однопользовательского приложения, где требования к логированию не столь высоки, как в многопользовательских или распределенных системах.

Кроме того, в данном проекте было принято решение сохранять логи в файл, а не в базу данных. Это связано с тем, что хранение логов в файле обеспечивает простоту и надежность, а также упрощает процесс доступа к логам для анализа

и отладки. Файловая система позволяет легко архивировать и удалять старые логи, не нагружая при этом базу данных и не требуя дополнительных ресурсов для управления логами.

Учётные записи и права администратора

В данном проекте предусмотрено использование только одной учётной записи с административными правами. Эта запись позволяет управлять всеми аспектами приложения, включая работу с фильмами, репертуаром, сеансами и билетами. Для входа в систему администратор использует логин и пароль, которые по умолчанию заданы как *admin*.

Поскольку проект разрабатывается для одиночного пользователя с административными функциями, реализация других уровней доступа или дополнительных ролей (например, кассиров, менеджеров) не требуется. Такой подход упрощает архитектуру приложения, минимизируя затраты времени на разработку сложных систем разграничения прав.

Если в будущем появится необходимость в поддержке нескольких пользователей или разделении их ролей, это можно будет доработать. Однако на текущем этапе реализация ограничивается только одним пользователем-администратором, что вполне соответствует поставленным требованиям.

Макет экранной формы

Введите логин	<input type="text"/>
Введите пароль	<input type="password"/>
<input type="button" value="Отмена"/>	<input type="button" value="Войти"/>

Рисунок 4.1.1 (Экранная форма входа)

Введите логин	<input type="text" value="admin"/>
Введите пароль	<input type="password" value="*****"/>
<input type="button" value="Отмена"/>	<input type="button" value="Войти"/>

Рисунок 4.1.2 (Заполненная экранная форма входа)

Х
Неправильный логин или пароль
[ок]

Рисунок 4.1.3 (Экранная форма входа неправильного логина/пароля)

Приложение

Фильмы

Репертуар

Сеансы

Билеты

Логи

Выход

+

-

Название	Режиссёр	Год выпуска	Жанр

Поиск по

Названию ▼

Поля для ввода...

Q

Рисунок 4.2 (Экранная форма приложения)

Приложение

Фильмы

Репертуар

Сеансы

Билеты

Логи

Выход

+

-

Название

Поиск по

Названию ▼

Журнал протоколирования

23:30:49,243 DEBUG main CinemaList:show:30 - App is started
23:31:03,442 DEBUG AWT-EventQueue-0
CinemaList:saveToFile:328 - The method of saving the file was called
23:31:10,847 DEBUG AWT-EventQueue-0
CinemaList:addMovie:224 - Method of adding the movie was called
23:31:17,285 DEBUG AWT-EventQueue-0
CinemaList:loadFromFile:368 - The method of loading from file was called
23:31:51,219 DEBUG main CinemaList:show:30 - App is started
23:32:01,836 DEBUG AWT-EventQueue-0
CinemaList:saveToFile:328 - The method of saving the file was called
20:39:51,356 DEBUG main CinemaList:show:30 - App is started

Рисунок 4.3 (Экранная форма приложения при нажатии вкладки “Логи”)

Название

Режиссёр

Год

Жанр

Ок

Отмена

Рисунок 4.4 (Экранная форма ввода данных)

Название

Режиссёр

X

Неверный формат данных

[ок]

Ок Отмена

Рисунок 4.5 (Экранная форма ввода данных неправильного формата)

Вы уверены, что хотите удалить данные?

да нет

Рисунок 4.6 (Экранная форма подтверждения удаления данных)

В таблицах ниже представлено описание экранных форм и их взаимодействие с пользователем

Таблица 4.1 (Панель вкладок)

Элементы управления	Действия пользователя	Отклик системы
Меню с вкладками:		
- "Фильмы"	Выбор вкладки "Фильмы"	Открывается экранная форма "Фильмы".
- "Репертуар"	Выбор вкладки "Репертуар"	Открывается экранная форма "Репертуар".
- "Сеансы"	Выбор вкладки "Сеансы"	Открывается экранная форма "Сеансы".
- "Билеты"	Выбор вкладки "Продажа билетов"	Открывается экранная форма "Продажа билетов".
- "Логи"	Выбор вкладки "Логи"	Открывается log.txt (журнал протоколирования приложения) поверх окна приложения
- "Выход"	Нажатие кнопки "Выход"	Приложение закрывается.

Таблица 4.2 (Панель инструментов)

Элементы управления	Действия пользователя	Отклик системы
Кнопки:		
- "Добавить данные"	Нажатие кнопки "Добавить данные"	Открывается окно для ввода информации о новых данных.
- "Изменить данные"	Выбор данных из списка и нажатие кнопки "Изменить данные"	Открывается окно для редактирования выбранных данных
- "Удалить данные"	Выбор данных из списка и нажатие кнопки "Удалить данные"	Выбранные данные удаляются из базы данных.
- "Сохранить данные"	Нажатие кнопки "Сохранить данные"	Сохранение текущих данных в базу данных.
- "Загрузить данные"	Нажатие кнопки "Загрузить данные"	Загрузка данных из базы данных в приложение.

Таблица 4.3 (Панель поиска)

Элементы управления	Действия пользователя	Отклик системы
Поле выбора:	Выбор поля для поиска (например, название фильма)	Определяется критерий для выполнения поиска.
Поле ввода:	Ввод значения для поиска	Значение используется для выполнения поиска.
Кнопка "Найти"	Нажатие кнопки "Найти"	Отображаются результаты поиска на основе заданных условий.

Таблица 4.4 (Панель входа)

Элементы управления	Действия пользователя	Отклик системы
Поле ввода "Логин"	Ввод логина	Используется для проверки учетных данных.
Поле ввода "Пароль"	Ввод пароля	Используется для проверки учетных данных.
Кнопка "Войти"	Нажатие кнопки "Войти"	При правильном вводе учетных данных открывается приложение.
		При неправильном вводе отображается сообщение об ошибке.

Кнопка "Отмена"	Нажатие кнопки "Отмена"	Приложение закрывается.
--------------------	----------------------------	-------------------------

Взаимодействие с данными при помощи панели инструментов

На всех вкладках управление данными работает одинаково: добавление, изменение, удаление, сохранение и загрузка. Единственное различие заключается в том, какая экранная форма и структура данных используются для каждой вкладки. Например, вкладка "Фильмы" предназначена для работы с информацией о фильмах, тогда как вкладка "Репертуар" отображает расписание и связанную информацию о фильмах.

Реализация экранной формы

В качестве графической библиотеки для разработки пользовательского интерфейса приложения была выбрана Swing. Это решение было обусловлено тем, что Swing предоставляет богатый набор компонентов и возможностей для создания нативных приложений на языке Java. Swing позволяет легко управлять графическими элементами, такими как кнопки, панели, меню и окна, обеспечивая при этом высокую гибкость и производительность. Кроме того, для использования в интерфейсе были взяты иконки с сайта lucide.dev, который предоставляет широкий выбор качественных и современных иконок, что позволило улучшить визуальную привлекательность и удобство пользовательского интерфейса.

В ходе проектирования пользовательского интерфейса было принято решение о внесении нескольких изменений в структуру и расположение элементов. В частности, был изменен порядок расположения панели инструментов и панели вкладок. Это было сделано для улучшения удобства использования и навигации по приложению. Кроме того, функции "Логи" и "Выход" были перенесены в панель инструментов, что позволило пользователям быстрее получать доступ к этим важным функциям. Также был изменен порядок элементов в самой панели инструментов, чтобы сделать интерфейс более интуитивно понятным и логичным. Эти изменения направлены на повышение удобства и эффективности работы пользователей с приложением.

Ниже представлены рисунки фактической экранной формы

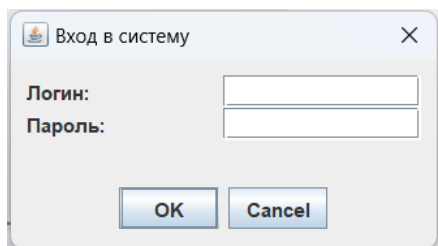


Рисунок 5.1.1 (Вход в систему)

Вход в систему

Логин: user

Пароль:

OK Cancel

Рисунок 5.1.2 (Заполненная форма входа в систему)

Ошибка

Неправильный логин или пароль.

OK

Рисунок 5.1.3 (Уведомление о неправильном логине или пароле при входе в систему)

Cinema Admin App

Фильмы Репертуар Сеансы Билеты

movie_id	movie_name	director	movie_year	genre
5	Whiplash	D. Shazell	2013	Drama
6	Fight Club	D. Fincher	1999	Thriller
8	The Irishman	M. Scorsese	2019	Thriller
9	Scott Pilgrim vs the World	E. Right	2010	Comedy
10	The Hush	A. Kernel	2012	Horror
11	Arithmiya	B. Hlebnikov	2017	Drama
13	The Social Network	D. Fincher	2010	Drama
14	Inception	C. Nolan	2010	Sci-Fi
15	The Dark Knight	C. Nolan	2008	Action
16	Pulp Fiction	Q. Tarantino	1994	Crime
17	Forrest Gump	R. Zemeckis	1994	Drama
18	The Shawshank Redemption	F. Darabont	1994	Drama
19	The Godfather	F. Coppola	1972	Crime
20	The Silence of the Lambs	J. Demme	1991	Thriller
21	Saving Private Ryan	S. Spielberg	1998	War
22	The Green Mile	F. Darabount	1999	Drama
23	Gladiator	R. Scott	2000	Action

Поиск по: Ключевое слово Поиск Сброс

Рисунок 5.2.1 (Стандартное окно приложения (выбран раздел “Фильмы”))

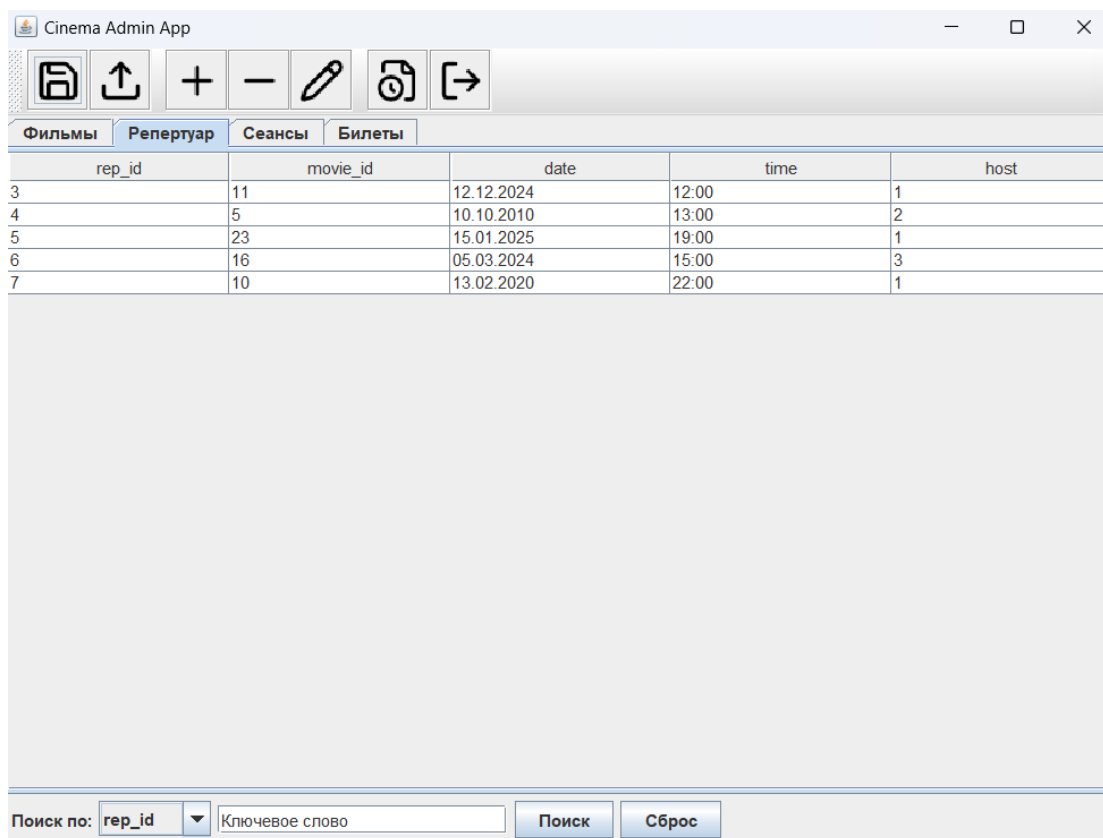


Рисунок 5.2.2 (Стандартное окно приложения (выбран раздел “Репертуар”))

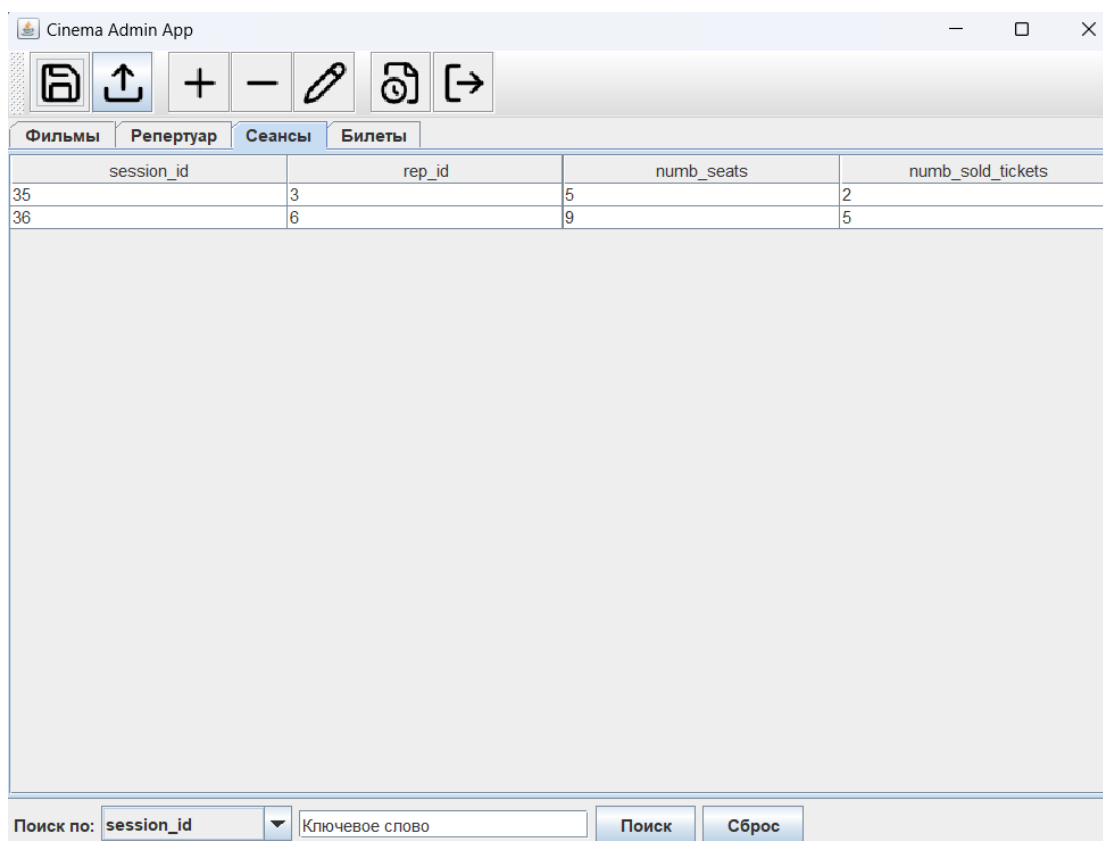


Рисунок 5.2.3 (Стандартное окно приложения (выбран раздел “Сеансы”))

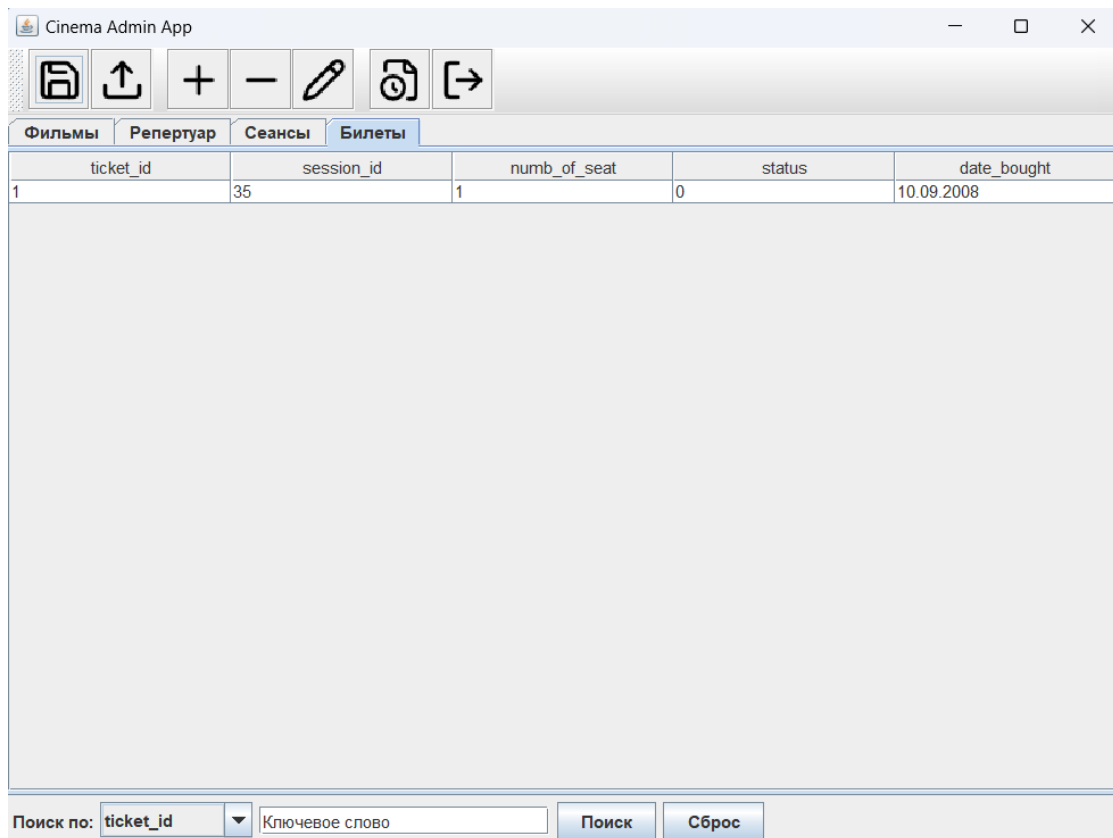


Рисунок 5.2.4 (Стандартное окно приложения (выбран раздел “Билеты”))

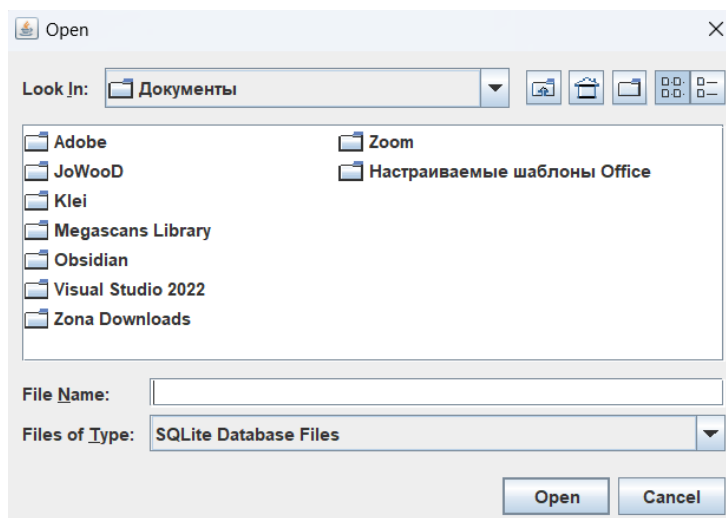


Рисунок 5.3 (Открыть файл базы данных)

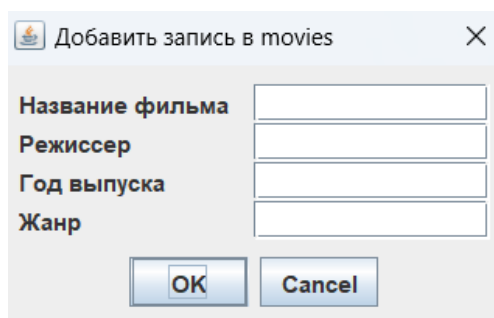


Рисунок 5.4.1 (Экранная форма добавления записи в раздел “Фильмы”)

Добавить запись в гер

ID фильма: Whiplash (5)

Дата:

Время:

Зал:

OK Cancel

Рисунок 5.4.2 (Экранная форма добавления записи в раздел “Репертуар”)

Добавить запись в sessions

ID репертуара: 3

Количество мест:

Количество проданных билетов:

OK Cancel

Рисунок 5.4.3 (Экранная форма добавления записи в раздел “Сеансы”)

Добавить запись в tickets

ID сеанса: 35

Номер места:

Статус:

Дата покупки:

OK Cancel

Рисунок 5.4.4 (Экранная форма добавления записи в раздел “Билеты”)

Ошибка

Неверный формат даты! Ожидается формат: дд.мм.гггг.

OK

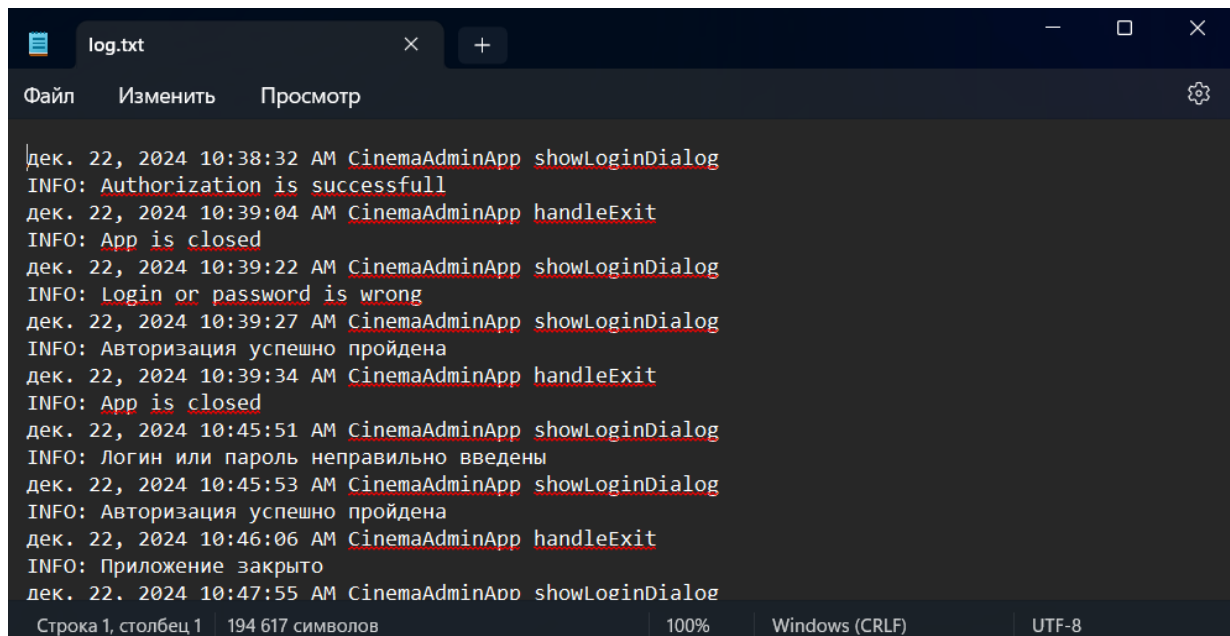
Рисунок 5.5 (Экранная форма неверного формата даты)

Подтверждение удаления

Вы уверены, что хотите удалить запись с ID: 5?

Yes No

Рисунок 5.6 (Экранная форма подтверждения удаления данных)



```
дек. 22, 2024 10:38:32 AM CinemaAdminApp showLoginDialog
INFO: Authorization is successfull
дек. 22, 2024 10:39:04 AM CinemaAdminApp handleExit
INFO: App is closed
дек. 22, 2024 10:39:22 AM CinemaAdminApp showLoginDialog
INFO: Login or password is wrong
дек. 22, 2024 10:39:27 AM CinemaAdminApp showLoginDialog
INFO: Авторизация успешно пройдена
дек. 22, 2024 10:39:34 AM CinemaAdminApp handleExit
INFO: App is closed
дек. 22, 2024 10:45:51 AM CinemaAdminApp showLoginDialog
INFO: Логин или пароль неправильно введены
дек. 22, 2024 10:45:53 AM CinemaAdminApp showLoginDialog
INFO: Авторизация успешно пройдена
дек. 22, 2024 10:46:06 AM CinemaAdminApp handleExit
INFO: Приложение закрыто
дек. 22, 2024 10:47:55 AM CinemaAdminApp showLoginDialog
```

Строка 1, столбец 1 | 194 617 символов | 100% | Windows (CRLF) | UTF-8

Рисунок 5.7 (“Логи”)

Валидация данных

В данном проекте особое внимание уделяется валидации данных, что является критически важным аспектом для обеспечения надежности и точности всей системы. Валидация данных включает в себя проверку корректности дат, чисел и временных меток, что позволяет избежать ошибок и несоответствий, которые могут привести к неправильным выводам и решениям. Проверка дат гарантирует, что все временные метки соответствуют реальным событиям и хронологическому порядку, что особенно важно для анализа временных рядов и исторических данных. Валидация чисел обеспечивает точность вычислений и предотвращает ошибки, связанные с неправильным вводом или интерпретацией числовых значений. Проверка времени позволяет синхронизировать различные процессы и события, что особенно важно в системах, где точность времени играет ключевую роль, таких как финансовые транзакции или управление реальным временем. В целом, валидация данных является фундаментальным элементом, который обеспечивает целостность и достоверность информации, что, в свою очередь, способствует принятию обоснованных решений и повышению общей эффективности системы.

Построение диаграммы программных классов

Диаграмма классов (class diagram) иллюстрирует спецификации будущих программных классов и интерфейсов. Она строится на основе объектной модели. В описание класса указываются три раздела: имя класса, состав компонентов класса и методы класса. Графически класс изображается в виде прямоугольника. Имя программного класса может совпадать с именем сущности или быть другим. Но поскольку для записи идентификаторов переменных в языках программирования используют латинские буквы, то и имена программных классов, и имена их атрибутов, как правило, записываются латинскими буквами. Атрибуты и операции класса

перечисляются в горизонтальных отделениях этого прямоугольника. Атрибутам и методам классов должны быть присвоены права доступа. Права доступа помечаются специальными знаками:

+ - означает открытый (public) доступ;

— - означает скрытый (private) доступ;

- означает наследуемый (protected) доступ.

При описании атрибутов после двоеточия указывается их тип, а при описании методов класса возвращаемое значение (для конструкторов возвращаемое значение не указывается).

В диаграмме классов могут вводиться дополнительно новые атрибуты, операции и связи или осуществляться конкретизация ассоциаций, указанных в объектной модели. На диаграмме классов могут быть три вида отношений: ассоциация, агрегация и наследование.

На диаграмме классов ассоциация имеет такое же обозначение, как и в объектной модели. На линиях ассоциации может присутствовать стрелка. Это стрелка видимости, которая показывает направление посылки запросов в ассоциации. Стрелка видимости также показывает, какой из классов содержит компоненты для реализации отношения ассоциации, иными словами, кто является инициатором посылки запроса к другому объекту. Ассоциация без стрелки является двунаправленной.

Агрегирование — это отношение между классами типа целое/часть. Агрегируемый класс в той или иной форме является частью агрегата. На практике это может быть реализовано по-разному. Например, объект класса-агрегата может хранить объект агрегируемого класса, или хранить ссылку на него. Агрегирование изображается на диаграмме полым ромбом на конце линии со стороны агрегирующего класса (агрегата). Если агрегируемый объект может быть создан только тогда, когда создан агрегат, а с уничтожением агрегата уничтожаются и все агрегируемые объекты, то такое агрегирование называется сильным и отображается в виде закрашенного ромба.

Наследование — это отношение типа общее-частное между классами. Его следует вводить в том случае, когда поведение и состояние различных классов имеют общие черты. Наследование связывает конкретные классы с общими или в терминологии языков программирования производные классы (подклассы) с базовыми классами (суперклассами). На диаграммах наследование изображается в виде стрелки с полым треугольником, идущей от производного класса к базовому. Если один производный класс наследует несколько базовых, то такое наследование называется множественным.

Диаграмма классов представлена на рис. 6

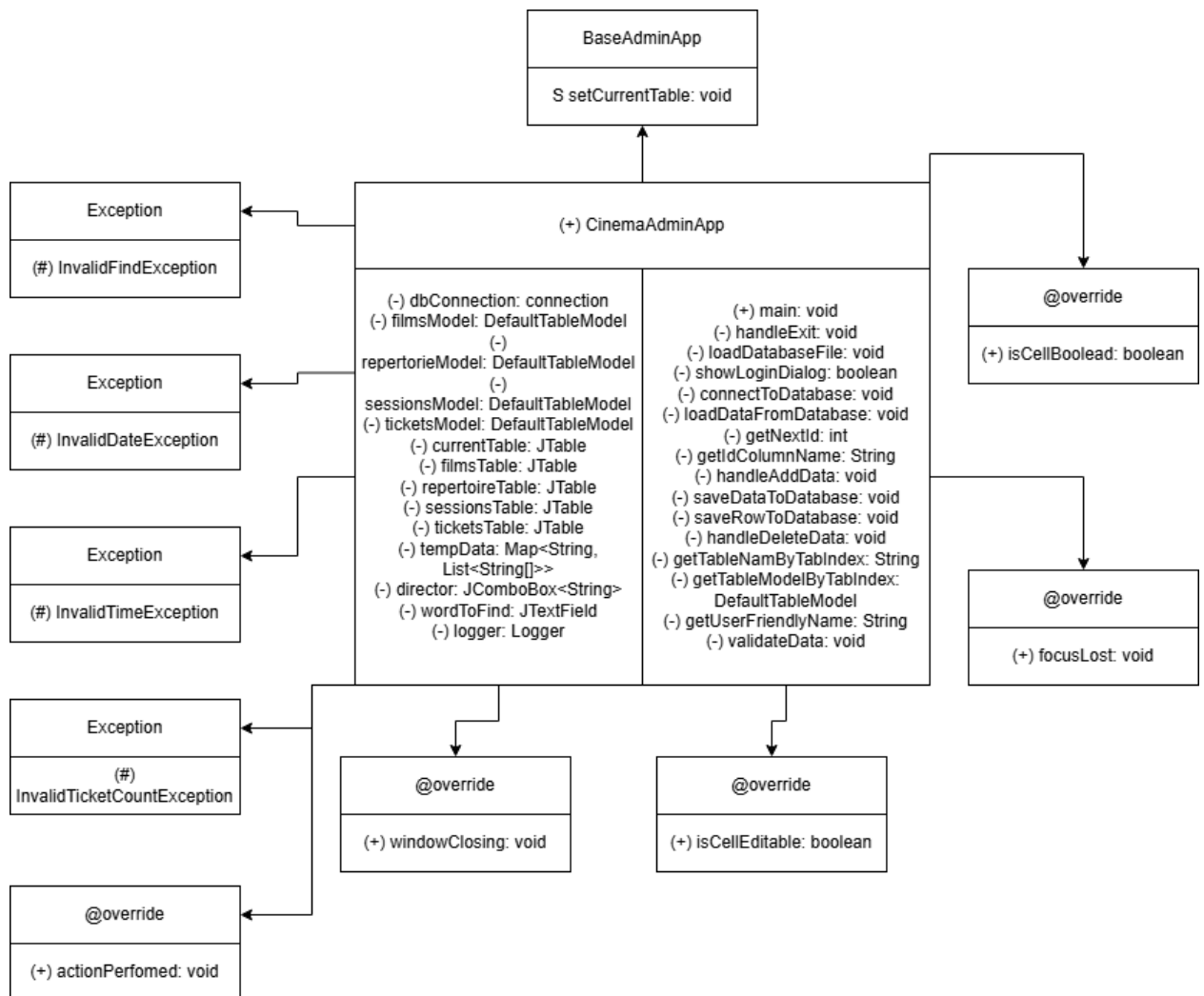


Рисунок 6

Описание поведения ПК

Поведение ПК представляет собой описание того, какие действия выполняет ПК, без определения механизма их реализации. Одной из составляющей такого описания является диаграмма последовательностей (sequence diagram). Диаграмма последовательностей является схемой, которая для определенного сценария прецедента показывает генерируемые пользователями и объектами события (запросы) на выполнение некоторой операции и их порядок. Диаграммы последовательности имеют две размерности: вертикальная представляет время, горизонтальная - различные объекты. Чтобы построить диаграмму последовательностей необходимо выполнить следующие действия:

1. Идентифицировать пользователей и объекты программных классов, участвующие в начальной стадии реализации сценария прецедента, и их изображения в виде прямоугольников расположить наверху в одну линию. Для каждого пользователя и объекта нарисовать вертикальную пунктирную линию, которая является линией их жизни. Внутри прямоугольника указываются подчеркнутое имя объекта и имя класса, к которому принадлежит объект.

2. Из объектной модели выбрать те операции, которые участвуют в реализации сценария. Если такие операции не были определены при построении диаграммы программных классов, то необходимо их описать и внести в модель.
3. На диаграмме последовательностей каждому запросу на выполнение операции должна соответствовать горизонтальная линия со стрелкой, начинающаяся от вертикальной линии того пользователя или объекта, который вызывает операцию, и заканчивающаяся на линии жизни того пользователя или объекта, который будет ее выполнять. Над стрелкой указывается номер операции, число итераций, имя операции и в скобках ее параметры. После описания операции может следовать комментарий, поясняющий смысл операции и начинающийся со знака "//".

Операция, которая реализует запрос, на линии жизни объекта обозначается прямоугольником. Порядок выполнения операций определяется ее номером, который указывается перед именем, и положением горизонтальной линии на диаграмме. Чем ниже горизонтальная линия, тем позже выполняется операция. В диаграммах последовательности принято применять вложенную систему нумерации, так как это позволяет отобразить их вложенность. Нумерация операций каждого уровня вложенности должна начинаться с 1.

На диаграмме последовательностей можно описать вызов операции по условию (конструкция if-else) и показать моменты создания и уничтожения объектов. Если объект создается или уничтожается на отрезке времени, представленном на диаграмме, то его линия жизни начинается и заканчивается в соответствующих точках, в противном случае линия жизни объекта проводится от начала до конца диаграммы. Символ объекта рисуется в начале его линии жизни; если объект создается не в начале диаграммы, то сообщение о создании объекта рисуется со стрелкой, проведенной к символу объекта. Если объект уничтожается не в конце диаграммы, то момент его уничтожения помечается большим крестиком "X".

Диаграмма последовательности занесения измененных строк в базу данных изображена на рис. 7

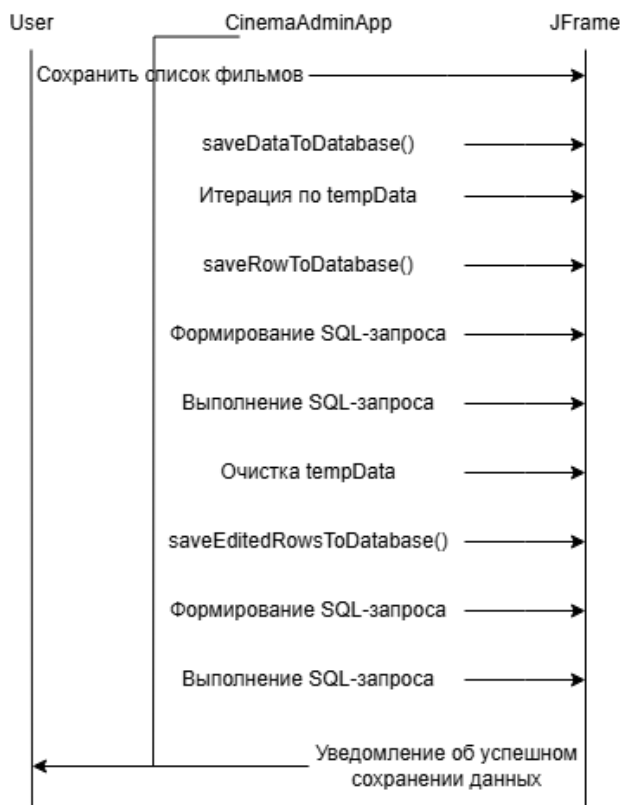


Рисунок 7

Описание шагов:

1. Пользователь нажимает кнопку "Сохранить список фильмов":
 - Пользователь взаимодействует с CinemaAdminApp через JFrame.
2. CinemaAdminApp вызывает метод saveDataToDatabase:
 - CinemaAdminApp начинает процесс сохранения данных.
3. saveDataToDatabase итерируется по временному списку tempData и вызывает метод saveRowToDatabase для каждой строки:
 - saveDataToDatabase проходит по всем временным данным и сохраняет их в базу данных.
4. saveRowToDatabase формирует SQL-запрос и выполняет его с помощью объекта Statement:
 - saveRowToDatabase создает SQL-запрос для вставки данных и выполняет его.
5. После успешного выполнения всех запросов, saveDataToDatabase очищает временный список tempData:
 - Временные данные очищаются после успешного сохранения.
6. CinemaAdminApp вызывает метод saveEditedRowsToDatabase для сохранения измененных строк:
 - CinemaAdminApp начинает процесс сохранения измененных строк.
7. saveEditedRowsToDatabase формирует SQL-запросы для обновления строк и выполняет их с помощью объекта Statement:
 - saveEditedRowsToDatabase создает SQL-запросы для обновления данных и выполняет их.

8. Пользователь получает уведомление об успешном сохранении данных:

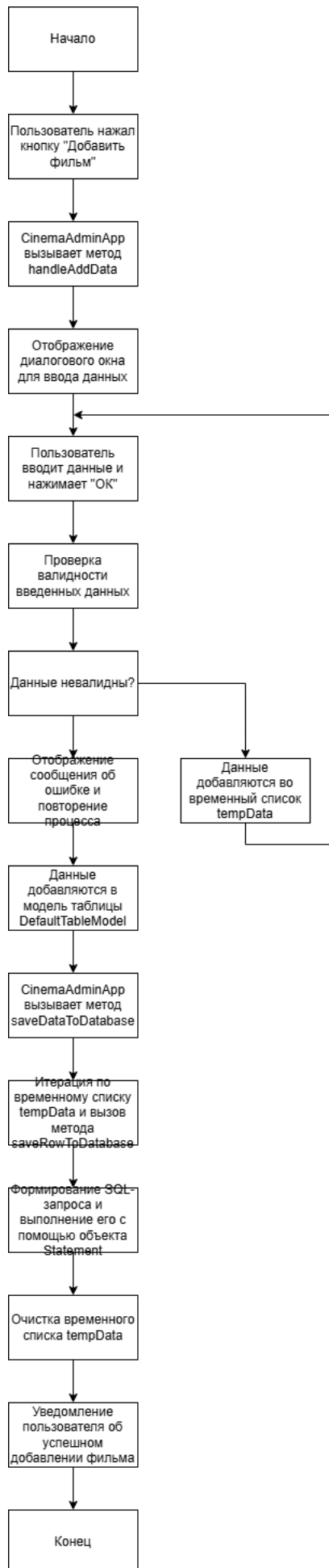
- Пользователь получает сообщение об успешном завершении операции.

Эта диаграмма последовательностей показывает, как данные добавляются и обновляются в базе данных через взаимодействие различных компонентов системы.

Построение диаграммы действий

Диаграмма действий (activity diagram) строится для сложных операций. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются действия, а дугами — переходы от одного действия к другому. Она очень похожа на блок-схемы алгоритмов. Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояние. Диаграмму деятельности принято строить таким образом, чтобы действия следовали сверху вниз. Отличительной чертой диаграммы действий является то, что в ней можно отобразить параллельные процессы. Для этой цели используется специальный символ (линия синхронизации), который позволяет задать разделение и слияние потоков управления. При этом разделение имеет один входящий переход и несколько выходящих, а слияние, наоборот, имеет несколько входящих переходов и один выходящий.

В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо иницируют выполнение действий, либо определяют некоторый результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Диаграмма действий изображена на рисунке 8



РУКОВОДСТВО ОПЕРАТОРА

Назначение программы

Программный комплекс (ПК) "Учет и администрирование репертуара кинотеатра" предназначен для автоматизации управления сведениями о фильмах, сеансах и билетах в рамках деятельности администратора кинотеатра. Программа входит в состав автоматизированной системы учета и администрирования информации, обеспечивая учет репертуара и продаж.

В рамках ПК "Учет и администрирование репертуара кинотеатра" администратор может:

- добавлять, изменять и удалять информацию о фильмах, доступных в репертуаре;
- добавлять, изменять и удалять сведения о сеансах и их расписании;
- управлять информацией о проданных билетах;
- формировать справочную информацию о текущем репертуаре и статистике продаж.

Условия выполнения программы

Программа предназначена для работы под операционной системой Windows, с использованием SQLite в качестве базы данных. Для корректного функционирования требуется:

1. Персональная электронно-вычислительная машина (ПЭВМ) с операционной системой Windows;
2. Минимальные характеристики системы:
 - процессор Intel Core i3 или аналогичный;
 - объем оперативной памяти — не менее 4 ГБ;
 - объем свободного дискового пространства — не менее 500 МБ;
 - стандартная клавиатура и манипулятор типа "мышь".

Описание задачи

Программный комплекс должен обеспечивать хранение сведений о фильмах, расписании сеансов и проданных билетах. Администратор кинотеатра может управлять этими данными и запрашивать следующую информацию:

- доступность фильма для показа;
- расписание сеансов для выбранного фильма;
- количество проданных билетов и оставшихся мест;
- статистику посещаемости по сеансам.

Обязательными требованиями при разработке кода ПК являются использование следующих конструкций языка Java:

- закрытые и открытые члены классов;
- наследование;
- конструкторы с параметрами;
- абстрактные базовые классы;

- виртуальные функции;
- обработка исключительных ситуаций;
- динамическое создание объектов.

Для реализации поставленной задачи была разработана общая модель программного комплекса с выявлением основных объектов и их взаимосвязей. В ходе проектирования были созданы программные классы для работы с информацией о фильмах и сеансах, а также их связи через базу данных

Входные и выходные данные

Входные данные:

- Информация о фильмах, сеансах и продажах вводится администратором через графический интерфейс программы. Ввод данных осуществляется в интерактивном режиме.

Выходные данные:

- Таблицы, отображающие список фильмов с характеристиками, расписание сеансов и статистику проданных билетов.

Выполнение программы

ЗАКЛЮЧЕНИЕ

Репозиторий

Исходный код курсовой работы размещен в публичном репозитории на GitHub и доступен по следующе - <https://github.com/brick1ng5654/course-2/tree/main/OOP/coursework>. В репозитории содержится полный исходный код проекта, а также документация, которая может быть полезна для дальнейшего изучения и эксплуатации программного комплекса.

Сгенерированная документация Javadoc, содержащая подробное описание классов, методов и их взаимодействий, находится в папке doc внутри репозитория. Она предоставляет структурированное представление о внутреннем устройстве приложения и может быть использована как справочное руководство для разработчиков.

Результат логирования

Лог-файл, содержащий около 1,5 тысячи строк, оказался крайне компактным, занимая менее 100 КБ. Такой небольшой объем памяти при хранении значительного количества информации стал возможным благодаря рациональному подходу к структуре и формату записей. Логирование сыграло ключевую роль в процессе отладки и разработки проекта. Оно позволило оперативно выявлять и анализировать ошибки, отслеживать последовательность выполнения операций и поведение системы в различных сценариях. Благодаря этому удалось не только ускорить процесс устранения багов, но и улучшить качество реализации функционала, обеспечив стабильность и надежность итогового решения.

Идеи и поддержка приложения

В рамках дальнейшего развития и улучшения приложения, предусмотрено внедрение ряда важных изменений и нововведений, направленных на повышение удобства использования и функциональности.

1. Система учётных записей: планируется интеграция системы регистрации и управления учётными записями пользователей. Это позволит каждому пользователю иметь индивидуальные настройки, обеспечит безопасность данных и позволит внедрить различные уровни доступа в зависимости от роли (администратор, пользователь и т. д.).
2. Улучшение пользовательского опыта: Важной частью дальнейшей работы над приложением является улучшение пользовательского интерфейса. Цель — сделать его более интуитивно понятным и комфортным для взаимодействия. В рамках этой работы будут учтены современные тренды дизайна, а также проведены исследования предпочтений целевой аудитории.
3. Упрощение структуры данных: Одной из задач является реорганизация структуры данных, чтобы она стала более логичной и гибкой, а также легко редактируемой. Это поможет улучшить производительность приложения, упростить поддержку и расширение функционала в будущем.
4. Интеграция детальной справочной информации: В приложение будет добавлена система справки, которая будет доступна прямо в интерфейсе приложения. Пользователи смогут получить всю необходимую информацию по вопросам, возникающим в процессе работы с программой, без необходимости обращаться к внешним источникам.
5. Обновление визуального стиля: В целях повышения визуальной привлекательности и соответствия современным стандартам, планируется обновление визуального стиля приложения. Это включает в себя переработку цветовой схемы, использование современных шрифтов, улучшение элементов управления и других визуальных аспектов интерфейса.
6. Исправление багов: в ходе тестирования приложения были выявлены различные ошибки и недочеты, которые будут устранены в будущих версиях. Также будет организована система отслеживания и исправления новых багов, чтобы минимизировать количество проблем, с которыми сталкиваются пользователи.

Эти улучшения позволят значительно повысить качество и удобство использования приложения, обеспечат его гибкость, безопасность и готовность к дальнейшему расширению

Итоги

В результате выполнения курсовой работы был разработан программный комплекс (ПК) «Учет и администрирование репертуара кинотеатра», предназначенный для управления сведениями о фильмах, расписании сеансов

и проданных билетах. Программа успешно автоматизирует процессы, связанные с учетом репертуара и продаж, упрощает администрирование и повышает оперативность получения данных.

В ходе выполнения курсового проекта была проделана следующая работа:

1. Проведен анализ требований и разработана спецификация функциональности программного комплекса.
2. Построена модель предметной области, выявлены основные сущности и их взаимосвязи.
3. Созданы описание вариантов использования ПК, включая сценарии добавления, изменения и удаления информации о фильмах, сеансах и билетах.
4. Разработан прототип графического интерфейса пользователя (GUI), ориентированный на удобство и интуитивность использования.
5. Построена объектная модель ПК и диаграмма классов, описывающая структуру программных компонентов и их взаимодействие.
6. Составлено описание поведения ПК и диаграммы действий, иллюстрирующие основные рабочие процессы программы.
7. Реализован программный код с учетом обязательных требований, включая использование наследования, абстрактных классов, обработки исключений и динамического создания объектов.
8. Проведено тестирование программного комплекса, выявлены и исправлены ошибки.
9. Разработано руководство оператора, описывающее порядок установки, настройки программного комплекса.

Курсовой проект полностью удовлетворяет поставленным требованиям и обеспечивает автоматизацию ключевых задач администратора кинотеатра. Программный комплекс разработан с использованием современных подходов к объектно-ориентированному программированию на языке Java, что гарантирует его надежность, расширяемость и удобство в эксплуатации.

ИСХОДНЫЕ ТЕКСТЫ ПК

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusAdapter;
import java.awt.event.FocusEvent;
import java.io.File;
import java.io.IOException;
import java.sql.*;
import java.util.logging.*;
import java.util.ArrayList;
import java.util.List;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

```

public class CinemaAdminApp {
    private static Connection dbConnection;
    private static DefaultTableModel filmsModel;
    private static DefaultTableModel repertoireModel;
    private static DefaultTableModel sessionsModel;
    private static DefaultTableModel ticketsModel;
    private static JTable currentTable;
    private static JTable filmsTable;
    private static JTable repertoireTable;
    private static JTable sessionsTable;
    private static JTable ticketsTable;

    // Хранит данные для каждой вкладки
    private static Map<String, List<String[]>> tempData = new HashMap<>();

    // Компоненты поиска
    private static JComboBox<String> director;
    private static JTextField wordToFind;

    // Флаг для редактирования
    private static boolean isEditingEnabled = false;

    private static final Logger logger =
        Logger.getLogger(CinemaAdminApp.class.getName());

    static {
        try {
            FileHandler fileHandler = new FileHandler("src/log.txt", true);
            fileHandler.setFormatter(new SimpleFormatter());
            logger.addHandler(fileHandler);
            logger.setLevel(Level.ALL);
        } catch (IOException e) {
            System.err.println("Не удалось настроить логирование: " +
                e.getMessage());
        }
    }

    public static void main(String[] args) {
        JButton filter;
        JButton filterClear;

        // Проверка авторизации
        logger.info("Приложение запущено");
        if (!showLoginDialog()) {
            logger.info("Приложение закрыто");
            System.exit(0);
        }

        // Создаем главное окно приложения
        JFrame frame = new JFrame("Cinema Admin App");
        frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        frame.setSize(800, 600);

        // Обработчик закрытия окна
        frame.addWindowListener(new java.awt.event.WindowAdapter() {
            @Override
            public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                handleExit(frame);
            }
        });

        // Создаем панель инструментов

```

```

JToolBar toolBar = new JToolBar();

JButton saveButton = new JButton(new ImageIcon("src/img/save.png"));
saveButton.setToolTipText("Сохранить список фильмов");
saveButton.addActionListener(e -> saveDataToDatabase(frame));

JButton addButton = new JButton(new ImageIcon("src/img/add.png"));
addButton.setToolTipText("Добавить фильм");
addButton.addActionListener(e -> {
    JTabbedPane tabbedPane = (JTabbedPane)
frame.getContentPane().getComponent(1);
    int selectedIndex = tabbedPane.getSelectedIndex();
    handleAddData(selectedIndex, frame);
});

JButton deleteButton = new JButton(new
ImageIcon("src/img/delete.png"));
deleteButton.setToolTipText("Удалить фильм");
deleteButton.addActionListener(e -> {
    handleDeleteData(frame);
});

JButton editButton = new JButton(new ImageIcon("src/img/edit.png"));
editButton.setToolTipText("Редактировать фильм");
editButton.addActionListener(e -> {
    isEditingEnabled = !isEditingEnabled;
    updateTableEditing(isEditingEnabled);
});

JButton loadButton = new JButton(new
ImageIcon("src/img/upload.png"));
loadButton.setToolTipText("Загрузить список фильмов");
loadButton.addActionListener(e -> loadDatabaseFile(frame));

JButton logsButton = new JButton(new ImageIcon("src/img/log.png"));
logsButton.setToolTipText("Открыть логи");

JButton exitButton = new JButton(new ImageIcon("src/img/exit.png"));
exitButton.setToolTipText("Выйти из приложения");

// Добавляем кнопки на панель инструментов
toolBar.add(saveButton);
toolBar.add(loadButton);
toolBar.addSeparator();
toolBar.add(addButton);
toolBar.add(deleteButton);
toolBar.add(editButton);
toolBar.addSeparator();
toolBar.add(logsButton);
toolBar.add(exitButton);

// Добавляем обработчик для кнопки "Логи"
logsButton.addActionListener(e -> {
    File logFile = new File("src/log.txt");
    if (logFile.exists()) {
        try {
            Desktop.getDesktop().open(logFile);
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(frame, "Ошибка открытия
log.txt: " + ex.getMessage(),
                "Ошибка", JOptionPane.ERROR_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(frame, "Файл log.txt не

```

```

найден.",
                                "Ошибка", JOptionPane.ERROR_MESSAGE);
        }
    });

    // Добавляем обработчик для кнопки "Выход"
    exitButton.addActionListener(e -> handleExit(frame));

    // Создаем панель вкладок
    JTabbedPane tabbedPane = new JTabbedPane();

    // Вкладка "Фильмы"
    JPanel filmsPanel = new JPanel(new BorderLayout());
    filmsTable = new JTable();
    filmsModel = new DefaultTableModel() {
        @Override
        public boolean isCellEditable(int row, int column) {
            return column != 0 && isEditingEnabled;
        }
    };
    filmsTable.setModel(filmsModel);
    filmsPanel.add(new JScrollPane(filmsTable), BorderLayout.CENTER);
    tabbedPane.addTab("Фильмы", filmsPanel);

    // Вкладка "Репертуар"
    JPanel repertoirePanel = new JPanel(new BorderLayout());
    repertoireTable = new JTable();
    repertoireModel = new DefaultTableModel() {
        @Override
        public boolean isCellEditable(int row, int column) {
            return column != 0 && isEditingEnabled;
        }
    };
    repertoireTable.setModel(repertoireModel);
    repertoirePanel.add(new JScrollPane(repertoireTable),
BorderLayout.CENTER);
    tabbedPane.addTab("Репертуар", repertoirePanel);

    // Вкладка "Сеансы"
    JPanel sessionsPanel = new JPanel(new BorderLayout());
    sessionsTable = new JTable();
    sessionsModel = new DefaultTableModel() {
        @Override
        public boolean isCellEditable(int row, int column) {
            return column != 0 && isEditingEnabled;
        }
    };
    sessionsTable.setModel(sessionsModel);
    sessionsPanel.add(new JScrollPane(sessionsTable),
BorderLayout.CENTER);
    tabbedPane.addTab("Сеансы", sessionsPanel);

    // Вкладка "Билеты"
    JPanel ticketsPanel = new JPanel(new BorderLayout());
    ticketsTable = new JTable();
    ticketsModel = new DefaultTableModel() {
        @Override
        public boolean isCellEditable(int row, int column) {
            return column != 0 && isEditingEnabled;
        }
    };
    ticketsTable.setModel(ticketsModel);
    ticketsPanel.add(new JScrollPane(ticketsTable), BorderLayout.CENTER);
    tabbedPane.addTab("Билеты", ticketsPanel);

```

```

// Подготовка компонентов поиска
director = new JComboBox<>();

wordToFind = new JTextField("Ключевое слово", 20);
wordToFind.addFocusListener(new FocusAdapter() {
    public void focusGained(FocusEvent e) {
        if (wordToFind.getText().equals("Ключевое слово")) {
            wordToFind.setText(""); // Очистить поле при получении
фокуса
        }
    }
    public void focusLost(FocusEvent e) {
        if (wordToFind.getText().isEmpty()) {
            wordToFind.setText("Ключевое слово"); // Вернуть текст,
если поле пустое
        }
    }
});

filterClear = new JButton("Сброс");
filterClear.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        loadDataFromDatabase("movies", filmsTable);
        loadDataFromDatabase("rep", repertoireTable);
        loadDataFromDatabase("sessions", sessionsTable);
        loadDataFromDatabase("tickets", ticketsTable);
        wordToFind.setText("Ключевое слово");
    }
});

filter = new JButton("Поиск");
filter.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            checkName(wordToFind);
            performSearch(frame);
        } catch (NullPointerException ex) {
            JOptionPane.showMessageDialog(frame, ex.toString(),
"Ошибка", JOptionPane.ERROR_MESSAGE);
        } catch (InvalidFindException myEx) {
            JOptionPane.showMessageDialog(frame, myEx.getMessage(),
"Ошибка", JOptionPane.ERROR_MESSAGE);
        }
    }
});

// Панель поиска
JPanel searchPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
searchPanel.add(new JLabel("Поиск по:"));
searchPanel.add(director);
searchPanel.add(wordToFind);
searchPanel.add(filter);
searchPanel.add(filterClear);

// Устанавливаем расположение элементов в главном окне
frame.setLayout(new BorderLayout());
frame.add(toolBar, BorderLayout.NORTH);
frame.add(tabbedPane, BorderLayout.CENTER);
frame.add(searchPanel, BorderLayout.SOUTH);

// Подключение к базе данных
connectToDatabase();

```

```

        // Загрузка данных из базы в таблицы
        loadDataFromDatabase("movies", filmsTable);
        loadDataFromDatabase("rep", repertoireTable);
        loadDataFromDatabase("sessions", sessionsTable);
        loadDataFromDatabase("tickets", ticketsTable);

        // Обновление полей поиска при смене вкладки
        tabbedPane.addChangeListener(e ->
updateSearchFields(tabbedPane.getSelectedIndex()));

        // Отображаем окно
        frame.setVisible(true);
    }

    /**
     * Обработчик закрытия приложения.
     * @param frame Главное окно приложения.
     */
    private static void handleExit(JFrame frame) {
        int choice = JOptionPane.showConfirmDialog(frame,
            "Вы уверены, что хотите выйти из приложения?",
            "Подтверждение",
            JOptionPane.YES_NO_OPTION);
        if (choice == JOptionPane.YES_OPTION) {
            logger.info("Приложение закрыто");
            if (dbConnection != null) {
                try {
                    dbConnection.close();
                } catch (SQLException e) {
                    logger.warning("Ошибка при закрытии соединения с базой
данных: " + e.getMessage());
                }
            }
            System.exit(0);
        }
    }

    /**
     * Загрузка базы данных из файла.
     * @param frame Главное окно приложения.
     */
    private static void loadDatabaseFile(JFrame frame) {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileFilter(new
javax.swing.filechooser.FileNameExtensionFilter("SQLite Database Files",
"db"));
        int returnValue = fileChooser.showOpenDialog(frame);

        if (returnValue == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            String dbUrl = "jdbc:sqlite:" + selectedFile.getAbsolutePath();

            try {
                if (dbConnection != null && !dbConnection.isClosed()) {
                    dbConnection.close();
                }
                dbConnection = DriverManager.getConnection(dbUrl);
                JOptionPane.showMessageDialog(frame, "База данных успешно
загружена.", "Успех", JOptionPane.INFORMATION_MESSAGE);
                logger.info("База данных успешно загружена из файла: " +
dbUrl);

                // Перезагрузка данных в таблицы

```

```

        loadDataFromDatabase("movies", filmsTable);
        loadDataFromDatabase("rep", repertoireTable);
        loadDataFromDatabase("sessions", sessionsTable);
        loadDataFromDatabase("tickets", ticketsTable);
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(frame, "Ошибка подключения к
базе данных: " + e.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
        logger.severe("Ошибка подключения к базе данных: " +
e.getMessage());
    }
}

/**
 * Отображение диалога авторизации.
 * @return true, если авторизация успешна, иначе false.
 */
private static boolean showLoginDialog() {
    JPanel panel = new JPanel(new GridLayout(3, 2));
    JLabel userLabel = new JLabel("Логин:");
    JTextField userField = new JTextField();
    JLabel passLabel = new JLabel("Пароль:");
    JPasswordField passField = new JPasswordField();

    panel.add(userLabel);
    panel.add(userField);
    panel.add(passLabel);
    panel.add(passField);

    JOptionPane optionPane = new JOptionPane(panel,
JOptionPane.PLAIN_MESSAGE, JOptionPane.OK_CANCEL_OPTION);
    JDialog dialog = optionPane.createDialog("Вход в систему");
    dialog.setModalityType(Dialog.ModalityType.APPLICATION_MODAL);
    dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

    while (true) {
        dialog.setVisible(true);

        int result = (Integer) optionPane.getValue();
        if (result == JOptionPane.OK_OPTION) {
            String username = userField.getText();
            String password = new String(passField.getPassword());
            if ("admin".equals(username) && "admin".equals(password)) {
                logger.info("Авторизация успешно пройдена");
                return true;
            } else {
                JOptionPane.showMessageDialog(dialog, "Неправильный логин
или пароль.",
                    "Ошибка", JOptionPane.ERROR_MESSAGE);
                logger.info("Логин или пароль неправильно введены");
                // Очистка полей логина и пароля
                userField.setText("");
                passField.setText("");
            }
        } else {
            return false;
        }
    }
}

/**
 * Подключение к базе данных.
 */
private static void connectToDatabase() {

```



```

        String url = "jdbc:sqlite:cinema.db";
        try {
            dbConnection = DriverManager.getConnection(url);
            logger.info("Соединение с базой данных успешно установлено");
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Ошибка подключения к базе
данных: " + e.getMessage(),
                "Ошибка", JOptionPane.ERROR_MESSAGE);
            logger.severe("Ошибка подключения к базе данных: " +
e.getMessage());
        }
    }

    /**
     * Загрузка данных из базы данных в таблицу.
     * @param tableName Имя таблицы в базе данных.
     * @param table Таблица для отображения данных.
     */
    private static void loadDataFromDatabase(String tableName, JTable table)
    {
        try (Statement stmt = dbConnection.createStatement()) {
            ResultSet rs = stmt.executeQuery("SELECT * FROM " + tableName);
            ResultSetMetaData metaData = rs.getMetaData();

            // Установка модели таблицы с динамическими колонками
            DefaultTableModel model = (DefaultTableModel) table.getModel();
            model.setColumnCount(0);
            model.setRowCount(0);
            int columnCount = metaData.getColumnCount();
            for (int i = 1; i <= columnCount; i++) {
                model.addColumn(metaData.getColumnName(i));
            }

            while (rs.next()) {
                Object[] rowData = new Object[columnCount];
                for (int i = 0; i < columnCount; i++) {
                    rowData[i] = rs.getObject(i + 1);
                }
                model.addRow(rowData);
            }
            logger.info("Данные из таблицы " + tableName + " успешно
загружены");
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Ошибка загрузки данных из
таблицы " + tableName + ": " + e.getMessage(),
                "Ошибка", JOptionPane.ERROR_MESSAGE);
            logger.warning("Ошибка загрузки данных из таблицы " + tableName +
": " + e.getMessage());
        }
    }

    /**
     * Получение следующего ID для новой записи в таблице.
     * @param tableName Имя таблицы.
     * @return Следующий ID.
     * @throws SQLException Если происходит ошибка при выполнении SQL-
запроса.
     */
    private static int getNextId(String tableName) throws SQLException {
        String idColumn = getIdColumnName(tableName);
        if (idColumn != null) {
            try (Statement stmt = dbConnection.createStatement();
                ResultSet rs = stmt.executeQuery("SELECT MAX(" + idColumn +
") FROM " + tableName)) {

```

```

        if (rs.next()) {
            return rs.getInt(1) + 1;
        }
    }
    return 1; // Если таблица пуста или столбец не найден, начинаем с 1
}

/**
 * Получение имени столбца ID для таблицы.
 * @param tableName Имя таблицы.
 * @return Имя столбца ID.
 * @throws SQLException Если происходит ошибка при выполнении SQL-
запроса.
 */
private static String getIdColumnName(String tableName) throws
SQLException {
    try (Statement stmt = dbConnection.createStatement();
        ResultSet rs = stmt.executeQuery("PRAGMA table_info(" +
tableName + ")")) {
        while (rs.next()) {
            String columnName = rs.getString("name");
            if (columnName.matches(".*_id")) {
                return columnName;
            }
        }
    }
    return null;
}

/**
 * Извлечение числа из строки в скобках.
 * @param input Входная строка.
 * @return Число в скобках или null, если число не найдено.
 */
private static String extractNumberFromString(String input) {
    Pattern pattern = Pattern.compile("\\((\\d+)\\)"); // Регулярное
выражение для числа в скобках
    Matcher matcher = pattern.matcher(input);
    if (matcher.find()) {
        return matcher.group(1); // Возвращаем только число
    }
    return null; // Если числа в скобках нет, возвращаем null
}

/**
 * Обработчик для кнопки "Добавить данные".
 * @param tabIndex Индекс текущей вкладки.
 * @param frame Главное окно приложения.
 */
private static void handleAddData(int tabIndex, JFrame frame) {
    String tableName = getTableNameByTabIndex(tabIndex);
    if (tableName == null) {
        JOptionPane.showMessageDialog(null, "Неизвестная вкладка.",
"Ошибка", JOptionPane.ERROR_MESSAGE);
        return;
    }

    JPanel inputPanel = new JPanel(new GridLayout(0, 2));
    try (Statement stmt = dbConnection.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM " + tableName +
" LIMIT 1")) {
        ResultSetMetaData metaData = rs.getMetaData();
        int columnCount = metaData.getColumnCount();

```

```

        JTextField[] inputFields = new JTextField[columnCount - 1]; //
Исключаем поле id

        for (int i = 2; i <= columnCount; i++) {
            String columnName = metaData.getColumnNames(i);
            inputPanel.add(new JLabel(getUserFriendlyName(columnName)));

            if (columnName.equalsIgnoreCase("rep_id") ||
columnName.equalsIgnoreCase("movie_id") ||
columnName.equalsIgnoreCase("session_id")) {
                JComboBox<String> comboBox = new JComboBox<>();
                loadComboBoxData(comboBox, columnName);
                inputPanel.add(comboBox);
            } else {
                inputFields[i - 2] = new JTextField();
                inputPanel.add(inputFields[i - 2]);
            }
        }

        JOptionPane optionPane = new JOptionPane(inputPanel,
JOptionPane.PLAIN_MESSAGE, JOptionPane.OK_CANCEL_OPTION);
        JDialog dialog = optionPane.createDialog("Добавить запись в " +
tableName);
        dialog.setModalityType(Dialog.ModalityType.APPLICATION_MODAL);
        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

        while (true) {
            dialog.setVisible(true);

            int result = (Integer) optionPane.getValue();
            if (result == JOptionPane.OK_OPTION) {
                String[] rowData = new String[columnCount];
                int nextId = getNextId(tableName);
                rowData[0] = String.valueOf(nextId);

                for (int i = 1; i < columnCount; i++) {
                    Component fieldComponent = inputPanel.getComponent(i
* 2 - 1); // Поле ввода/выбор

                    if (fieldComponent instanceof JTextField) {
                        JTextField textField = (JTextField)
fieldComponent;

                        rowData[i] = textField.getText(); // Получаем
текст из поля ввода

                    } else if (fieldComponent instanceof JComboBox) {
                        JComboBox<?> comboBox = (JComboBox<?>)
fieldComponent;

                        try {
                            rowData[i] = comboBox.getSelectedItem() !=
null ? comboBox.getSelectedItem().toString() : null; // Получаем выбранное
значение из JComboBox

                        } catch (Exception e) {
                            System.out.println("Ошибка при получении
значения из JComboBox: " + e.getMessage());
                        }
                    }
                }

                // Проверка валидности данных
                try {
                    validateData(tableName, rowData);
                    if (!tempData.containsKey(tableName)) {
                        tempData.put(tableName, new ArrayList<>());
                    }
                }
            }
        }
    }
}

```

```

        tempData.get(tableName).add(rowData);

        // Отладочные сообщения
        StringBuilder debugMessage = new
StringBuilder("Добавление строки в модель таблицы " + tableName + ": ");
        for (String data : rowData) {
            debugMessage.append(data).append(", ");
        }
        logger.info(debugMessage.toString());

        DefaultTableModel model =
getTableModelByTabIndex(tabIndex);
        model.addRow(rowData);
        logger.info("Запись успешно добавлена в интерфейс
таблицы " + tableName);
        saveDataToDatabase(frame);

        break; // Выход из цикла, если данные корректны
    } catch (Exception e) {
        JOptionPane.showMessageDialog(dialog, e.getMessage(),
"Ошибка", JOptionPane.ERROR_MESSAGE);
    }
    } else {
        break; // Выход из цикла, если пользователь отменил ввод
    }
}
} catch (SQLException e) {
    logger.warning("Ошибка добавления записи в интерфейс таблицы " +
tableName + ": " + e.getMessage());
    JOptionPane.showMessageDialog(null, "Ошибка добавления записи: "
+ e.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
}
}

/**
 * Сохранение данных в базу данных.
 * @param frame Главное окно приложения.
 */
private static void saveDataToDatabase(JFrame frame) {
    try (Statement stmt = dbConnection.createStatement()) {
        for (Map.Entry<String, List<String[]>> entry :
tempData.entrySet()) {
            String tableName = entry.getKey();
            for (String[] values : entry.getValue()) {
                System.out.println(values);
                saveRowToDatabase(tableName, values, stmt);
            }
        }
        tempData.clear(); // Очищаем временный список после сохранения
        saveEditedRowsToDatabase(frame);
        logger.info("Данные успешно сохранены в базу данных");
        JOptionPane.showMessageDialog(frame, "Данные успешно сохранены!",
"Успех", JOptionPane.INFORMATION_MESSAGE);
    } catch (SQLException e) {
        logger.warning("Ошибка сохранения данных в базу данных: " +
e.getMessage());
        JOptionPane.showMessageDialog(frame, "Ошибка сохранения данных: "
+ e.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Сохранение строки в базу данных.
 * @param tableName Имя таблицы.

```

```

    * @param values Значения для сохранения.
    * @param stmt Statement для выполнения SQL-запросов.
    * @throws SQLException Если происходит ошибка при выполнении SQL-
запроса.
    */
    private static void saveRowToDatabase(String tableName, String[] values,
Statement stmt) throws SQLException {
        System.out.println(values);
        StringBuilder query = new StringBuilder("INSERT INTO " + tableName +
" (");
        ResultSet rs = stmt.executeQuery("SELECT * FROM " + tableName + "
LIMIT 1");
        ResultSetMetaData metaData = rs.getMetaData();
        int columnCount = metaData.getColumnCount();

        // Формируем список столбцов, исключая автоинкрементируемое поле
        for (int i = 2; i <= columnCount; i++) { // Начинаем с 1, чтобы
включить все столбцы
            String columnName = metaData.getColumnName(i);
            query.append(columnName);
            if (i < columnCount) query.append(", ");
        }
        query.append(") VALUES (");
        for (int i = 1; i < values.length; i++) {
            System.out.println(values[i]);
            query.append("'").append(values[i]).append("'");
            if (i < values.length - 1) query.append(", ");
        }
        query.append(")");
        System.out.println(query.toString()); // Для отладки
        stmt.executeUpdate(query.toString());
    }

    /**
    * Обработчик для кнопки "Удалить данные".
    * @param frame Главное окно приложения.
    */
    private static void handleDeleteData(JFrame frame) {
        JTabbedPane tabbedPane = (JTabbedPane)
frame.getContentPane().getComponent(1);
        int selectedIndex = tabbedPane.getSelectedIndex();
        Component component = tabbedPane.getComponentAt(selectedIndex);
        if (component instanceof JPanel) {
            JPanel panel = (JPanel) component;
            JScrollPane scrollPane = (JScrollPane) panel.getComponent(0); //
Предполагаем, что JScrollPane первый компонент в JPanel
            JTable selectedTable = (JTable)
scrollPane.getViewPort().getView();
            String tableName = getTableNameByTabIndex(selectedIndex);

            // Устанавливаем текущую таблицу
            setCurrentTable(selectedTable, tableName);

            if (currentTable == null || currentTable.getSelectedRow() == -1)
{
                JOptionPane.showMessageDialog(null, "Выберите строку для
удаления.", "Ошибка", JOptionPane.ERROR_MESSAGE);
                return;
            }

            int selectedRow = currentTable.getSelectedRow();
            String idColumn = null;
            switch (tableName) {
                case "movies":

```

```

        idColumn = "movie_id";
        break;
    case "rep":
        idColumn = "rep_id";
        break;
    case "sessions":
        idColumn = "session_id";
        break;
    case "tickets":
        idColumn = "ticket_id";
        break;
    }

    String id = currentTable.getValueAt(selectedRow, 0).toString();
    // Предполагается, что ID в первом столбце

    int confirm = JOptionPane.showConfirmDialog(null,
        "Вы уверены, что хотите удалить запись с ID: " + id +
        "?",
        "Подтверждение удаления", JOptionPane.YES_NO_OPTION);

    if (confirm != JOptionPane.YES_OPTION) {
        return;
    }

    try (Statement stmt = dbConnection.createStatement()) {
        int rowsAffected = stmt.executeUpdate("DELETE FROM " +
        tableName + " WHERE " + idColumn + " = " + id);

        if (rowsAffected > 0) {
            JOptionPane.showMessageDialog(null, "Запись успешно
            удалена.", "Успех", JOptionPane.INFORMATION_MESSAGE);
            ((DefaultTableModel)
            currentTable.getModel()).removeRow(selectedRow); // Удаляем строку из таблицы
            logger.info("Запись с ID " + id + " удалена из таблицы "
            + tableName);
        } else {
            JOptionPane.showMessageDialog(null, "Запись с указанным
            ID не найдена.", "Ошибка", JOptionPane.ERROR_MESSAGE);
            logger.warning("Запись с ID " + id + " не найдена в
            таблице " + tableName);
        }
    } catch (SQLException e) {
        logger.warning("Ошибка удаления записи из таблицы " +
        tableName + ": " + e.getMessage());
        JOptionPane.showMessageDialog(null, "Ошибка удаления записи:
        " + e.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
    }
    } else {
        JOptionPane.showMessageDialog(frame, "Неверный компонент
        вкладки.", "Ошибка", JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Вспомогательный метод для обновления текущей таблицы.
 * @param table Таблица для обновления.
 * @param tableName Имя таблицы.
 */
public static void setCurrentTable(JTable table, String tableName) {
    currentTable = table;
    currentTable.putClientProperty("tableName", tableName);
}

```

```

/**
 * Получение имени таблицы по индексу вкладки.
 * @param tabIndex Индекс вкладки.
 * @return Имя таблицы или null, если вкладка неизвестна.
 */
private static String getTableNameByTabIndex(int tabIndex) {
    switch (tabIndex) {
        case 0: return "movies";
        case 1: return "rep";
        case 2: return "sessions";
        case 3: return "tickets";
        default: return null;
    }
}

/**
 * Получение модели таблицы по индексу вкладки.
 * @param tabIndex Индекс вкладки.
 * @return Модель таблицы или null, если вкладка неизвестна.
 */
private static DefaultTableModel getTableModelByTabIndex(int tabIndex) {
    switch (tabIndex) {
        case 0: return filmsModel;
        case 1: return repertoireModel;
        case 2: return sessionsModel;
        case 3: return ticketsModel;
        default: return null;
    }
}

/**
 * Получение пользовательского имени столбца.
 * @param columnName Имя столбца.
 * @return Пользовательское имя столбца.
 */
private static String getUserFriendlyName(String columnName) {
    switch (columnName.toLowerCase()) {
        case "movie_name": return "Название фильма";
        case "director": return "Режиссер";
        case "movie_year": return "Год выпуска";
        case "genre": return "Жанр";
        case "date": return "Дата";
        case "time": return "Время";
        case "host": return "Зал";
        case "movie_id": return "ID фильма";
        case "rep_id": return "ID репертуара";
        case "numb_seats": return "Количество мест";
        case "numb_sold_tickets": return "Количество проданных билетов";
        case "ticket_id": return "ID билета";
        case "session_id": return "ID сеанса";
        case "numb_of_seat": return "Номер места";
        case "status": return "Статус";
        case "date_bought": return "Дата покупки";
        default: return columnName;
    }
}

// Исключения для проверки валидности данных
private static class InvalidFindException extends Exception {
    public InvalidFindException(String message) {
        super(message);
    }
}

```

```

private static class InvalidDateException extends Exception {
    public InvalidDateException(String message) {
        super(message);
    }
}

private static class InvalidTimeException extends Exception {
    public InvalidTimeException(String message) {
        super(message);
    }
}

private static class InvalidTicketCountException extends Exception {
    public InvalidTicketCountException(String message) {
        super(message);
    }
}

// Методы проверки валидности данных
private static void checkName(JTextField bName) throws
InvalidFindException, NullPointerException {
    String sName = bName.getText();
    if (sName.contains("Ключевое слово"))
        throw new InvalidFindException("Вы не ввели слова для поиска");
    if (sName.isEmpty()) throw new NullPointerException();
}

private static void checkDate(String date) throws InvalidDateException {
    if (!date.matches("\\d{2}\\d{2}\\d{4}")) {
        throw new InvalidDateException("Неверный формат даты! Ожидается
формат: дд.мм.гггг.");
    }
}

private static void checkTime(String time) throws InvalidTimeException {
    if (!time.matches("\\d{2}:\\d{2}")) {
        throw new InvalidTimeException("Неверный формат времени!
Ожидается формат: чч:мм.");
    }
}

private static void checkTicketCount(String ticketsSold) throws
InvalidTicketCountException, NumberFormatException {
    try {
        int count = Integer.parseInt(ticketsSold);
        if (count < 0) {
            throw new InvalidTicketCountException("Количество проданных
билетов не может быть отрицательным!");
        }
    } catch (NumberFormatException ex) {
        throw new InvalidTicketCountException("Количество проданных
билетов должно быть числом!");
    }
}

// Метод для проверки валидности данных перед добавлением
private static void validateData(String tableName, String[] rowData)
throws Exception {
    for (int i = 1; i < rowData.length; i++) {
        String columnName = getColumnIndexByIndex(tableName, i);
        String value = rowData[i];
        switch (columnName.toLowerCase()) {
            case "date":
                checkDate(value);

```



```

        break;
    case "time":
        checkTime(value);
        break;
    case "numb_sold_tickets":
        checkTicketCount(value);
        break;
    // Добавьте другие проверки по мере необходимости
}
}

// Вспомогательный метод для получения имени столбца по индексу
private static String getColumnByNameByIndex(String tableName, int index)
throws SQLException {
    try (Statement stmt = dbConnection.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM " + tableName +
" LIMIT 1")) {
        ResultSetMetaData metaData = rs.getMetaData();
        return metaData.getColumnName(index + 1); // Индексы столбцов
начинаются с 1
    }
}

// Метод для выполнения поиска
private static void performSearch(JFrame frame) {
    JTabbedPane tabbedPane = (JTabbedPane)
frame.getContentPane().getComponent(1);
    int selectedIndex = tabbedPane.getSelectedIndex();
    String tableName = getTableNameByTabIndex(selectedIndex);
    String searchField = director.getSelectedItemAt() != null ?
director.getSelectedItemAt().toString() : "";
    String searchKeyword = wordToFind.getText();

    if (tableName == null || searchField.isEmpty() ||
searchKeyword.isEmpty()) {
        JOptionPane.showMessageDialog(frame, "Выберите поле для поиска и
введите ключевое слово.", "Ошибка", JOptionPane.ERROR_MESSAGE);
        return;
    }

    try (Statement stmt = dbConnection.createStatement()) {
        String query = "SELECT * FROM " + tableName + " WHERE " +
searchField + " LIKE ?";
        PreparedStatement pstmt = dbConnection.prepareStatement(query);
        pstmt.setString(1, "%" + searchKeyword + "%");
        ResultSet rs = pstmt.executeQuery();

        DefaultTableModel model = getTableModelByTabIndex(selectedIndex);
        model.setRowCount(0); // Очищаем текущую таблицу

        ResultSetMetaData metaData = rs.getMetaData();
        int columnCount = metaData.getColumnCount();

        while (rs.next()) {
            Object[] rowData = new Object[columnCount];
            for (int i = 0; i < columnCount; i++) {
                rowData[i] = rs.getObject(i + 1);
            }
            model.addRow(rowData);
        }

        JOptionPane.showMessageDialog(frame, "Поиск завершен.", "Успех",
JOptionPane.INFORMATION_MESSAGE);
    }
}

```

```

    } catch (SQLException e) {
        JOptionPane.showMessageDialog(frame, "Ошибка выполнения поиска: "
+ e.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
        logger.warning("Ошибка выполнения поиска: " + e.getMessage());
    }
}

// Метод для обновления полей поиска при смене вкладки
private static void updateSearchFields(int selectedIndex) {
    String tableName = getTableNameByTabIndex(selectedIndex);
    if (tableName == null) {
        return;
    }

    try (Statement stmt = dbConnection.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM " + tableName +
" LIMIT 1")) {
        ResultSetMetaData metaData = rs.getMetaData();
        int columnCount = metaData.getColumnCount();
        String[] columnNames = new String[columnCount];

        for (int i = 1; i <= columnCount; i++) {
            columnNames[i - 1] = metaData.getColumnName(i);
        }

        director.setModel(new DefaultComboBoxModel<>(columnNames));
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ошибка обновления полей
поиска: " + e.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
        logger.warning("Ошибка обновления полей поиска: " +
e.getMessage());
    }
}

// Метод для обновления редактирования таблицы
private static void updateTableEditing(boolean isEditingEnabled) {
    filmsModel.fireTableDataChanged();
    repertoireModel.fireTableDataChanged();
    sessionsModel.fireTableDataChanged();
    ticketsModel.fireTableDataChanged();
}

// Метод для загрузки данных в раскрывающийся список
private static void loadComboBoxData(JComboBox<String> comboBox, String
columnName) {
    try (Statement stmt = dbConnection.createStatement()) {
        String query = "";
        if (columnName.equalsIgnoreCase("movie_id")) {
            query = "SELECT movie_name, movie_id FROM movies";
        } else if (columnName.equalsIgnoreCase("rep_id")) {
            query = "SELECT rep_id FROM rep";
        } else if (columnName.equalsIgnoreCase("session_id")) {
            query = "SELECT session_id FROM sessions";
        }
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            String item = columnName.equalsIgnoreCase("movie_id") ?
rs.getString("movie_name") + " (" +
rs.getString("movie_id") + ")" :
rs.getString(1);
            comboBox.addItem(item);
            System.out.println("Добавлен элемент в JComboBox: " + item);
        }
    }
}

```

```

    } catch (SQLException e) {
        logger.warning("Ошибка загрузки данных в раскрывающийся список: "
+ e.getMessage());
        JOptionPane.showMessageDialog(null, "Ошибка загрузки данных в
раскрывающийся список: " + e.getMessage(), "Ошибка",
JOptionPane.ERROR_MESSAGE);
    }
}

// Метод для сохранения измененных строк в базе данных
private static void saveEditedRowsToDatabase(JFrame frame) {
    try (Statement stmt = dbConnection.createStatement()) {
        JTabbedPane tabbedPane = (JTabbedPane)
frame.getContentPane().getComponent(1);
        int selectedIndex = tabbedPane.getSelectedIndex();
        String tableName = getTableNameByTabIndex(selectedIndex);
        DefaultTableModel model = getTableModelByTabIndex(selectedIndex);

        for (int row = 0; row < model.getRowCount(); row++) {
            String idColumn = null;
            switch (tableName) {
                case "movies":
                    idColumn = "movie_id";
                    break;
                case "rep":
                    idColumn = "rep_id";
                    break;
                case "sessions":
                    idColumn = "session_id";
                    break;
                case "tickets":
                    idColumn = "ticket_id";
                    break;
            }

            String id = model.getValueAt(row, 0).toString(); //
Предполагается, что ID в первом столбце
            StringBuilder query = new StringBuilder("UPDATE " + tableName
+ " SET ");

            boolean isRowChanged = false;
            StringBuilder setClause = new StringBuilder(); // Для
накопления изменений в строках

            for (int col = 1; col < model.getColumnCount(); col++) {
                String columnName = model.getColumnName(col);
                String value = model.getValueAt(row, col).toString();

                // Сравнение текущего значения с оригинальным
                if (!value.equals(getOriginalValue(tableName, id,
columnName))) {
                    isRowChanged = true;
                    if (setClause.length() > 0) {
                        setClause.append(", "); // Добавляем запятую
между парами "column = value"
                    }
                    setClause.append(columnName).append(" =
'").append(value).append("'");
                }
            }

            if (isRowChanged) {
                // Формируем финальный запрос
                query.append(setClause).append(" WHERE
").append(idColumn).append(" = ").append(id);
            }
        }
    }
}

```

```

        System.out.println(query.toString()); // Для отладки
        stmt.executeUpdate(query.toString());
    }
}

logger.info("Измененные строки успешно сохранены в базе данных");
} catch (SQLException e) {
    logger.warning("Ошибка сохранения измененных строк в базе данных:
" + e.getMessage());
    JOptionPane.showMessageDialog(frame, "Ошибка сохранения
измененных строк: " + e.getMessage(), "Ошибка", JOptionPane.ERROR_MESSAGE);
}
}

private static String getOriginalValue(String tableName, String id,
String columnName) throws SQLException {
    try (Statement stmt = dbConnection.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT " + columnName + " FROM
" + tableName + " WHERE " + getIdColumn(tableName) + " = " + id)) {
        if (rs.next()) {
            return rs.getString(columnName);
        }
    }
    return null;
}

private static String getIdColumn(String tableName) {
    switch (tableName) {
        case "movies":
            return "movie_id";
        case "rep":
            return "rep_id";
        case "sessions":
            return "session_id";
        case "tickets":
            return "ticket_id";
        default:
            return null;
    }
}
}
}

```