# ВВЕДЕНИЕ

Данная работа посвящена исследованиям, лежащим в области бухгалтерского учета, и касается автоматизации процесса учета аренды помещений в государственных и коммерческих учреждениях.

Актуальность темы заключается в том, что учет аренды помещений в настоящее время проводится повсеместно, а существующие инструменты для решения задач учета имеют множество недостатков. В различных отраслях экономики бухгалтерский учет имеет множество специфических особенностей. В зависимости от отрасли, в которой проводится учет, меняется и весь процесс работы бухгалтера.

По мере технологического развития человечества, процесс учета также претерпевал серьёзные изменения. Текущие возможности для учета кардинально отличаются от прежних способов ведения аренды, таких как запечатление и припоминание арендаторов в своей памяти или их фиксация на листах бумаги. В настоящее время доступен гораздо более удобный режим управления арендой – через файлы и документы на электронно-вычислительной машине. Но само использование компьютера не избавляет от ручного труда. Для решения проблем работников бухгалтерии, улучшения условий труда и повышения их производительности необходимо создать соответствующие инструменты, которые автоматизируют рабочий процесс.

Современный мир уже невозможно представить без учета аренды помещений. Вместе с развитием технологий и государства, увеличилось и количество сопутствующих сдаче в аренду зависимостей, таких как коммунальные услуги, тарификация, индексы стоимостей и налоги. Для всего вышеперечисленного также необходимо проводить расчеты и вести учет, так как это уже является неотъемлемой часть сдачи помещений в аренду.

Для реализации коммерческой деятельности компании, зачастую, необходимо помещение, которое может находиться в собственности или в аренде. Не все предприниматели имеют возможность приобрести себе помещение, поэтому наиболее распространенным считается второй вариант. Но аренда предусматривает возникновение постоянных затрат, связанных с её уплатой.

Без учета аренды многим компаниям было бы невозможно вести свою деятельность, поэтому необходимо вести правильный учет аренды.

Основной задачей бухгалтерского учета аренды является поддержка экономических процессов различных учреждений, компаний и предприятий.

В данной работе в качестве участка бухгалтерского учета, который будет подвержен автоматизации, будет выступать аренда помещений государственного учреждения.

Цель дипломной работы заключается в анализе предметной области бухгалтерского учета и разработке автоматизированного рабочего места бухгалтера для учета аренды помещений.

**Для достижения обозначенной цели выделены следующие задачи**:

1. Исследовать общие принципы ведения бухгалтерского учета аренды помещений и провести анализ продуктов-аналогов на рынке.
2. Сформулировать требования к разрабатываемому продукту.
3. Обосновать выбор инструментов разработки.
4. Спроектировать БД.
5. Написать код программы АРМ бухгалтера.

**Объект:** бухгалтерский учет аренды.

**Предмет:** автоматизированное рабочее место бухгалтера для учета аренды помещений.

В первом разделе описывается предметная область дипломной работы – учёт аренды помещений, а также рассматриваются общие принципы его ведения. Выделяются информационные потребности пользователей программного продукта, поясняется процесс ручного и автоматизированного учета аренды. Приводятся аналоги разрабатываемой системы, их недостатки, достоинства, а также их сравнение.

Второй раздел систематизирует требования, предъявляемые к программному продукту – его входные и выходные данные, процесс их обработки. Также здесь представлены функциональные возможности программы с их подробным описанием.

Обоснование выбора инструментов разработки, описание целесообразности различных решений для написания приложения и работы с базой данных представлены в третьем разделе.

Информация о проектировании базы данных и приложения помещена в четвертый раздел. Он содержит различные модели данных и UML-диаграммы, описывающие процессы, происходящие в приложении и состояние его элементов.

В пятом разделе описывается все, что относится к процессу конструирования программы – иерархии классов, потоки данных приложения, существующие классы и методы, их назначение.

Шестой раздел содержит информацию, касающуюся охраны труда – об искусственном освещении, вентиляции помещений и пожарной безопасности.

# ОБЩИЕ ПРИНЦИПЫ ВЕДЕНИЯ УЧЁТА АРЕНДЫ

## Структура организации бухгалтерского учета аренды

Исследование организации бухгалтерского учета аренды помещений направлено на выявление возможностей повышения эффективности управления этой сферой деятельности на базе использования средств вычислительной техники и современных экономико-математических методов.

Любая организация должна решать определённый комплекс поставленных перед ней задач в ее предметной области [14].

Исходя из этого, можно выделить некую организационную структуру, которая включает следующие сущности:

* арендаторы;
* здания;
* договоры;
* сотрудники;
* периоды;
* помещения.

Услуги, предоставляемые данной системой должны соответствовать потребностям людей, которые будут пользоваться ею.

В функции обслуживания входят: внесение договоров в базу данных, управление расчетами аренды, предоставление отчетов и данных поиска, вывод информации на печать.

## Информационные потребности пользователей

При анализе предметной области и работы пользователей, для которых будет полезен разрабатываемый продукт, были выявлены следующие информационные потребности:

* список арендаторов;
* перечень зданий;
* перечень договоров;
* перечень сотрудников;
* список месяцев аренды для каждого договора;
* перечень помещений.

Основными пользователями программного комплекса являются сотрудники бухгалтерии по учету аренды.

## Процесс учета аренды

### Описание ручного процесса

Регистрация и учет договоров по арендной плате в простейшем виде включает в себя чтение данных, полученных с утвержденного арендодателем договора, и последующую запись в определенный электронный файл, к примеру, формата Excel.

Для каждого нового договора вводится информация об арендаторе, о помещении, а также даты начала и окончания аренды. Время действия договора разбивается на периоды длительностью до месяца.

Периоды содержат расчеты для оплаты за аренду помещения, оборудования и коммунальные услуги. На их основании выписывается счета на оплату платежей, которые задействованы в договоре.

По итогу, данные ежемесячных расчетов копируются в различные бухгалтерские документы – накопительную ведомость, мемориальный ордер, ведомость начисления, акты сверок.

Все эти ведомости и счета обладают большим недостатком – они не взаимосвязаны друг с другом, поэтому чтобы сделать две ведомости, надо вручную копировать или вновь набирать все значения.

Так же ситуация обстоит при создания новых договоров, но с повторяющими арендаторами или помещениями – также приходится набирать все заново.

### Преимущества автоматизации

Автоматизированное рабочее место разработано для удобного использования и управления информационным потоком данных и автоматизирования расчетов в сфере бухгалтерского учета аренды помещений. Данные, вносимые работниками бухгалтерии, помогают организовать работу компаний, предприятий и государственных учреждений.

Создание автоматизированного рабочего места позволяет не только уменьшить трудозатраты на рабочий процесс и ускорить выполнение задач, связанных с расчетом аренды, но и предоставит цельную и логически структурированную базу данных вместо рассредоточенных по разным документам договоров, расчетов, счетов и ведомостей [15].

Это позволяет эффективно и быстро оценивать имеющиеся данные, и, при необходимости, сразу выдавать информацию об аренде или выводить её на печать.

Так как сущности связаны друг с другом и сохранены в базе данных, то благодаря предоставленному программой интерфейсу, создав единожды экземпляр арендатора при заключении первого договора нет необходимости создавать его заново при заключении каждого последующего договора.

Достаточно выбрать ранее созданного арендатора в списке арендаторов. Аналогичная ситуация обстоит и с любыми другими сущностями, участвующими в аренде. Например, здание аренды и помещение.

Данные об учете экспортируются в, различного рода, отчётности, счета и расчеты для последующего удобного использования.

## Анализ существующих программных аналогов

Система программ «1С: Предприятие 8» включает в себя платформу и прикладные решения, разработанные на ее основе, для автоматизации деятельности организаций и частных лиц. Сама платформа не является программным продуктом для использования конечными пользователями, которые обычно работают с одним из многих прикладных решений (конфигураций), разработанных на данной платформе. Такой подход позволяет автоматизировать различные виды деятельности, используя единую технологическую платформу [2].

Гибкость платформы позволяет применять «1С: Предприятие 8» в самых разнообразных областях:

* автоматизация производственных и торговых предприятий, бюджетных и финансовых организаций, предприятий сферы обслуживания и т.д.
* поддержка оперативного управления предприятием;
* автоматизация организационной и хозяйственной деятельности;
* ведение бухгалтерского учета с несколькими планами счетов и произвольными измерениями учета, регламентированная отчетность;
* широкие возможности для управленческого учета и построения аналитической отчетности, поддержка многовалютного учета;
* решение задач планирования, бюджетирования и финансового анализа;
* расчет зарплаты и управление персоналом;
* другие области применения.

Продукт предназначен для собственников коммерческой недвижимости, управляющих компаний, девелоперских организаций. Продукт позволяет повысить эффективность бизнеса компаний за счет автоматизации процессов по ведению реестрового учета объектов недвижимости, управлению договорами аренды и расчетами с арендаторами, эксплуатации объектов недвижимости. Программа обеспечивает решение задач бухгалтерского, управленческого, юридического и административного учета.

Программа позволяет эффективно управлять недвижимостью разных типов: торговыми и офисными центрами, рынками, выставочными площадями, складами и т.п.

Продукт разработан на последней версии технологической платформы «1С:Предприятие 8» и выпускается в двух вариантах- на основе типовых конфигураций «Бухгалтерия предприятия» и «Управление производственным предприятием».

Учет аренды, подобный тому, который реализуется данным проектом, осуществляет конфигурация "Аренда и управление недвижимостью" для "Бухгалтерия государственного учреждения".

Достоинства:

1. Мощный функционал
2. Гибкость в настройке
3. Максимальное соответствие стандартам
4. Удобный интерфейс
5. Постоянное развитие и поддержка
6. Отечественный продукт

Исходя из анализа конфигурации аренды и системы программ 1С в целом, можно выделить следующие недостатки, отталкивающие конечных пользователей:

1. Высокая сложность программного продукта. Высокая сложность затрудняет понимание пользователя и увеличивает порог вхождения для использования программы.
2. Стоимость ПО. 1С – коммерческая организация, за использование продуктов которой необходимо платить деньги. В виду невозможности некоторых организаций оплачивать программное обеспечение и тенденции к использованию бесплатных аналогов, это уводит от программного продукта множество потенциальных клиентов. В 1С же весьма дорогие расценки, и надо платить как за саму платформу, так и за большинство конфигураций, которые захочется использовать.
3. Нагроможденность. Этот недостаток присутствует как в функциональном плане в виде довольно большого скопления функций и возможностей в одном месте, так и в визуальном плане – интерфейс непростой.
4. Отсутствие узкой специализации. Система программ «1С: Предприятие» рассчитана на выполнение большого количества задач. В данной конфигурации «1С: Аренда и управление недвижимостью» есть огромное количество инструментов, которые не связаны напрямую с арендой, и которые сильно влияют на способность пользователей учиться пользоваться программой. Слишком большое количество нюансов в управлении арендой в том числе.
5. Долгая настройка конфигураций. Для подготовки рабочего окружения в первый раз требуется значительное количество времени, прежде чем можно будет приступить к работе.
6. Невозможность печатать отчёты в формате Excel.

## Преимущества разработки отдельной программы

Достоинства отдельной программы:

1. Узкая специализация. Возможность сосредоточиться на конкретной функциональности – аренде помещений. Это позволит избежать возможного испуга и недопониманий со стороны пользователя, который рассчитывает получить инструменты для решения своих задач, а по итогу получает сборник из множества намешанных возможностей.
2. Простота в обучении и легкий интерфейс, понятный и удобный для простых пользователей. Использование присущих предметной области терминов, минимализм в управлении и функционале отвергает возможную путаницу в работе приложения.
3. Распечатка в формат Excel. Для клиентов, которым исходя из требований к работе необходимо создавать и фиксировать Excel-документы по аренде, этот пункт будет весьма полезным.
4. Маленький размер и высокая производительность. В виду небольшого размера и отсутствия связи с сетью, и использованию встроенной БД, программный продукт устанавливается, запускается, и выдает полезный результат гораздо быстрей аналогов.
5. Бесплатность. Продукт не требует оплаты для пользователя.

# ТРЕБОВАНИЯ К ПРОГРАММЕ

## Входные данные

Входными данными для программного продукта являются данные, вводимые пользователем с клавиатуры, такие как:

* наименования организаций-арендодателей;
* личные данные сотрудников и клиентов-арендаторов;
* подробности договора;
* ежемесячные платежи.

Все вводимые значения являются строковыми и целочисленными данными.

При необходимости, входные данные можно предоставить с помощью файл базы данных формата «\*.sql».

## Процессы обработки договоров аренды

Процесс обработки обеспечивается через создание нового или выбор существующего договора посредством нажатия кнопок мыши над соответствующими элементами интерфейса программы. После ввода требуемой информации о договоре, происходит создание новой сущности договора в базе данных, которая включает в себя число месяцев, на которые заключен договор [16].

Программа позволяет:

1. добавлять, изменять, удалять информацию о всех сущностях, участвующих в аренде;
2. осуществлять поиск по договорам;
3. осуществлять вывод списков договоров, арендаторов и помещений на экран;
4. вывод счетов, расчетов и ведомостей на печать;

## Выходные данные

Выходными данными являются значения, выводимые программой пользователю на экран или полученные в процессе обработки данных, такие как:

* списки арендаторов, сотрудников, зданий, помещений и т.д.
* ежемесячные расчеты;
* счета и ведомости;

## Функциональные возможности программы

Программный продукт должен реализовывать следующие возможности:

1. Прием данных с помощью удобного интерфейса, абстрагированного от таблиц, колонок и столбцов, вместо которых будет занимать работа с цельными сущностями.
2. Автоматизация бухгалтерских расчётов, связанных с оплатой аренды помещений.
3. Система уведомлений для контроля за сроками оплаты согласно договору.
4. Гибкая настройка шаблонов, позволяющая использовать приложение без его привязки к конкретному учреждению.
5. Быстрое предоставление необходимой информации с помощью поиска.
6. Архивация данных. Экспорт для переноса. Автоматический бэкап для защиты от потери.
7. Объединение счетов за оплату в единые ведомости и мемориальные ордера.
8. Поддержка различных форматов для вывода данных.

## Подробное описание требований к функционалу

Данный подраздел охватывает причины выбора перечисленных требований, а также их подробное описание:

1. Рабочая область Excel представляет собой набор строк и столбцов, которые являются универсальными, годными для построения документов в любых сферах деятельности. Однако, для человека, работа с наглядным представлением информации в виде цельных готовых сущностей более понятна и удобна. Поэтому приложение должно реализовать интерфейс на основе присущих области бухгалтерского учета аренды сущностей, таких как арендатор, договор, помещение аренды, счетам.
2. Автоматизация расчетов должна предполагать возможность внесения самого минимального, но необходимого для расчётов объёма информации об аренде, чтобы существенная часть работы приходилась на программное обеспечение. Должна быть осуществлена оптимизация монотонных, однотипных и длительных действий.
3. Оповещения необходимы, когда появляется задолженность арендаторов по оплате за месяц, или истекает срок действия договора аренды, чтобы вовремя предупредить об этом оферента, на случай если он захочет продлить договор. Для этого необходимо создать систему, обладающую способностью отслеживать время и управлять датами. Благодаря работе этой системы должна быть возможность отмечать месяцы аренды как просроченные по оплате, и не просроченные.
4. Структура различных ведомостей, типовые формы счетов и расчетов могут быть различными, в зависимости от учреждения или предприятия, в котором проводится учет аренды. В следствии этого, для соответствия выходных документов требуемому виду, программа должна иметь функционал для принятия образцов документов, которые будут выступать в роли шаблонов, и на их основе выдавать новые документы.
5. Приложение может быть использовано на больших предприятиях и учреждениях, где необходим учёт множества договоров и связанных с ним сущностей. Для упрощения работы с неопределенным количеством объектов, должна быть разработана система поиска. Это позволит быстро найти необходимую информацию, например, получить историю оплаты прибывшего в отдел бухгалтерии арендатора.
6. В целях экспорта данных на другую машину или для предостережения пользователя от непредвиденной потери затраченного времени в следствии поломки, программа обязана иметь настройки для автоматического сохранения информации, делая сжатые копии БД.
7. Все собранные данные по счетам и расчетам должны собираться в единые типовые документы – различные накопительные ведомости, мемориальные ордера, ведомости начислений. Такая автоматизация процесса учёта аренды поможет избежать существенной доли монотонного копирования-вставки из файла в файл и уменьшит влияние человеческого фактора на результат, избавляя от потенциальных ошибок.
8. Вывод данных в Excel: от простого счета для оплаты по договору до объединенных в единый мемориальный ордер документов. Выходные данные должны иметь формат электронных таблиц, соответствующих формату Microsoft Excel, начиная от 2003 версии и выше, т.е. .xsl и .xslx. Поддержка старого формата необходима для максимального охвата потенциальной аудитории пользователей приложения. Возможна поддержка .doc формата.

# ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТОВ РАЗРАБОТКИ

## Целесообразность написания десктоп приложения

Исходя из требований к проекту, было принято решение о разработке десктоп приложения. Основное требование, которое привело к этому решению, было возможность доступа к приложению вне зависимости от наличия интернет-соединения или веб-обозревателя на рабочей станции.

Также, в виду того, что приложение задумано однопользовательским, его размещение в сети интернет не обусловлено целями и не соответствует решению задач проекта [17].

## Недостатки применения Microsoft Excel и макросов VBA

Задачи, стоящие перед проектом, могут быть осуществлены по большей степени с помощью специальной реализации языка программирования Visual Basic – VBA, разработанной для встраивания в приложения компании Microsoft.

VBA является скриптовым языком, то есть не требует компиляции, в связи с чем в программы, написанные на нём удобно вносить правки. Также к достоинствам можно отнести легкость в изучении и освоении языка, чтобы быстро начать писать полезные скрипты. VBA, будучи реализацией языка Visual Basic, предназначена для работы с приложениями Microsoft Office [18].

Однако этот язык обладает существенными недостатками. К ним относят открытость кода, в который может залезть даже случайный пользователь, и проблемы с обратной совместимостью, связанные с разными версиями пакета Office и других программ Microsoft, в следствии которых код, открытый в документе на одной машине, может не работать на другой.

В качестве основного, субъективного недостатка, который сыграл ключевую роль для отказа от VBA – ограниченность в построении хорошей архитектуры приложения, создающая невозможность работы со средними по размеру и большими проектами. Это скриптовый язык, живущий исключительно в среде приложений компании Microsoft.

## Описание стека технологий

### Язык программирования Java

Для разработки автоматизированного рабочего места выбран язык программирования Java. Огромная популярность языка, и, соответственно, большое сообщество обеспечивает возможность быстро найти ответы на всплывающие в процессе разработки вопросы для решения поставленных задач.

Одна из основных особенностей языка, из-за которой он был выбран – кроссплатформенность, достигаемая через использование виртуальной машины JVM. Поэтому приложение, написанное на Java, будет работать на любых операционных системах и независимо от оборудования, если для них установлена соответствующая среда JRE, необходимая для исполнения Java-приложений. Она уже включает в себя JVM и специальную библиотеку классов.

Также важной особенностью языка считается гибкость системы безопасности. Исполнение программы стеснена границами JVM, которая полностью контролирует процесс исполнения программы, и прерывает любые операции, выходящие за пределы полномочий, выданных приложению [3].

### Библиотека JavaFX

JavaFX – платформа на основе языка Java для создания приложений с насыщенным графическим интерфейсом.

Библиотека JavaFX предлагает все необходимые инструменты для построения интерфейсов на языке Java, и в то же время не требует слишком много затрат времени на изучение, в отличие от Spring – мощного фреймворка, который содержит в себе множество готовых решений, а также инструментов для разработки приложений. Сопоставив время, затрачиваемое на изучение инструментов разработки, и необходимые возможности, которые они предоставляют, для решения поставленных задач была выбрана библиотека JavaFX [5].

### Язык разметки FXML и конструктор JavaFX Scene Builder

Вместе с библиотекой JavaFX используется FXML – язык разметки пользовательского интерфейса, имеющий специальный синтаксис с xml-подобной структурой, и который предназначен для удобного написания визуальной части JavaFX приложения. Таким образом происходит разделение представления от реализации, что облегчает написание и понимание исходного кода.

Однако работать с графическими элементами интерфейса можно гораздо более просто, быстро и эффективно, чем писать их в текстовом редакторе. Поэтому для создания визуальной части приложения был выбран инструмент JavaFX Scene Builder, который позволяет собирать пользовательский интерфейс как конструктор, генерируя при этом соответствующий код [19].

## Выбор библиотеки Apache POI для работы с Microsoft Excel

Пакет программ Microsoft Office не содержит API, а значит не предусматривает работу с другими приложениями. Для работы с пакетом из другого приложения необходимо разобрать формат файлов, но, так как этот процесс длительный и усложняется намеренными запутыванием формата, необходимо выбрать библиотеку для чтения и записи файлов в форматах Microsoft Office.

Одной из таких библиотек является Apache POI. Её особенность в том, что она кроссплатформенна, имеет наибольший функционал среди конкурентов, предоставляет удобный API и, созданная ещё в начале 2000-х, имеет поддержку и по сей день [20].

## Аргументация в пользу системы управления базами данных h2

Выбор СУБД во многом определяется спецификой данных, размерами приложения и требованиями к скорости доступа к данным. Приложение имеет небольшой размер и потоки данных, не требует высокую скорость обработки данных.

Открытая кроссплатформенная СУБД h2, написанная на Java, имеет маленький вес, около 1мБ, т.к. не использует внешние библиотеки, и написана нативно. Также, в h2 применяется стандартный SQL-синтаксис для осуществления запросов, а грамотно написанная, подробная и понятная документация помогает быстро получать ответы на возникающие вопросы [6].

Согласно результатам сравнения с помощью тестов производительности, h2 и некоторых других популярных СУБД (в частности, HSQLDB, PostgreSQL и MySQL), h2 в целом более производительна, чем указанные СУБД. То есть в скорости обработки данных она не уступает другим СУБД, а встроенная версия h2, которая используется в данном приложении, не требует создания сервера и наличия интернет-соединения, отсутствие которых прописано в требованиях к проекту.

# ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ И ПРИЛОЖЕНИЯ

## Проектирование базы данных

### Сущности базы данных

Программа предназначена для работы с базой данных отдела бухгалтерии, которая включает следующие сущности:

* данные об арендаторах;
* данные о территориях;
* данные об сотрудниках;
* данные о договорах;
* данные о периодах аренды,
* данные о платежах.

Сущность договора является центральной при учете аренды помещений, с которой напрямую связаны такие сущности, как арендатор, территория, сотрудник и период.

Каждый арендатор может иметь один и более договоров, или не иметь их вовсе. Для каждого договора, напротив, обязан существовать и соответствовать один, и только один арендатор. В данном случае подходит связь один-ко многим. Такая же ситуация обстоит с территориями и сотрудниками по отношению к договору.

Периодов в договоре может быть множество, но как минимум один, иначе смысла в договоре нет. К тому же каждый период обязательно принадлежит договору, и только одному, поэтому связь один-ко многим.

В зависимости от содержания договора, некоторые виды платежей могут быть ненужными. Период может содержать платеж или не содержать его. Платеж же обязательно должен быть определен в периоде. Поэтому для всех видов платежей связь один-к одному является наиболее оптимальным вариантом [21].

В некоторых случаях платежи могут иметь остаток баланса – дебет и кредит. Но, так как в большинстве случаев эти поля в платежах будут пустыми и лишними, то для минимизации затрат ресурсов, поля дебета и кредита следует вынести из каждого типа платежа в отдельную таблицу баланса.

### Модель и архитектура базы данных

Предложена реляционная модель базы данных для хранения информации в виде отношений. Разработаны методы ограничения целостности базы данных [23].

Так как приложение предназначено для работы с базой данных на одном компьютере и будет работать без сети, то в качестве структуры взаимодействия с базой данных была выбрана централизованная архитектура.

### Нормализация отношений

Программный продукт разработан с созданием упорядоченной структуры данных третьей нормальной формой.

Все атрибуты являются атомарными и хранят лишь скалярные значения, поэтому первая нормальная форма достигнута.

Вторая нормальная форма предполагает, что каждый не ключевой атрибут неприводимо зависит от первичного ключа, и созданные таблицы отвечают этому требованию.

Не ключевые поля, содержание которых может относиться к нескольким таблицам, извлечены в новые таблицы. Это означает, что каждый не ключевой атрибут нетранзитивно зависит от первичного ключа, поэтому база данных соответствует третьей нормальной форме.

Выполненная нормализация таблиц позволяет существенно экономить используемое для работы время, обеспечивая понятный доступ к данным, отсутствие повторяемости значений и легкость масштабирования базы данных в будущем [23].

### Концептуальная модель базы данных

Концептуальная модель базы данных описывает информационные объекты предметной области учета аренды, а также связи между ними, и содержит 13 сущностей, 1 из которых выступает в роли справочника (см. рис. 4.1).



Рис. 4.1 Концептуальная модель базы данных

### Логическая модель базы данных

Логическая модель данных построена на основе реляционной модели данных. Сущности преобразованы в схемы отношений. Указаны первичные ключи, внешние – существуют в виде связей между отношениями (см. рис.4.2).



Рис. 4.2 Логическая модель базы данных

### Физическая модель базы данных

Физическая модель базы данных создана с учетом специфики СУБД MySQL, так как она имеет наибольшее синтаксическое сходство с используемой в проекте СУБД h2.



Рис. 4.3 Физическая модель базы данных

## Проектирование приложения

### Диаграмма вариантов использования

В качестве актора выступает будущий пользователь системы – бухгалтер.

Варианты использования позволяют моделировать диалог между активным субъектом и системой и отображают функции системы. С каждым вариантом использования связан определенный поток событий, происходящих по мере выполнения соответствующих функций системы. При описании потока событий определяется, что необходимо осуществить, и игнорируются аспекты того, как это делается [25].



Рис. 4.4 Диаграмма вариантов использования

Среди вариантов использования следует выделить варианты вывода в файл, настройки шаблонов, авто-заполнения полей и открытия списков – арендаторов, зданий или договоров.

### Диаграмма видов деятельности

Диаграмма видов деятельности отражает динамические аспекты поведения системы. По существу, эта диаграмма представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой.



Рис. 4.5 Диаграмма видов деятельности

Арендатор заключает договор, после чего бухгалтер должен его занести в базу данных. Спустя время, по окончанию периода, происходит расчет. Если счет не оплачен – на сумму накладывается пеня.

### Диаграмма состояний для счетов

Исходя из анализа предметной области, платежный счёт может быть в четырех состояниях:

* новый;
* оплаченный;
* задержанный;
* неоплаченный.

В состояние «Новый» счёт попадает сразу после своего создания и находится в нем до момента фиксации его бухгалтером в состояние «Оплаченный». Событием к переходу является поступление денег в кассу и получение квитанции об оплате. Условие перехода – оплата должна производиться не позднее 15 числа текущего месяца, когда был выписан счет. В случае если оплата не производится в течение отведенных 15 дней, к счету добавляется пеня, которая с каждым просроченным днем увеличивает платеж. Когда оплата вовсе не производится, счет переходит в состояние «Неоплаченный». Соответствующая диаграмма состояний представлена на рисунке 4.6:



Рис. 4.6 Диаграмма состояний платежного счета

### Диаграмма состояний для договоров

Исходя из анализа предметной области, договор может быть в трех состояниях:

* новый;
* действительный;
* расторгнутый.

В состояние «Новый» договор попадает сразу после своего создания и находится в нем до момента добавления периодов и каких-либо данных об аренде, в следствие чего переводится бухгалтером в состояние «Действительный». Условие перехода – новые данные либо продление договора. В случае, если оплата по счетам за периоды договора не производится, либо его срок действия заканчивается, договор переходит в состояние «Расторгнутый». Соответствующая диаграмма состояний представлена на рисунке 4.7:



Рис. 4.7 Диаграмма видов деятельности

### Диаграмма развертывания

Диаграмма развертывания является физической диаграммой в языке UML. Она отображает физические взаимосвязи между программными и аппаратным компонентами проектируемой системы.

Автоматизированное рабочее место бухгалтера устанавливается на рабочей станции вместе с сервером базы данных. Так как приложение и сервер находятся в пределах одной машины и не взаимодействуют в глобальной сети с другими машинами, то архитектура взаимодействия с базой данных является централизованной (см. рис. 4.8).



Рис. 4.8 Диаграмма развертывания

# РАЗРАБОТКА ПРОГРАММЫ

## Потоки данных

Управляющее воздействие: стандарт «Аренда», налоговый кодекс, земельный кодекс, постановление, инструкции и законы.

Механизмы: экономист.

Выходные данные: готовые Excel-файлы, сформированные по определенному шаблону.

Входные данные: договор с арендатором.

## Рабочий процесс пользователя и программы

Перед началом работы с приложением экономист-бухгалтер получает договор, заключенный арендатором. Главное окно приложение содержит множество сцен и меню (см. рис. 5.1).

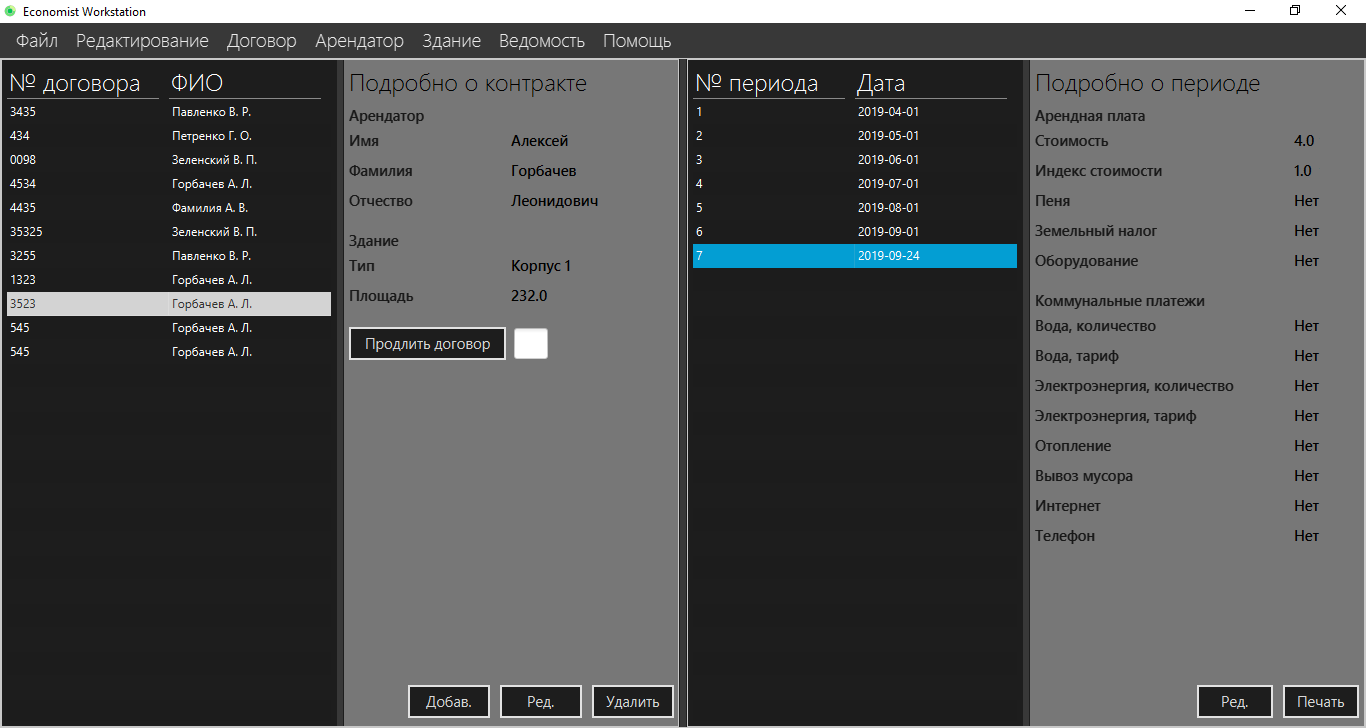


Рис. 5.1 Сцена управления договорами

На этом этапе заполняются все поля на основе сведений о договоре.

При создании договора открывается отдельное окно формы, которое содержит поле номера договора, период его действия, а также выпадающие списки арендаторов и зданий аренды (см. рис. 5.2).

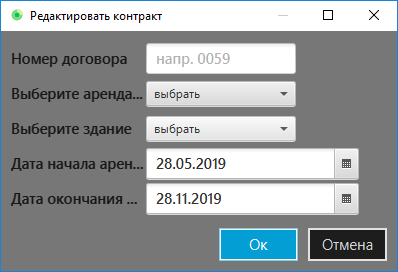


Рис. 5.2 Окно создания/редактирования договора

В списке договоров можно перейти к подробной информации о конкретном договоре, а также произвести расчет арендной платы, пени, коммунальных услуг, земельного налога и аренды оборудования.

Расчет производится помесячно, для каждого месяца формируется свой Excel-документ для распечатки счёта.

Существует 2 типа счетов – для оплаты аренды и для оплаты услуг – коммунальных, телефонных и интернет-услуг.

Счет выстраивается на основании расчетов, в зависимости от площади помещения, указанного в договоре, интенсивности пользования услугами, сроков оплаты.

Перед созданием договора необходимо сформировать список существующих арендаторов и зданий аренды, доступ и создание которых аналогично созданию договора (см. рис. 5.3).

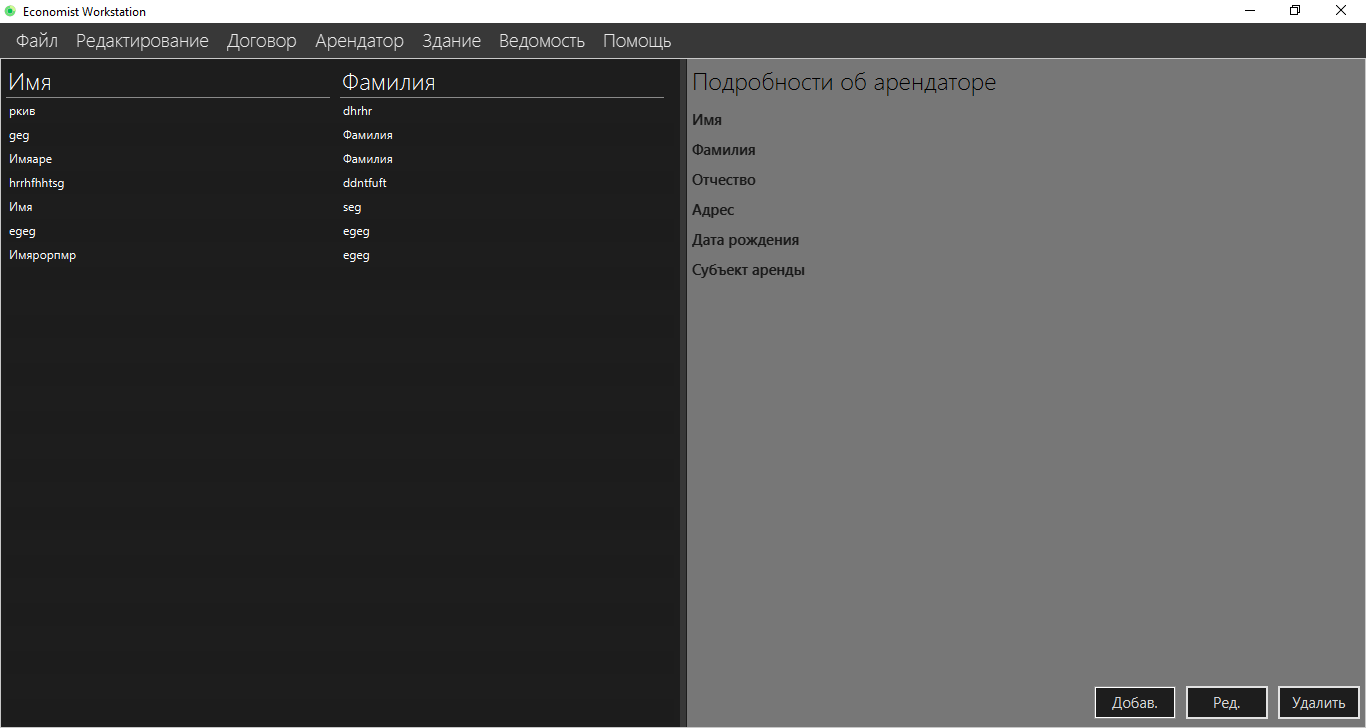


Рис. 5.3 Сцена управления арендаторами

Преимущество такого подхода состоит в возможности использовать повторяющиеся сущности от одного договора к другому, без необходимости их повторного ввода.

Занеся однажды арендатора в список арендаторов, нам не придется заново вводить данные об этом арендаторе при последующем заключении новых договоров с этим арендатором.

Это позволяет ускорить работу в дальнейшем.

Сущность здания представляет собой территорию с определенной совокупностью помещений, предоставленных к сдаче в аренду и обладающих единой системой тарифов для платежей, таких как тариф на водоснабжение и на электроэнергию.

При создании договора автоматически формируется список периодов (см. рис. 5.4.), в зависимости от указанных даты заключения и окончания действия договора.

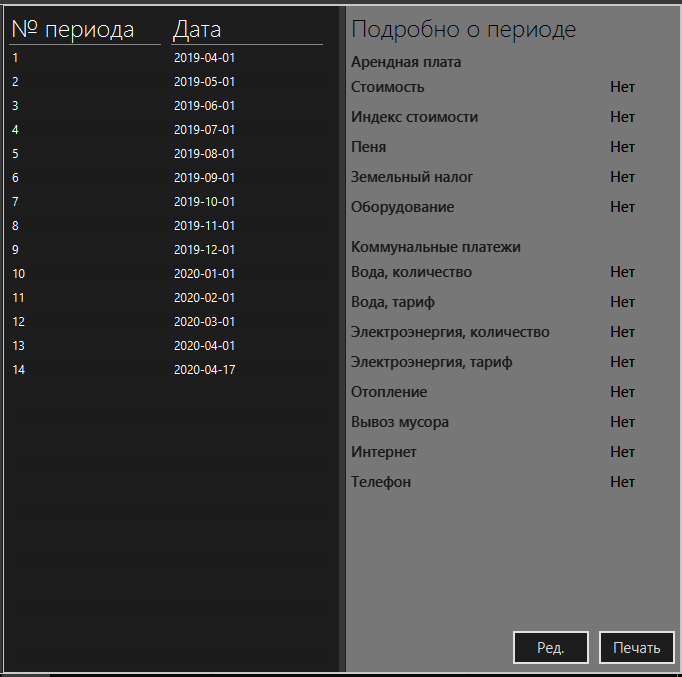


Рис. 5.4 Список периодов

Каждый период может содержать в себе 5 тип платежей – за аренду помещения, пеню, аренду оборудования, земельный налог и услуги. Чтобы включить платеж в период достаточно заполнить любое поле платежа (см. рис. 5.5).

При оплате счёта арендатором, экономист помечает соответствующий месяц погашенным.

В случае неуплаты, месяц помечается как просроченный, и идет автоматический расчет пени.

За некоторое количество дней до окончания срока аренды срабатывает система уведомлений.

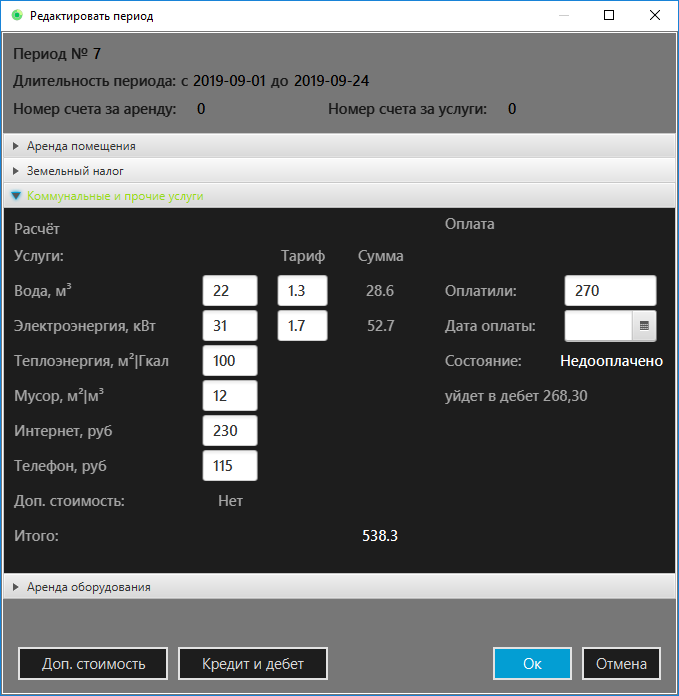


Рис. 5.5 Окно редактирования периода

Действие кнопки «Доп. стоимость» приводит к открытию окна, в котором можно указать дополнительную стоимость, в случае если прошлые платежи содержали ошибки (см. рис. 5.6).

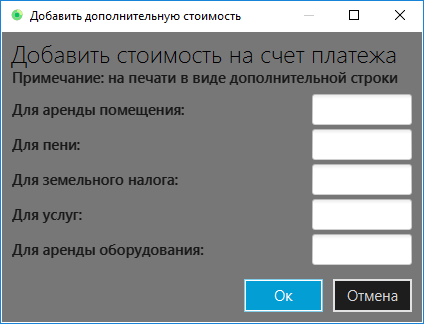


Рис. 5.6 Окно дополнительной стоимости

Кнопка «Кредит и дебет» открывается окна информации о балансе, накопленном от всех прошлых платежей (см. рис. 5.7).

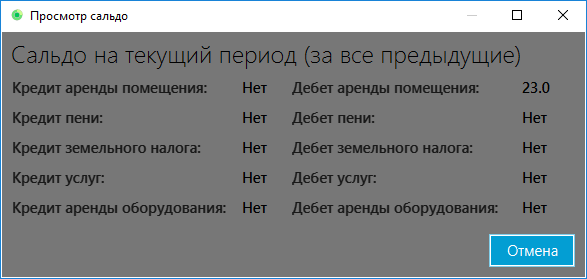


Рис. 5.7 Окно баланса

Действие кнопки «Печать» приводит к открытию формы для вывода счетов и расчетов в формат Excel по указанному пути (см. рис. 5.8).

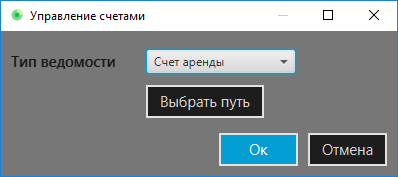


Рис. 5.8 Окно вывода

Меню содержит пункт для авто-заполнение некоторых полей, значения которых часто повторяются для конкретных периодов (см. рис. 5.9).

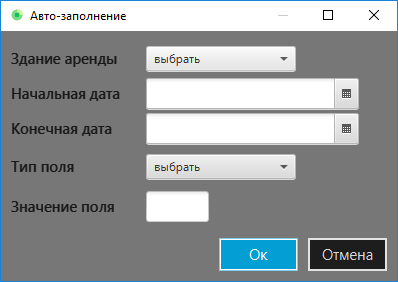


Рис. 5.9 Окно авто-заполнения

Для каждого месяца есть возможность сформировать обобщающие документы – накопительную ведомость, мемориальный ордер и ведомость начисления. Они охватывают различные данные по всем договорам за весь месяц.

## Диаграммы классов

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Классы в рассматриваемом текущем контексте представляют не абстрактные сущности предметной области, а элементы программной системы (в процессах проектирования и реализации).

Payment является абстрактным базовым классом для всех платежей, и содержит общее состояние и логику работы каждого вида платежей.

Общие состояния: идентификатор, стоимость и дата оплаты.

В общее поведение входят методы для работы с состояниями, методы проверки полей и расчета таблицы баланса (см. рис. 5.10).

payment-image

Рис. 5.10 Иерархия классов Payment

Controller является абстрактным базовым классом для всех контроллеров программы, и содержит общее состояние и логику работы каждого контроллера сцен.

Общее состояние: ссылка на текущее окно.

В общее поведение входят методы для работы с состоянием, открытия форм и оповещений, загрузки FXML представлений и создания диалога.

Некоторые из контроллеров, которые обязаны при инициализации производить работу с данными, реализуют интерфейс Initializable (см. рис. 5.11).

base-image

Рис. 5.11 Иерархия классов Controller

BaseFormController является абстрактным базовым классом для всех контроллеров форм программы, и наследуется от BaseController. Содержит общее состояние и логику работы каждого вида контроллеров форм.

Общее состояние: логическая переменная для проверки действий с окном.

В общее поведение входят методы для работы с состоянием, обработка кнопок и закрытие формы, проверка полей на заполнение данными [24].

Некоторые из контроллеров, которые обязаны при инициализации производить работу с данными, реализуют интерфейс Initializable (см. рис. 5.12).

baseform-image

Рис. 5.12 Иерархия классов BaseFormController

## Классы и методы

Создание приложения на JavaFX предполагает некоторый набор действий, совершаемый вне зависимости от назначения программы.

Для работы с JavaFX первое, что требуется сделать в главном классе – наследовать класс Application [1].

Следующий шаг – переопределить метод start, задав для него настройки главного окна программы (см. листинг 5.1).

public void start(Stage primaryStage) throws Exception {

this.primaryStage = primaryStage;

this.primaryStage.setTitle("Economist Workstation");

this.primaryStage.setMaximized(true);

this.primaryStage.getIcons().add(new Image("file:resources/images/icon.png"));

initRootLayout();

initMenu();

}

Листинг 5.1 Переопределение метода start

Метод start получает в качестве аргумента объект главного окна приложения или экрана – Stage, которому устанавливается заголовок, иконка и задаются размеры окна.

После этого инициализируется корневой контейнер и вставляется в главное окно приложения в виде объекта сцены Scene (см. листинг 5.2).

public void initRootLayout() throws IOException {

// Загружаем корневой макет из fxml файла.

FXMLLoader loader = new FXMLLoader();

loader.setLocation(

EconomistWorkstation.class.getResource("View/RootLayout.fxml"));

rootLayout = (BorderPane) loader.load();

// Отображаем сцену, содержащую корневой макет.

Scene scene = new Scene(rootLayout);

primaryStage.setScene(scene);

primaryStage.show();

}

Листинг 5.2 Инициализация корневого контейнера

При инициализации меню подключение происходит уже к верху созданной корневой сцены (см. листинг 5.3).

public void initMenu() throws IOException {

FXMLLoader loaderMenu = new FXMLLoader();

loaderMenu.setLocation(EconomistWorkstation.class.getResource("View/Menu.fxml"));

menu = (MenuBar) loaderMenu.load();

rootLayout.setTop(menu);

MenuController controller = loaderMenu.getController();

controller.setRoot(rootLayout);

controller.setDialogStage(primaryStage);

}

Листинг 5.3 Инициализация меню

В проекте все методы, содержащие логику загрузки FXML файла либо обрабатывают исключение с помощью блока try-catch, либо, как в листингах выше, пропускают его в вызывающий метод.

Для подключения к базе данных используется класс Database, который инкапсулирует в себе подключение через метод connect (см. листинг 5.4).

private void connect () {

try {

Class.forName("org.h2.Driver");

props = new Properties();

try(InputStream in = Files.newInputStream(Paths.get("src/config/database.properties"))){

props.load(in);

}

String url = props.getProperty("url");

String username = props.getProperty("username");

String password = props.getProperty("password");

conn = DriverManager.getConnection(url, username, password);

stmt = conn.createStatement();

} catch (IOException | ClassNotFoundException | SQLException ex) {

Logger.getLogger(EconomistWorkstation.class.getName()).log(Level.SEVERE, null, ex);

}

}

Листинг 5.4 Метод connect

Для простого доступа к базе данных из любого участка кода в приложении, получение доступа к ней было реализовано через паттерн «Одиночка». Таким образом, подключение к базе данных совершается лишь один раз. В последующие случаи обращения к базе данных возвращается уже созданное подключение (см. листинг 5.5).

public static Database getInstance() {

if (db == null) {

db = new Database();

db.connect();

}

return db;

}

Листинг 5.5 Метод getInstance

Абстрактный класс BaseController наследуется всеми контроллерами программы, содержащими логику создания новых сцен и управления их поведением.

В первую очередь стоит выделить метод для задания текущему классу объекта окна primaryStage, на основе которого было открыто текущее окно либо сцена (см. листинг 5.6).

protected Stage dialogStage;

public void setDialogStage(Stage dialogStage) {

this.dialogStage = dialogStage;

}

Листинг 5.6 Метод setDialogStage

Благодаря этому методу данный контроллер знает о существовании родительского окна, и может с ним взаимодействовать.

Следующий метод открывает окно формы, принимая в качестве параметров данные для передачи их контроллеру формы, название нового окна и путь к визуальному представлению – fxml файлу (см. листинг 5.7).

protected boolean showForm(ContractData data, String dialogName, String path) {

try {

FXMLLoader loader = new FXMLLoader();

AnchorPane container = loadFXML(path, loader);

Stage newDialogStage = createDialog(dialogName, container);

// Передаём адресата в контроллер.

BaseFormController controller = loader.getController();

controller.setDialogStage(newDialogStage);

controller.setData(data);

// Отображаем диалоговое окно и ждём, пока пользователь его не закроет

newDialogStage.showAndWait();

return controller.isOkClicked();

} catch (IOException ex) {

Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);

return false;

}

}

Листинг 5.7 Метод showForm

Здесь мы создаем загрузчик fxml файлов, загружаем соответствующий файл и получаем корневой контейнер из представления.

Контейнер необходимо передать в метод createDialog, который возвращает новый объект окна (см. листинг 5.8).

protected Stage createDialog(String title, Parent container) throws IOException {

// Создаём диалоговое окно Stage.

Stage newDialogStage = new Stage();

newDialogStage.setTitle(title);

newDialogStage.getIcons().add(new Image("file:resources/images/icon.png"));

newDialogStage.initModality(Modality.WINDOW\_MODAL);

newDialogStage.initOwner(dialogStage);

Scene scene = new Scene(container);

newDialogStage.setScene(scene);

return newDialogStage;

}

Листинг 5.8 Метод createDialog

В этом методе задается название создаваемой сцены, добавляется иконка, устанавливается модальный режим и происходит связывание с контейнером из fxml файла.

В завершение ко всему, класс BaseController содержит логику вывода на экран оповещений об ошибках, об успехе, а также информационных оповещений (см. листинг 5.9).

private void showAlert(String title, String header, String content,

AlertType type) {

Alert alert = new Alert(type);

alert.initOwner(dialogStage);

alert.setTitle(title);

alert.setHeaderText(header);

alert.setContentText(content);

alert.showAndWait();

}

Листинг 5.9 Метод showAlert

Меню позволяет обратиться ко всем возможным и предусмотренным требованиями к программному обеспечению возможностям.

Основной работа с приложением ведется через списки арендаторов, зданий и договоров. Для их открытия создан метод showList, который считывает идентификатор выбранного пункта меню, передает его методу getPathName для составления правильного пути к соответствующему файлу представления, и загружает fxml файл (см. листинг 5.10).

void showList(ActionEvent event) throws IOException {

currentOpenPage = (MenuItem) event.getSource();

String pathName = getPathName((MenuItem) event.getSource());

FXMLLoader loader = new FXMLLoader();

loader.setLocation(EconomistWorkstation.class.getResource(pathName));

Parent container = loader.load();

rootLayout.setCenter(container);

BaseController controller = loader.getController();

controller.setDialogStage(dialogStage);

}

Листинг 5.10 Метод showList

Кроме списков, меню предоставляет доступ к справочному окну «About», к окнам управления ведомостями и шаблонами, а также вспомогательные пункты для автоматизации учета аренды, например, окно авто-заполнения полей и обновления номеров счетов.

В приложении предусмотрен список арендаторов, за который отвечает класс RenterController, список зданий, за который ответственен BuildingController, и список договоров – ContractController.

Любой список реализует интерфейс Initializable, чтобы обработать представление сразу после загрузки всех его элементов (см. листинг 5.11).

@Override

public void initialize(URL url, ResourceBundle rb) {

renters = RenterModel.getRenters();

renterTable.setItems(renters);

firstNameColumn.setCellValueFactory(

cellData -> cellData.getValue().firstNameProperty());

lastNameColumn.setCellValueFactory(

cellData -> cellData.getValue().lastNameProperty());

// Очистка дополнительной информации об адресате.

showDetails(null);

// Слушаем изменения выбора, и при изменении отображаем

// дополнительную информацию об адресате.

renterTable.getSelectionModel().selectedItemProperty().addListener(

(observable, oldValue, newValue) -> showDetails(newValue));

}

Листинг 5.11 Метод initialize

Данный листинг представляет логику обработки списка арендаторов. При инициализации контроллера из базы данных извлекаются все сущности арендаторов и добавляются в таблицу. Далее происходит настройка колонок таблицы, в которые заносятся соответствующие поля класса Renter.

В конечном счете на каждую строку таблицы навешивается обработчик события щелчка мыши, который вызывает метод showDetails для отображения подробной информации об арендаторе (см. листинг 5.12).

public void showDetails(Renter renter) {

if (renter != null) {

firstNameLabel.setText(renter.getFirstName());

lastNameLabel.setText(renter.getLastName());

patronymicLabel.setText(renter.getPatronymic());

addressLabel.setText(renter.getAddress());

birthdayLabel.setText(renter.getBirthday());

subjectLabel.setText(renter.getSubject());

} else {

firstNameLabel.setText("");

lastNameLabel.setText("");

patronymicLabel.setText("");

addressLabel.setText("");

birthdayLabel.setText("");

subjectLabel.setText("");

}

}

Листинг 5.12 Метод showDetails

Здесь продемонстрировано заполнение меток значениями полей класса Renter, отвечающего за конкретного арендатора. В случае, если арендатора не существует, или он не выбран – метки заполняются пустыми строками.

Каждый из рассматриваемых контроллеров также имеет кнопки для добавления, редактирования и удаления соответствующих списку сущностей. Добавление арендатора показано на листинге 5.13.

private void handleNewRenter() {

Renter tempRenter = new Renter();

boolean okClicked = openRenterForm(tempRenter);

if (okClicked) {

int id = RenterModel.addRenter(tempRenter);

tempRenter.setId(id);

renters.add(tempRenter);

renterTable.getSelectionModel().select(tempRenter);

}

}

Листинг 5.13 Метод handleNewRenter

Здесь создается пустой объект класса Renter, который передается в форму арендаторов. В случае успешного заполнения полей текущий новый объект арендатора, подвергавшийся изменению в форме, добавляется в базу данных. Из база данных возвращается идентификатор добавленного арендатора, и после его присвоения объект арендатора добавляется в таблицу.

Редактирование имеет схожую логику с добавлением (см. листинг 5.14).

private void handleEditRenter() {

Renter selectedRenter = renterTable.getSelectionModel().getSelectedItem();

if (selectedRenter != null) {

boolean okClicked = openRenterForm(selectedRenter);

if (okClicked) {

RenterModel.updateRenter(selectedRenter.getId(), selectedRenter);

showDetails(selectedRenter);

}

} else {

showAlertWarning("Не выбран",

"Арендатор не выбран",

"Пожалуйста, выберите арендатора в таблице.");

}

}

Листинг 5.14 Метод handleEditRenter

Существенное отличие от предыдущего метода в том, что здесь не создается пустой объект типа Renter, а извлекается из таблицы уже существующий. В случае, если ни один арендатор не выбран, происходит вызов информационного оповещения. После успешного изменения данных об арендаторе, также обновляется и подробная информация о нём.

Удаление осуществляется исходя из выбранного элемента в списке, из которого мы получаем идентификатор, который используем для удаления из базы данных. В случае, если в таблице ничего не выбрано, вызывается оповещение (см. листинг 5.15).

private void handleDelete() {

int selectedIndex = renterTable.getSelectionModel().getSelectedIndex();

if (selectedIndex >= 0) {

Renter renter = renterTable.getSelectionModel().getSelectedItem();

int id = renter.getId();

renterTable.getItems().remove(renter);

RenterModel.deleteRenter(id);

} else {

showAlertWarning("Не выбран",

"Арендатор не выбран",

"Пожалуйста, выберите арендатора в таблице.");

}

}

Листинг 5.15 Метод handleDelete

Для работы с данными, их изменением и добавлением, используются формы. Класс, который содержит в себе всю общую логику для форм называется BaseFormController, который в свою очередь наследуется от BaseController, так как каждая форма имеет также функционал отдельной сцены (см. листинг 5.16).

protected abstract boolean isInputValid();

protected abstract void handleOk();

protected abstract void setData(ContractData data);

private boolean okClicked = false;

public boolean isOkClicked() {

return okClicked;

}

@FXML

private void handleCancel() {

dialogStage.close();

}

protected void closeForm() {

okClicked = true;

dialogStage.close();

}

Листинг 5.16 Основные методы формы

Каждая форма, наследующая BaseFormController, должна реализовать метод setData, который вызывается после открытия формы и передает контроллеру данные для работы. После заполнения формы метод handleOk должен обработать её и проверить на корректность заполнения с помощью метода isInputValid. В случае, если форма заполнена корректно, поле okClicked принимает значение true через метод closeForm, текущее окно закрывается и управление переходит к коду, вызвавшему форму.

Вызывающий код обращается к методу isOkClicked, тем самым проверяя, была ли форма успешно закрыта. Если действительно была нажата кнопка «Ok», то изменения сохраняются в базе данных.

Данный класс имеет также множество методов для проверки данных формы, некоторые из которых представлены на листинге 5.17.

protected boolean fieldIsEmpty(TextField tf) {

return tf.getText() == null || tf.getText().length() == 0;

}

protected boolean fieldIsEmpty(DatePicker dp) {

String text = dp.getEditor().getText();

try {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy");

LocalDate.parse(text, formatter);

return false;

} catch(DateTimeParseException e) {

System.out.println("Date with error.");

return true;

}

}

Листинг 5.17 Методы проверки данных

Первый метод принимает в качестве параметра текстовое поле, и проверяет его содержание на заполненность хотя бы одним символом.

Второй метод принимает поле даты, извлекает из него текст и пытается задать форматирование. В случае, если это не удается, ловится исключение.

Основная задача приложения – получить результат работы. В контексте учета аренды помещений в качестве выходных данных выступают различного рода документы – ведомости, счета, расчеты, ордера. Класс Document определяет базовый функционал для создания всех видов документов (см. листинг 5.18).

Document(ContractDataParameters data, String srcPath, String destPath) {

srcFile = new File(srcPath);

destFile = new File(destPath);

prepareStream();

this.parser = createTagParser(data);

}

Листинг 5.18 Конструктор класса Document

Все классы, наследующие Document, передают данные в его конструктор. В нём задаются новые файлы на основании переданных параметров путей, и с помощью метода prepareStream в объекте создаются потоки ввода и вывода.

Данные, необходимые для построения документа передаются в метод createTagParser, который является абстрактным. Это важный момент, поскольку таким образом клиентский код не привязывается к конкретному классу документа и парсера тегов. Каждый вид документов возвращает свой собственный парсер. Это распространённая логика работы имеет название порождающего паттерна проектирования «Фабричный метод».

Основной метод класса Document – метод печати print. Он запускает конструирование документа, записывает его в выходной поток и закрывает его (см. листинг 5.19).

public void print() {

iterateWorkbook();

write();

}

protected void iterateWorkbook() {

for (Sheet currSheet : workbook) {

this.sheet = currSheet;

for (Row row : sheet) {

iterateCells(row);

}

clearRows();

removeRows();

}

}

Листинг 5.19 Метод print

Следует отметить метод iterateWorkbook, который занимается прохождением по всему шаблону документа, на основании которого будет построен новый. В конце обработки листа происходит очистка и удаление ненужных строк. Удаление в конце обработки рабочего листа обусловлено необходимостью избежать неконтролируемое изменение порядка строк.

Классы документов, отвечающие за построение ведомостей, также имеют методы для добавления строк, в зависимости от количества сущностей, передаваемых в массиве данных (см. листинг 5.20).

private void prepareDocument(ContractDataParameters data) {

sizeList = data.getDataList().size();

addTemplateRows(sizeList);

}

protected int addTemplateRows(int contractsSize) {

if (indexStartRow != -1)

for (int i = indexStartRow + 1; i < contractsSize + indexStartRow; i++) {

copyRow(indexStartRow, indexStartRow + 1);

}

return indexStartRow;

}

Листинг 5.20 Методы класса Document

Класс Parser занимается преобразованием тегов, записанных в ячейках шаблонов, в соответствующие им значения. Исходные данные, необходимые для преобразования, принимаются через конструктор. В случае отсутствия каких-либо данных, во избежание исключений и для заполнения ячеек нулевыми значениями, создаются пустые экземпляры объектов (см. листинг 5.21).

Parser(ContractData data) {

useContractData(data);

}

protected void useContractData(ContractData data) {

period = data.getPeriod();

renter = data.getRenter();

building = data.getBuilding();

contract = data.getContract();

rent = isExist(period.getRentPayment())

? period.getRentPayment() : new Rent();

fine = isExist(period.getFinePayment())

? period.getFinePayment() : new Fine();

taxLand = isExist(period.getTaxLandPayment())

? period.getTaxLandPayment() : new TaxLand();

equipment = isExist(period.getEquipmentPayment())

? period.getEquipmentPayment() : new Equipment();

services = isExist(period.getServicesPayment())

? period.getServicesPayment() : new Services();

}

Листинг 5.21 Конструктор класса Parser

Для преобразования тегов вызывается метод convertTags, принимающий ячейку в качестве параметра (см. листинг 5.22).

public void convertTags(Cell cell) {

this.cell = cell;

cellString = cell.getStringCellValue();

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

Matcher matcher = pattern.matcher(cellString);

while(matcher.find()) {

String foundedTag = matcher.group();

System.out.println(foundedTag);

String newValue = parse(foundedTag);

if (!isExist(newValue)) return;

String result = cellString.replaceAll(foundedTag, newValue);

cell.setCellValue(result);

cellString = result;

}

}

Листинг 5.22 Метод convertTags

Этот метод проводит проверку текста, содержащегося в ячейке, и в случае нахождения там тега – вызывает метод parse (см. листинг 434). Если значение не было возвращено – значит в строке, вероятно, больше чем один тег, поэтому мы проходим далее по циклу перебора (см. листинг 5.23).

protected String parse(String foundedTag) {

String newValue = "<Tag not founded>";

if ("<square>".equals(foundedTag)) {

newValue = safeDecFormat(building.getSquare(), Locale.getDefault());

}

if ("<subject>".equals(foundedTag)) {

newValue = renter.getSubject();

}

if ("<fullName>".equals(foundedTag)) {

newValue = renter.getFullName();

}

if ("<numContract>".equals(foundedTag)) {

newValue = Integer.toString(period.getIdContract());

}

return newValue;

}

Листинг 5.23 Метод parse

Также стоит отметить метод для поиска определенного тега, который пригождается для составления ведомостей, чтобы узнать первую строку с реальными данными, обычно обозначаемую тегом <num> (см. листинг 5.24).

public static boolean findTag(Cell cell, String srcPattern) {

String cellString = cell.getStringCellValue();

Pattern pattern = Pattern.compile(srcPattern);

Matcher matcher = pattern.matcher(cellString);

while(matcher.find()) {

return true;

}

return false;

}

Листинг 5.24 Метод findTag

Программный код любого периода, входящего в договор, оперирует такими сущностями как платежи. Существуют такие разновидности платежей, как аренда за помещения, пеня, земельный налог, аренда оборудования и услуги.

Рассмотрим конструктор класса Payment, показанный на листинге 5.25.

public Payment(Object paid, String datePaid) {

this.id = new SimpleIntegerProperty(0);

this.paid = new SimpleObjectProperty(paid);

this.datePaid = new SimpleStringProperty(datePaid);

this.state = new SimpleStringProperty();

this.info = new SimpleStringProperty();

}

Листинг 5.25 Конструктор класса Payment

Согласно листингу, каждый платеж имеет идентификатор, оплаченную сумму, дату оплаты и два строковых свойства, в которых хранится информация о долгах и переплатах, показываемая пользователю.

База данных хранит платежи в разных таблицах, поэтому доступ к их модификации, добавлению и удалению имеет разные запросы, которые должны быть реализованы наследниками (см. листинг 5.26).

public abstract PreparedStatement getInsertStatement(Database db) throws SQLException;

public abstract PreparedStatement getUpdateStatement(Database db) throws SQLException;

public abstract PreparedStatement getDeleteStatement(Database db) throws SQLException;

Листинг 5.26 Методы доступа к БД

Платеж должен предоставлять всю сумму для оплаты с помощью метода sumToPay, проверять свои поля на наполненность данными методом isEmpty, делать копию текущего экземпляра класса copy, проверять заполнение полей формы с помощью checkFields, привязывать платеж к периоду bindPeriod, сохранять значения из формы в текущий экземпляр saveValuesOf (см. листинг 5.27).

public abstract Double sumToPay();

public abstract boolean isEmpty();

public abstract Payment copy();

public abstract void bindPeriod(Period period);

public abstract String checkFields(Field field);

public abstract void saveValuesOf(Field field);

public abstract boolean fieldsIsFilled(Field field);

public abstract void fill(Field field);

public abstract void setLabels(Map<String, Label> labels);

Листинг 5.27 Абстрактные методы класса Payment

Так как существует отдельная таблица с балансом для всех платежей, то они должны обеспечить методы для работы с соответствующими полями таблицы баланса (см. листинг 5.28).

public abstract Double getCredit(BalanceTable balanceTable);

public abstract void setCredit(BalanceTable balanceTable, Double credit);

public abstract Double getDebit(BalanceTable balanceTable);

public abstract void setDebit(BalanceTable balanceTable, Double debit);

Листинг 5.28 Методы сальдо для Payment

# ОХРАНА ТРУДА

## Искусственное освещение производственных помещений

При соответствующем технико-экономическом обосновании в целях безопасного производства работ, сохранения жизни и здоровья работников, минимизации последствий аварий, инцидентов и других происшествий, охраны объектов экономики на их территориях, в зданиях и помещениях должно монтироваться и надежно функционировать искусственное освещение [7].

Виды искусственного освещения:

1. Рабочее освещение;
2. Аварийное освещение;
3. Дежурное освещение;
4. Охранное освещение;
5. Ремонтное освещение.

Ниже представлено более подробное описание видов.

1. Рабочее освещение– это освещение, обеспечивающее нормируемые осветительные условия (освещенность, качество освещения) в помещениях и в местах производства работ вне зданий и создающее благоприятные условия труда, возможность передвижения людей и транспорта во время отсутствия или недостатка естественного освещения. Рабочее освещение устанавливается в виде общего освещения или комбинированного освещения.

Общее освещение – это освещение, при котором светильники размещаются в верхнем лоне помещения равномерно (общее равномерное освещение) или применительно к расположению оборудования.

Комбинированное освещение – это освещение, при котором к общему освещению добавляется местное. Местным освещением называют освещение, дополнительное к общему, создаваемое светильниками, концентрирующими световой поток непосредственно на рабочих местах. Общее освещение производственных помещений устраивается так, чтобы исключалось ослепление крановщиков в кабинах кранов.

2. Аварийное освещение разделяется на освещение безопасности и эвакуационное освещение.

Освещение безопасности предназначено для продолжения работы при аварийном отключении рабочего освещения. Светильники рабочего освещения и светильники освещения безопасности в производственных и общественных зданиях должны питаться от независимых источников.

Светильники и световые указатели эвакуационного освещения в производственных зданиях с естественным освещением присоединяются к сети, не связанной с сетью рабочего освещения, начиная от щита подстанции (распределительного пункта освещения) или, при наличии только одного ввода, начиная от вводного распределительного устройства.

В производственных зданиях без естественного света в помещениях, где может одновременно находиться 20 человек и более, независимо от наличия освещения безопасности должно предусматриваться эвакуационное освещение по основным проходам и световые указатели «выход», автоматически переключаемые при прекращении их питания на третий независимый внешний или местный источник (аккумуляторная батарея, дизель-генераторная установка и т.п.), не используемый в нормальном режиме для питания рабочего освещения, освещения безопасности и эвакуационного освещения, или светильники эвакуационного освещения и указатели «выход» должны иметь автономный источник питания.

При отнесении всех или части светильников освещения безопасности и эвакуационного освещения к особой группе первой категории по надежности электроснабжения, надо предусматривать их питание от третьего независимого источника. Использование сетей, питающих силовые электроприемники, для питания освещения безопасности и эвакуационного освещения в производственных зданиях без естественного освещения не допускается.

3. Дежурное освещение следует устраивать для охраны и осмотра помещений в нерабочее время. Для этой цели следует выделять часть светильников рабочего или аварийного освещения. В помещениях с непрерывным технологическим процессом дежурное освещение не требуется.

4. Охранное освещение должно предусматриваться при отсутствии специальных технических средств охраны вдоль границ и территорий промышленных объектов в ночное время. Общее освещение территории организации допускается с помо­щью прожекторов и (или) светильников напряжением 127 либо 220 В.

5. Ремонтное (переносное) освещение предусматривают на таких операциях, где стационарным освещением невозможно создать нормируемый уровень освещенности, а также для осмотра, ремонта и наладки производственного (технологического) оборудования. Переносные ручные светильники ремонтного освещения должны питаться от сети напряжения не выше 50 В, а при повышенной опасности по­ражения электротоком – не выше 12 В [8].

## Вентиляция производственных помещений. Её виды

Под вентиляцией понимают систему мероприятий и устройств, предназначенных для обеспечения на постоянных рабочих местах, в рабочей и обслуживаемой зонах помещений метеорологических условий и чистоты воздушной среды, соответствующих гигиеническим и техническим требованиям.

Рационально спроектированные и правильно эксплуатируемые вентиляционные системы способствуют улучшению самочувствия работающих и повышению производительности труда. По имеющимся данным, кондиционирование воздуха может повысить производительность труда на 4... 10%.

Системы вентиляции классифицируют по способу перемещения воздуха, направлению потока воздуха, зоне действия, времени работы.

В зависимости от способа перемещения воздуха различают вентиляцию естественную и механическую. Естественную вентиляцию подразделяют на организованную и неорганизованную. Организованная естественная вентиляция может быть канальной и бесканальной (аэрация).

Аэрация – это организованная управляемая естественная вентиляция, осуществляемая за счет разности гравитационного давления наружного и внутреннего воздуха и действия ветра.

В этом случае необходимый воздухообмен в здании может быть обеспечен только при устройстве открывающихся отверстий достаточной площади. Аэрацию предусматривают, если с ее помощью могут быть обеспечены нормируемые условия воздушной среды и если она допустима по технологическим условиям. Особенно целесообразно устройство аэрации в горячих цехах.

При неорганизованной естественной вентиляции воздух поступает и удаляется через щели, окна, двери и т. п. Если перемещение воздуха производят с помощью вентиляторов с электроприводом, вентиляцию называют механической.

Существуют и смешанные системы вентиляции.

В зависимости от направления потока воздуха вентиляция бывает приточной и вытяжной. Вентиляцию в производственных зданиях обычно выполняют приточно-вытяжной [9].

По зоне действия различают вентиляцию общеобменную, местную и смешанную (комбинированную). При общеобменной вентиляции происходит обмен воздуха во всем помещении. Она применяется тогда, когда выделения вредных факторов незначительны и равномерно распределены по всему объему помещения. Местная вентиляция может быть вытяжной и приточной. Вытяжная предназначена для удаления воздуха непосредственно от мест образования или выхода вредных выделений, приточная - для подачи чистого воздуха на определенные рабочие места или участки.

Процесс создания и автоматического поддержания в производственном помещении определенных параметров воздушной среды называют кондиционированием. При кондиционировании независимо от наружных метеорологических условий и режима работы технологического оборудования в помещении поддерживаются необходимые температура, относительная влажность, чистота и скорость движения воздуха. Различают комфортное и технологическое кондиционирование воздуха. Целью комфортного кондиционирования является создание в помещениях воздушной среды, наиболее благоприятной для работы и отдыха людей, что способствует повышению производительности труда и снижению заболеваемости. Технологическое кондиционирование воздуха обеспечивает создание параметров воздушной среды, удовлетворяющих требованиям технологического процесса. При этом для помещений, где работают люди, учитываются и санитарно-гигиенические требования [10].

В холодный и переходный периоды года воздух, подаваемый в помещение системами механической вентиляции, должен подогреваться. Температуру и скорость подаваемого воздуха следует рассчитывать в соответствии с «Указаниями по расчету приточных воздухоразделительных устройств» (серия АЗ-358) для того, чтобы метеорологические условия в рабочей зоне соответствовали нормам. При естественной вентиляции холодный воздух не должен непосредственно действовать на работающих. Необходимо предусматривать воздушные или воздушно-тепловые завесы, причем на время открывания ворот и дверей температура воздуха на постоянных рабочих местах в зависимости от тяжести выполняемой работы не должна опускаться ниже 8...14°С.

Правильный выбор систем вентиляции имеет большое санитарно-гигиеническое и экономическое значение, при этом можно руководствоваться несколькими общими положениями:

* необходимо максимально использовать местные вытяжные системы для предотвращения распространения вредностей по всему объему помещения;
* механическую вентиляцию следует применять только в тех случаях, когда требуемые параметры воздушной среды не могут быть обеспечены естественной вентиляцией;
* при проектировании механической вентиляции необходимо предусматривать установку резервных вентиляторов с электродвигателями, включаемыми автоматически, или сооружать не менее двух приточных и двух вытяжных установок, обеспечивающих при включении одной из них объем соответственно вытяжки или притока не менее 50 % требуемого воздухообмена;
* приточную вентиляцию целесообразно совмещать с воздушным отоплением, при этом необходимо учитывать возможность выхода из строя вентиляционной системы и необходимость в любом случае поддержания температуры в помещении не ниже +5°С;
* температуру воздуха, выходящего из воздухораспределителей, расположенных в пределах рабочей зоны, следует принимать не более 45 °С и не менее 5°С.

Большое внимание при проектировании и эксплуатации вентиляционных установок должно быть уделено борьбе с шумом. Работа вентиляционных установок любых систем характеризуется значительным шумом, который ни в коем случае не должен увеличивать уровень шума в помещениях выше допускаемого нормами [11].

## Основы пожарной безопасности

Обеспечение пожарной безопасности достигает­ся выполнением **требований пожарной безопасности** – специальных условий социального и технического характера, установленных законодательством государства, нормативными документами или уполномоченным государственным органом.

Нормативными документами по пожарной безопасности являют­ся национальные стандарты, своды правил, а также иные документы, содержащие требования пожарной безопасности [12].

**По виду горючего материала пожары классифицируются следующим образом:**

* **А** – пожары твердых горючих веществ и материалов
* **В** – пожары горючих жидкостей или плавящихся твердых веществ и материалов;
* **С** – пожары газов
* **D** – пожары металлов
* **Е** – пожары горючих веществ и материалов электроустановок, находящихся под напряжением
* **F** – пожары ядерных материалов, радиоактивных отходов и ра­диоактивных веществ.

**К опасным факторам пожара, воздействующим на людей и иму­щество, относятся:**

* пламя и искры;
* тепловой поток;
* повышенная температура окружающей среды;
* повышенная концентрация токсичных продуктов горения и термического разложения;
* пониженная концентрация кислорода;
* снижение видимости в дыму.

**Сопутствующими проявлениями опасных факторов пожара яв­ляются:**

* осколки, части разрушившихся зданий, сооружений, технологических установок, оборудования, агрегатов, изделий и иного имущества;
* радиоактивные и токсичные вещества и материалы, попавшие в окружающую среду из разрушенных технологических установок, обо­рудования, агрегатов, изделий и иного имущества;
* вынос высокого напряжения на токопроводящие части технологических установок, оборудования, агрегатов, изделий и иного имущества;
* опасные факторы взрыва, происшедшего вследствие пожара;
* воздействие огнетушащих веществ.

**Руководители организаций обязаны:**

* соблюдать требования пожарной безопасности, а также вы­полнять предписания, постановления и иные законные требования должностных лиц пожарной охраны;
* разрабатывать и осуществлять меры по обеспечению пожарной безопасности;
* проводить противопожарную пропаганду, а также обучать своих работников мерам пожарной безопасности;
* включать в коллективный договор (соглашение) вопросы по­жарной безопасности;
* содержать в исправном состоянии системы и средства противо­пожарной защиты, включая первичные средства тушения пожаров, не допускать их использования не по назначению;
* оказывать содействие пожарной охране при тушении пожаров, установлении причин и условий их возникновения и развития, а также при выявлении лиц, виновных в нарушении требований пожарной без­опасности и возникновении пожаров;
* предоставлять в установленном порядке при тушении пожаров на территориях организаций необходимые силы и средства;
* обеспечивать доступ должностным лицам пожарной охраны при осуществлении ими служебных обязанностей на территории, в зда­ния, сооружения и на иные объекты организаций;
* предоставлять по требованию должностных лиц государствен­ного пожарного надзора сведения и документы о состоянии пожарной безопасности в организациях, в том числе о пожарной опасности про­изводимой ими продукции, а также о происшедших на их территориях пожарах и их последствиях;
* незамедлительно сообщать в пожарную охрану о возникших пожарах, неисправностях имеющихся систем и средств противопожар­ной защиты, об изменении состояния дорог и проездов;
* содействовать деятельности добровольных пожарных.

Руководители организаций осуществляют непосредственное руководство системой пожарной безопасности в пределах своей компетенции на подведомственных объектах и несут персональную ответственность за соблюдение требований пожарной безопасности.

**Каждый гражданин при обнаружении пожара или признаков горе­ния (задымление, запах гари и т.п.) должен** незамедлительно сообщить об этом по телефону в пожарную охрану и принять по возможности меры по эвакуации людей.

Лица, виновные в нарушении требований пожарной безопасности, а также за иные правонарушения в области пожарной безопасности, мо­гут быть привлечены к дисциплинарной, административной или уголов­ной ответственности в соответствии с действующим законодательством [13].

# ВЫВОД

В дипломной работе был проведён анализ предметной области бухгалтерского учета и разработано автоматизированное рабочее место бухгалтера для учета аренды помещений.

Среди достоинств выделены: удобный интерфейс, возможность печати в формат Excel, маленький размер, высокая производительность, бесплатность и кроссплатформенность.

Основными информационными потребностями пользователей программного продукта являются управление расчетами аренды, предоставление отчетов и данных поиска, вывод информации на печать.

Автоматизация позволяет уменьшить трудозатраты на рабочий процесс и ускорить выполнение задач, связанных с расчетом аренды.

Конфигурация «Аренда и управление недвижимостью» платформы «1С:Предприятие» является главным конкурентом в деле автоматизации учета аренды и основным аналогом разработанного программного продукта. Среди недостатков выделены необходимость оплаты, нагроможденный интерфейс и отсутствие автономности. Достоинства: мощный функционал, соответствие стандартам, поддержка [4].

Систематизированы требования к программному продукту и описаны функциональные возможности.

Входные данные приложения: личные данные сотрудников и арендаторов, подробности договора, ежемесячные платежи.

Выходные данные приложения: списки арендаторов и договоров, ежемесячные расчеты, счета и ведомости.

В качестве языка программирования использован язык Java. Построение интерфейса выполнено с помощью библиотеки JavaFX. Управление данными происходит через СУБД h2.

Спроектирована база данных третьей нормальной формы на 13 таблиц, 1 из которых в виде справочника. Построены инфологическая, даталогическая и физическая модели данных. Даталогическая модель ориентирована на реляционную модель базы данных. На языке UML выполнены диаграммы вариантов использования, видов деятельности, состояний и развертывания, а также построены иерархии классов приложения.

Приведены основные существующие классы и методы, а также их назначение.

Описанные положения об искусственном освещении, вентиляции помещений и пожарной безопасности соответствует государственным и межгосударственным нормам и стандартам.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Классы. Объектно-ориентированное программирование. Классы и объекты [Электронный ресурс] – <https://metanit.com/java/tutorial/3.1.php>
2. Программный комплекс 1С [Электронный ресурс] – <http://1c.ua/v8/index.php>
3. Документация языка программирования Java [Электронный ресурс] –<https://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html>
4. Конфигурация 1С: «Аренда и управление недвижимостью» [Электронный ресурс] – <https://solutions.1c.ru/catalog/arbgu/features>
5. Краткое руководство по использованию JavaFX [Электронный ресурс] – <https://metanit.com/java/javafx/1.1.php>
6. Документация базы данных h2 [Электронный ресурс] – <http://www.h2database.com/html/main.html>
7. **СНиП 23-05-2010**. Естественное и искусственное освещение. – Введ. 2011-01-01. – М.: 2010 год – 13 с.
8. Охрана труда в России – ОХРАНА ТРУДА [Электронный ресурс] – https://websot.jimdo.com/обучение/учебный-курс/искусственное-освещение-производственных-территорий-зданий-и-помещений/
9. Студенческая библиотека [Электронный ресурс] – <https://students-library.com/library/read/36928-ventilacia-proizvodstvennyh-pomesenij>
10. **ГОСТ 22270-2018**. Системы отопления, вентиляции и кондиционирования. Термины и определения. – Введ. 2018-01-11. – М.: Стандартинформ, 2018 год – 5 с.
11. **ГОСТ 32548-2013**. Вентиляция зданий. Воздухораспределительные устройства. Общие технические условия. – Введ. 2015-01-01. – М.: Стандартинформ, 2014 год – 8 с.
12. **ГОСТ 12.1.004-91**. Система стандартов безопасности труда. Пожарная безопасность. Общие требования. – Введ. 1992-07-01. – М.: Стандартинформ, 2006 год – 3 с.
13. Охрана труда в России – ОХРАНА ТРУДА [Электронный ресурс] – <https://websot.jimdo.com/обучение/учебный-курс/общие-положения-и-основные-понятия-пожарной-безопасности/>
14. Аренда: бухгалтерский учет и налогообложение [Электронный ресурс] – <http://www.nalvest.com/nv-articles/detail.php?ID=26701>
15. Предметная область автоматизации [Электронный ресурс] – <https://works.doklad.ru/view/uo58IXzAIWo.html>
16. Составление требований и спецификаций [Электронный ресурс] – <https://life-prog.ru/1_2857_sostavlenie-trebovaniy-i-spetsifikatsiy.html>
17. Разработка программного обеспечения [Электронный ресурс] – <https://customserv.ru/software>
18. Определение VBA, преимущества, возможности применения [Электронный ресурс] – <http://www.askit.ru/custom/vba_office/m1/01_02_basis_vba.htm>
19. Часть 1: Scene Builder [Электронный ресурс] – <https://code.makery.ch/ru/library/javafx-tutorial/part1/>
20. Busy Developers' Guide to HSSF and XSSF Features [Электронный ресурс] – <https://poi.apache.org/components/spreadsheet/quick-guide.html>
21. Основные понятия баз данных [Электронный ресурс] – <http://inf.susu.ac.ru/Klinachev/lc_sga_26.htm>
22. Архитектура СУБД. Архитектура баз данных. [Электронный ресурс] – <https://zametkinapolyah.ru/zametki-o-mysql/arxitektura-subd-arxitektura-baz-dannyx-logicheskaya-struktura-subd-opisanie-dannyx-v-baze-dannyx-bazy-dannyx-sxema-dannyx.html>
23. Описание основных приемов нормализации базы данных [Электронный ресурс] – <https://support.microsoft.com/ru-ru/help/283878/description-of-the-database-normalization-basics>
24. Построение диаграммы классов [Электронный ресурс] – <https://flexberry.github.io/ru/gpg_class-diagram.html>
25. Построение диаграммы вариантов использования [Электронный ресурс] – <https://flexberry.github.io/ru/gpg_use-case-diagram.html>

# ПРИЛОЖЕНИЯ

## Приложение А. Скрипт генерации базы данных

/\*==============================================================\*/

/\* DBMS name: MySQL 5.0 \*/

/\* Created on: 27.05.2019 22:14:22 \*/

/\*==============================================================\*/

drop table if exists post;

drop table if exists extraCost;

drop table if exists balance;

drop table if exists contract;

drop table if exists employee;

drop table if exists period;

drop table if exists renter;

drop table if exists "Платеж на аренду оборудования";

drop table if exists "Платеж на аренду помещения";

drop table if exists "Платеж на земельный налог";

drop table if exists "Платеж на пени";

drop table if exists "Платеж на услуги";

drop table if exists Территория;

drop table if exists заключает;

/\*==============================================================\*/

/\* Table: post \*/

/\*==============================================================\*/

create table post

(

id\_post int not null,

name text,

primary key (id\_post)

);

/\*==============================================================\*/

/\* Table: extraCost \*/

/\*==============================================================\*/

create table extraCost

(

id\_extra\_cost int not null,

id\_period int not null,

extra\_cost\_rent float,

extra\_cost\_fine float,

extra\_cost\_services float,

extra\_cost\_tax\_land float,

extra\_cost\_equipment float,

primary key (id\_extra\_cost)

);

/\*==============================================================\*/

/\* Table: balance \*/

/\*==============================================================\*/

create table balance

(

id\_balance int not null,

id\_period int not null,

debit\_rent float,

credit\_rent float,

debit\_fine float,

credit\_fine float,

debit\_tax\_land float,

credit\_tax\_land float,

debit\_equipment float,

credit\_equipment float,

debit\_services float,

credit\_services float,

primary key (id\_balance)

);

/\*==============================================================\*/

/\* Table: contract \*/

/\*==============================================================\*/

create table contract

(

id\_contract int not null,

id\_employee int not null,

date\_start date,

date\_end date,

primary key (id\_contract)

);

/\*==============================================================\*/

/\* Table: employee \*/

/\*==============================================================\*/

create table employee

(

id\_employee int not null,

id\_post int not null,

name\_employee text,

surname\_employee text,

patronymic\_employee text,

primary key (id\_employee)

);

/\*==============================================================\*/

/\* Table: period \*/

/\*==============================================================\*/

create table period

(

id\_period int not null,

id\_rent int,

id\_services int,

id\_contract int not null,

id\_extra\_cost int,

id\_fine int,

id\_equipment int,

id\_tax\_land int,

id\_balance int,

number int,

number\_rent\_acc int,

number\_services\_acc int,

date\_end\_period date,

primary key (id\_period)

);

/\*==============================================================\*/

/\* Table: renter \*/

/\*==============================================================\*/

create table renter

(

id\_renter int not null,

first\_name text,

last\_name text,

patronymic text,

address text,

birthday date,

subject text,

primary key (id\_renter)

);

/\*==============================================================\*/

/\* Table: "Платеж на аренду оборудования" \*/

/\*==============================================================\*/

create table "Платеж на аренду оборудования"

(

id\_equipment int not null,

id\_period int not null,

cost\_equipment float,

paid\_equipment float,

date\_paid\_equipment date,

primary key (id\_equipment)

);

/\*==============================================================\*/

/\* Table: "Платеж на аренду помещения" \*/

/\*==============================================================\*/

create table "Платеж на аренду помещения"

(

id\_rent int not null,

id\_period int not null,

cost float,

index\_cost float,

paid\_rent float,

date\_paid\_rent date,

primary key (id\_rent)

);

/\*==============================================================\*/

/\* Table: "Платеж на земельный налог" \*/

/\*==============================================================\*/

create table "Платеж на земельный налог"

(

id\_tax\_land int not null,

id\_period int not null,

tax\_land float,

paid\_tax\_land float,

date\_paid\_tax\_land date,

primary key (id\_tax\_land)

);

/\*==============================================================\*/

/\* Table: "Платеж на пени" \*/

/\*==============================================================\*/

create table "Платеж на пени"

(

id\_fine int not null,

id\_period int not null,

fine float,

paid\_fine float,

date\_paid\_fine date,

primary key (id\_fine)

);

/\*==============================================================\*/

/\* Table: "Платеж на услуги" \*/

/\*==============================================================\*/

create table "Платеж на услуги"

(

id\_services int not null,

id\_period int not null,

count\_water float,

tariff\_water float,

count\_electricity float,

tariff\_electricity float,

cost\_meter\_heading float,

cost\_meter\_garbage float,

cost\_internet float,

cost\_telephone float,

paid\_services float,

date\_paid\_services date,

primary key (id\_services)

);

/\*==============================================================\*/

/\* Table: Территория \*/

/\*==============================================================\*/

create table Территория

(

id\_building int not null,

id\_contract int,

type text,

square float,

cost\_balance float,

cost\_residue float,

primary key (id\_building)

);

/\*==============================================================\*/

/\* Table: заключает \*/

/\*==============================================================\*/

create table заключает

(

id\_contract int not null,

id\_renter int not null,

primary key (id\_contract, id\_renter)

);

alter table Entity\_16 add constraint FK\_determines3 foreign key (id\_period)

references period (id\_period) on delete restrict on update restrict;

alter table Entity\_17 add constraint FK\_contains5 foreign key (id\_period)

references period (id\_period) on delete restrict on update restrict;

alter table contract add constraint FK\_contributes foreign key (id\_employee)

references employee (id\_employee) on delete restrict on update restrict;

alter table employee add constraint FK\_occupies foreign key (id\_post)

references Entity\_14 (id\_post) on delete restrict on update restrict;

alter table period add constraint FK\_contains2 foreign key (id\_extra\_cost)

references Entity\_16 (id\_extra\_cost) on delete restrict on update restrict;

alter table period add constraint FK\_contains3 foreign key (id\_tax\_land)

references "Платеж на земельный налог" (id\_tax\_land) on delete restrict on update restrict;

alter table period add constraint FK\_contains6 foreign key (id\_services)

references "Платеж на услуги" (id\_services) on delete restrict on update restrict;

alter table period add constraint FK\_determines foreign key (id\_rent)

references "Платеж на аренду помещения" (id\_rent) on delete restrict on update restrict;

alter table period add constraint FK\_determines1 foreign key (id\_contract)

references contract (id\_contract) on delete restrict on update restrict;

alter table period add constraint FK\_determines2 foreign key (id\_fine)

references "Платеж на пени" (id\_fine) on delete restrict on update restrict;

alter table period add constraint FK\_determines4 foreign key (id\_equipment)

references "Платеж на аренду оборудования" (id\_equipment) on delete restrict on update restrict;

alter table period add constraint FK\_determines5 foreign key (id\_balance)

references Entity\_17 (id\_balance) on delete restrict on update restrict;

alter table "Платеж на аренду оборудования" add constraint FK\_contains4 foreign key (id\_period)

references period (id\_period) on delete restrict on update restrict;

alter table "Платеж на аренду помещения" add constraint FK\_contains foreign key (id\_period)

references period (id\_period) on delete restrict on update restrict;

alter table "Платеж на земельный налог" add constraint FK\_determines7 foreign key (id\_period)

references period (id\_period) on delete restrict on update restrict;

alter table "Платеж на пени" add constraint FK\_contains1 foreign key (id\_period)

references period (id\_period) on delete restrict on update restrict;

alter table "Платеж на услуги" add constraint FK\_determines6 foreign key (id\_period)

references period (id\_period) on delete restrict on update restrict;

alter table Территория add constraint FK\_indicates foreign key (id\_contract)

references contract (id\_contract) on delete restrict on update restrict;

alter table заключает add constraint FK\_concludes foreign key (id\_renter)

references renter (id\_renter) on delete restrict on update restrict;

alter table заключает add constraint FK\_concludes2 foreign key (id\_contract)

references contract (id\_contract) on delete restrict on update restrict;

## Приложение Б. Точка входа в приложение

package economistworkstation;

import economistworkstation.Controller.MenuController;

import java.io.IOException;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.stage.Stage;

import javafx.scene.control.MenuBar;

import javafx.scene.layout.AnchorPane;

import javafx.scene.layout.BorderPane;

public class EconomistWorkstation extends Application {

private Stage primaryStage;

private BorderPane rootLayout;

private MenuBar menu;

@Override

public void start(Stage primaryStage) throws Exception {

this.primaryStage = primaryStage;

this.primaryStage.setTitle("Economist Workstation");

this.primaryStage.setMaximized(true);

initRootLayout();

initMenu();

}

public void initRootLayout() throws IOException {

// Загружаем корневой макет из fxml файла.

FXMLLoader loader = new FXMLLoader();

loader.setLocation(

EconomistWorkstation.class.getResource("View/RootLayout.fxml"));

rootLayout = (BorderPane) loader.load();

// Отображаем сцену, содержащую корневой макет.

Scene scene = new Scene(rootLayout);

primaryStage.setScene(scene);

primaryStage.show();

}

public void initMenu() throws IOException {

FXMLLoader loaderMenu = new FXMLLoader();

loaderMenu.setLocation(

EconomistWorkstation.class.getResource("View/Menu.fxml"));

menu = (MenuBar) loaderMenu.load();

rootLayout.setTop(menu);

MenuController controller = loaderMenu.getController();

controller.setRoot(rootLayout);

controller.setMainApp(this);

}

public void showMainPage() throws IOException {

// Загружаем сведения об адресатах.

FXMLLoader loader = new FXMLLoader();

loader.setLocation(

EconomistWorkstation.class.getResource("View/MainPage.fxml"));

AnchorPane mainPage = (AnchorPane) loader.load();

// Помещаем сведения об адресатах в центр корневого макета.

rootLayout.setCenter(mainPage);

}

public Stage getPrimaryStage() {

return primaryStage;

}

public static void main(String[] args) {

launch(args);

}

}

## Приложение В. Класс базы данных Database

package economistworkstation;

import java.sql.\*;

import java.nio.file.\*;

import java.io.\*;

import java.util.\*;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author fnajer

\*/

public class Database {

private static Database db;

Properties props = null;

public Connection conn = null;

public Statement stmt = null;

private void connect () {

try {

Class.forName("org.h2.Driver");

props = new Properties();

try(InputStream in = Files.newInputStream(Paths.get("src/config/database.properties"))){

props.load(in);

}

String url = props.getProperty("url");

String username = props.getProperty("username");

String password = props.getProperty("password");

conn = DriverManager.getConnection(url, username, password);

stmt = conn.createStatement();

} catch (IOException | ClassNotFoundException | SQLException ex) {

Logger.getLogger(EconomistWorkstation.class.getName()).log(Level.SEVERE, null, ex);

}

}

public static Database getInstance() {

if (db == null) {

db = new Database();

db.connect();

}

return db;

}

}

## Приложение Г. Листинг базовых классов

package economistworkstation.Controller;

import economistworkstation.ContractData;

import economistworkstation.EconomistWorkstation;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.image.Image;

import javafx.scene.layout.AnchorPane;

import javafx.stage.Modality;

import javafx.stage.Stage;

Класс BaseController.

public abstract class BaseController {

protected Stage dialogStage;

public void setDialogStage(Stage dialogStage) {

this.dialogStage = dialogStage;

}

protected boolean showForm(ContractData data, String dialogName, String path) {

try {

FXMLLoader loader = new FXMLLoader();

AnchorPane container = loadFXML(path, loader);

Stage newDialogStage = createDialog(dialogName, container);

// Передаём адресата в контроллер.

BaseFormController controller = loader.getController();

controller.setDialogStage(newDialogStage);

controller.setData(data);

// Отображаем диалоговое окно и ждём, пока пользователь его не закроет

newDialogStage.showAndWait();

return controller.isOkClicked();

} catch (IOException ex) {

Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);

return false;

}

}

protected AnchorPane loadFXML(String path, FXMLLoader loader) throws IOException {

// Загружаем fxml-файл и создаём новую сцену

// для всплывающего диалогового окна.

loader.setLocation(EconomistWorkstation.class.getResource(path));

return (AnchorPane) loader.load();

}

protected Stage createDialog(String title, Parent container) throws IOException {

// Создаём диалоговое окно Stage.

Stage newDialogStage = new Stage();

newDialogStage.setTitle(title);

newDialogStage.getIcons().add(new Image("file:resources/images/icon.png"));

newDialogStage.initModality(Modality.WINDOW\_MODAL);

newDialogStage.initOwner(dialogStage);

Scene scene = new Scene(container);

newDialogStage.setScene(scene);

return newDialogStage;

}

private void showAlert(String title, String header, String content,

AlertType type) {

Alert alert = new Alert(type);

alert.initOwner(dialogStage);

alert.setTitle(title);

alert.setHeaderText(header);

alert.setContentText(content);

alert.showAndWait();

}

protected void showAlertWarning(String title, String header, String content) {

showAlert(title, header, content, AlertType.WARNING);

}

protected void showAlertSuccess(String title, String header, String content) {

showAlert(title, header, content, AlertType.INFORMATION);

}

protected void showAlertError(String title, String header, String content) {

showAlert(title, header, content, AlertType.ERROR);

}

}

Класс BaseFormController.

package economistworkstation.Controller;

import economistworkstation.ContractData;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.time.format.DateTimeParseException;

import javafx.fxml.FXML;

import javafx.scene.control.DatePicker;

import javafx.scene.control.TextField;

public abstract class BaseFormController extends BaseController {

protected abstract boolean isInputValid();

protected abstract void handleOk();

protected abstract void setData(ContractData data);

private boolean okClicked = false;

public boolean isOkClicked() {

return okClicked;

}

@FXML

private void handleCancel() {

dialogStage.close();

}

protected void closeForm() {

okClicked = true;

dialogStage.close();

}

protected boolean fieldIsEmpty(TextField tf) {

return tf.getText() == null || tf.getText().length() == 0;

}

protected boolean fieldIsEmpty(DatePicker dp) {

String text = dp.getEditor().getText();

try {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy");

LocalDate.parse(text, formatter);

return false;

} catch(DateTimeParseException e) {

System.out.println("Date with error.");

return true;

}

}

protected boolean costIsInvalid(TextField tf) {

try {

Double.parseDouble(tf.getText());

return false;

} catch(NumberFormatException e) {

return true;

}

}

protected boolean errorNotExist(String errorMessage) {

if (errorMessage.length() == 0) {

return true;

} else {

showAlertError("Некорректные данные",

"Заполните поля корректно",

errorMessage);

return false;

}

}

}

Класс Document.

package economistworkstation.Entity;

import economistworkstation.ContractDataParameters;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.ArrayList;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.apache.poi.ss.usermodel.Cell;

import org.apache.poi.ss.usermodel.CellType;

import static org.apache.poi.ss.usermodel.CellType.STRING;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.Workbook;

public abstract class Document {

private final File srcFile;

private final File destFile;

private OutputStream out;

private InputStream input;

protected Workbook workbook;

protected Sheet sheet;

protected Parser parser;

private String logName;

Document(ContractDataParameters data, String srcPath, String destPath) {

srcFile = new File(srcPath);

destFile = new File(destPath);

prepareStream();

this.parser = createTagParser(data);

}

private void prepareStream() {

try {

out = new FileOutputStream(destFile);

input = new FileInputStream(srcFile);

workbook = new HSSFWorkbook(input);

} catch (IOException ex) {

Logger.getLogger(Document.class.getName()).log(Level.SEVERE, null, ex);

}

}

protected void write() {

try {

workbook.write(out);

out.close();

System.out.println(String.format("Обновлен документ %s", getLogName()));

} catch (IOException ex) {

Logger.getLogger(Document.class.getName()).log(Level.SEVERE, null, ex);

}

}

public void print() {

iterateWorkbook();

write();

}

protected void iterateWorkbook() {

for (Sheet currSheet : workbook) {

this.sheet = currSheet;

for (Row row : sheet) {

iterateCells(row);

}

clearRows();

removeRows();

}

}

private void iterateCells(Row row) {

for (Cell cell : row) {

CellType cellType = cell.getCellType();

if (cellType == STRING) {

parser.convertTags(cell);

}

}

}

private void clearRows() {

ArrayList<Integer> rowsForClear = parser.getRowsForClear();

for (int numRow : rowsForClear) {

Row row = sheet.getRow(numRow);

for (Cell cell : row) {

cell.setCellValue("");

}

}

}

private void removeRows() {

ArrayList<Integer> rowsForDelete = parser.getRowsForDelete();

int deletion = 0;

for (int row : rowsForDelete) {

removeRow(row - deletion);

deletion++;

}

}

private void removeRow(int rowIndex) {

int lastRowNum=sheet.getLastRowNum();

if(rowIndex>=0&&rowIndex<lastRowNum){

sheet.shiftRows(rowIndex+1,lastRowNum, -1);

}

if(rowIndex==lastRowNum){

Row removingRow=sheet.getRow(rowIndex);

if(removingRow!=null){

sheet.removeRow(removingRow);

}

}

}

protected String getLogName() {

return logName;

}

protected void setLogName(String logName) {

this.logName = logName;

}

public abstract Parser createTagParser(ContractDataParameters data);

}

Класс Parser.

package economistworkstation.Entity;

import economistworkstation.ContractData;

import static economistworkstation.Util.Util.isExist;

import java.math.RoundingMode;

import java.text.DecimalFormat;

import java.text.DecimalFormatSymbols;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.Locale;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.poi.ss.usermodel.Cell;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.util.CellReference;

public abstract class Parser {

protected Period period;

protected Renter renter;

protected Building building;

protected Contract contract;

protected Cell cell;

protected String cellString;

protected Rent rent;

protected Fine fine;

protected TaxLand taxLand;

protected Equipment equipment;

protected Services services;

Parser() {

}

Parser(ContractData data) {

useContractData(data);

}

protected void useContractData(ContractData data) {

period = data.getPeriod();

renter = data.getRenter();

building = data.getBuilding();

contract = data.getContract();

rent = isExist(period.getRentPayment())

? period.getRentPayment() : new Rent();

fine = isExist(period.getFinePayment())

? period.getFinePayment() : new Fine();

taxLand = isExist(period.getTaxLandPayment())

? period.getTaxLandPayment() : new TaxLand();

equipment = isExist(period.getEquipmentPayment())

? period.getEquipmentPayment() : new Equipment();

services = isExist(period.getServicesPayment())

? period.getServicesPayment() : new Services();

}

public void convertTags(Cell cell) {

this.cell = cell;

cellString = cell.getStringCellValue();

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

Matcher matcher = pattern.matcher(cellString);

while(matcher.find()) {

String foundedTag = matcher.group();

System.out.println(foundedTag);

String newValue = parse(foundedTag);

if (!isExist(newValue)) return;

String result = cellString.replaceAll(foundedTag, newValue);

cell.setCellValue(result);

cellString = result;

}

}

protected String formatDate(LocalDate date) {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy");

return date.format(formatter);

}

protected double getDoubleValue(Double value) {

String decValue = safeDecFormat(value, Locale.US);

return Double.parseDouble(decValue);

}

protected String getStringValue(Double value) {

String decValue = safeDecFormat(value, Locale.US);

return decValue;

}

protected DecimalFormat decFormat(Locale locale) {

DecimalFormatSymbols otherSymbols = new DecimalFormatSymbols(locale);

DecimalFormat df = new DecimalFormat("#.##", otherSymbols);

df.setRoundingMode(RoundingMode.HALF\_UP);

return df;

}

protected String getColumnLetter(Cell cell) {

int columnNumber = cell.getColumnIndex();

return CellReference.convertNumToColString(columnNumber);

}

protected String safeDecFormat(Double doubleObject, Locale locale) {

DecimalFormat df = decFormat(locale);

try {

return df.format(doubleObject);

} catch(IllegalArgumentException e) {

System.err.println(String.format(

"%s: value '%s' is not correct. Replaced by 0.0",

this.getClass().getSimpleName(),

doubleObject));

return "0.0";

}

}

private final ArrayList<Integer> rowsForClear = new ArrayList();

private final ArrayList<Integer> rowsForDelete = new ArrayList();

public ArrayList<Integer> getRowsForClear() {

return rowsForClear;

}

public ArrayList<Integer> getRowsForDelete() {

return rowsForDelete;

}

protected boolean rowWillClear(double value, Cell cell) {

if (value <= 0) {

Row row = cell.getRow();

int rowIndex = row.getRowNum();

if (!rowsForClear.isEmpty() && rowsForClear.contains(rowIndex))

return true;

rowsForClear.add(rowIndex);

return true;

}

return false;

}

protected boolean rowWillDelete(double value, Cell cell) {

if (value <= 0) {

Row row = cell.getRow();

int rowIndex = row.getRowNum();

if (!rowsForDelete.isEmpty() && rowsForDelete.contains(rowIndex))

return true;

rowsForDelete.add(rowIndex);

return true;

}

return false;

}

public static boolean findTag(Cell cell, String srcPattern) {

String cellString = cell.getStringCellValue();

Pattern pattern = Pattern.compile(srcPattern);

Matcher matcher = pattern.matcher(cellString);

while(matcher.find()) {

return true;

}

return false;

}

protected String parse(String foundedTag) {

String newValue = "<Tag not founded>";

if ("<square>".equals(foundedTag)) {

newValue = safeDecFormat(building.getSquare(), Locale.getDefault());

}

if ("<subject>".equals(foundedTag)) {

newValue = renter.getSubject();

}

if ("<fullName>".equals(foundedTag)) {

newValue = renter.getFullName();

}

if ("<numContract>".equals(foundedTag)) {

newValue = Integer.toString(period.getIdContract());

}

if ("<dateStartContract>".equals(foundedTag)) {

LocalDate date = LocalDate.parse(contract.getDateStart());

newValue = formatDate(date);

}

if ("<currentMonthNameAndYear>".equals(foundedTag)) {

LocalDate date = LocalDate.parse(period.getEndPeriod());

int monthNum = date.getMonth().getValue();

String monthName = period.getMonthName(monthNum, false);

int monthYear = date.getYear();

newValue = monthName + ' ' + Integer.toString(monthYear);

}

if ("<monthNameAndYear>".equals(foundedTag)) {

LocalDate date = LocalDate.parse(period.getEndPeriod());

int monthNum = date.getMonth().minus(1).getValue();

String monthName = period.getMonthName(monthNum, false);

int monthYear = date.getYear();

if (monthNum == 12) {

monthYear--;

}

newValue = monthName + ' ' + Integer.toString(monthYear);

}

if ("<user>".equals(foundedTag)) {

User user = new User();

newValue = user.getFullName();

}

return newValue;

}

}

Класс Payment.

package economistworkstation.Entity;

import economistworkstation.Database;

import static economistworkstation.Util.Util.isExist;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Map;

import java.util.Objects;

import javafx.beans.property.IntegerProperty;

import javafx.beans.property.ObjectProperty;

import javafx.beans.property.SimpleIntegerProperty;

import javafx.beans.property.SimpleObjectProperty;

import javafx.beans.property.SimpleStringProperty;

import javafx.beans.property.StringProperty;

import javafx.scene.control.Label;

public abstract class Payment {

private final IntegerProperty id;

private final ObjectProperty<Double> paid;

private final StringProperty datePaid;

public Payment() {

this(null, null);

}

public Payment(Object paid, String datePaid) {

this.id = new SimpleIntegerProperty(0);

this.paid = new SimpleObjectProperty(paid);

this.datePaid = new SimpleStringProperty(datePaid);

this.state = new SimpleStringProperty();

this.info = new SimpleStringProperty();

}

public abstract PreparedStatement getInsertStatement(Database db) throws SQLException;

public abstract PreparedStatement getUpdateStatement(Database db) throws SQLException;

public abstract PreparedStatement getDeleteStatement(Database db) throws SQLException;

public abstract Double sumToPay();

public abstract boolean isEmpty();

public abstract Payment copy();

public abstract void bindPeriod(Period period);

public abstract String checkFields(Field field);

public abstract void saveValuesOf(Field field);

public abstract boolean fieldsIsFilled(Field field);

public abstract void fill(Field field);

public abstract void setLabels(Map<String, Label> labels);

public abstract Payment createNewPayment();

public abstract Double getCredit(BalanceTable balanceTable);

public abstract void setCredit(BalanceTable balanceTable, Double credit);

public abstract Double getDebit(BalanceTable balanceTable);

public abstract void setDebit(BalanceTable balanceTable, Double debit);

public int getId() {

return id.get();

}

public void setId(int id) {

this.id.set(id);

}

public Double getPaid() {

return paid.get();

}

public void setPaid(Double paid) {

this.paid.set(paid);

}

public String getDatePaid() {

return datePaid.get();

}

public void setDatePaid(String datePaid) {

this.datePaid.set(datePaid);

}

public Double getDiff() {

try {

Double needPay = sumToPay();

Double diff = needPay - safeGetSum(getPaid());

return diff;

} catch(NullPointerException e) {

System.out.println(String.format(

"%s: %s diff is null",

this.getClass().getSimpleName(),

this));

return 0.0;

}

}

public Double safeGetPaid() {

if (getPaid() == null)

return 0.0;

else

return getPaid();

}

protected Double safeGetSum(Double value) {

if (value == null)

return 0.0;

else

return value;

}

protected Double safeGetSum(Double value1, Double value2) {

try {

return value1 \* value2;

} catch(NullPointerException e) {

return 0.0;

}

}

private void setEmptyBalance() {

setState("Нет платежа");

setInfo("");

}

protected boolean fieldsIsEmpty() {

try {

Double needPay = sumToPay();

Double paid = safeGetSum(getPaid());

return needPay == 0.0 && paid == 0.0;

} catch(NullPointerException e) {

return true;

}

}

public void prepareToDelete() {

setPaid(-1.0);

}

private BalanceTable balanceTable;

private BalanceTable nextBalanceTable;

public void calcPartOfBalance(BalanceTable balanceTable, BalanceTable nextBalanceTable) {

if (fieldsIsEmpty() && !isExist(balanceTable)) {

setEmptyBalance();

return;

}

if (!isExist(balanceTable))

balanceTable = new BalanceTable();

this.balanceTable = balanceTable;

this.nextBalanceTable = nextBalanceTable;

double diff = getDiff();

calculateValuesAfter(diff);

this.balanceTable = null;

this.nextBalanceTable = null;

}

private void log(String text) {

System.out.println(text);

}

private String format(String text) {

String formatString = text;

return formatString;

}

private String format(String text, double value) {

String formatString = String.format(text, value);

return formatString;

}

private String format(String text, double value1, double value2) {

String formatString = String.format(text, value1, value2);

return formatString;

}

private void saveResult(String state, String info) {

setState(state);

setInfo(info);

log(info);

}

public void calcWithCredit(Double credit, Double diff) {

if (Objects.equals(credit, diff)) {

setCredit(nextBalanceTable, null);

setDebit(nextBalanceTable, null);

saveResult("Недооплачено",

format("взято с кредита: %.2f, без остатка", diff));

} else if (credit > diff) {

credit -= diff;

setCredit(nextBalanceTable, credit);

setDebit(nextBalanceTable, null);

saveResult("Недооплачено",

format("взято с кредита: %.2f, остаток кредита: %.2f", diff, credit));

} else { //if (credit < diff)

diff -= credit;

setCredit(nextBalanceTable, null);

setDebit(nextBalanceTable, diff);

saveResult("Недооплачено",

format("взято с кредита %.2f, уйдет в дебет: %.2f", credit, diff));

}

}

public void calcWithDebit(Double debit, Double diff) {

if (Objects.equals(debit, diff)) {

setCredit(nextBalanceTable, null);

setDebit(nextBalanceTable, null);

saveResult("Переоплачено",

format("оплачен дебет: %.2f, без остатка", debit));

} else if (debit > diff) {

debit -= diff;

setCredit(nextBalanceTable, null);

setDebit(nextBalanceTable, debit);

saveResult("Переоплачено",

format("оплачен дебет: %.2f, остаток дебета %.2f", diff, debit));

} else { //if (debit < diff)

diff -= debit;

setCredit(nextBalanceTable, diff);

setDebit(nextBalanceTable, null);

saveResult("Переоплачено",

format("оплачено дебета %.2f, уйдет в кредит: %.2f", debit, diff));

}

}

private void calcIfZero(Double credit, Double debit) {

if (isExist(credit)) {

setDebit(nextBalanceTable, null);

setCredit(nextBalanceTable, credit);

saveResult("Оплачено",

format("остается кредит: %.2f", credit));

} else if (isExist(debit)) {

setDebit(nextBalanceTable, debit);

setCredit(nextBalanceTable, null);

saveResult("Оплачено",

format("остается дебет: %.2f", debit));

} else {

setDebit(nextBalanceTable, null);

setCredit(nextBalanceTable, null);

saveResult("Оплачено",

format("долгов нет"));

}

}

private void calcIfNotEnought(Double credit, Double debit, Double diff) {

if (isExist(credit)) {

calcWithCredit(credit, diff);

} else if (isExist(debit)) {

setDebit(nextBalanceTable, diff + debit);

setCredit(nextBalanceTable, null);

saveResult("Недооплачено",

format("текущий дебет: %.2f + новый %.2f", debit, diff));

} else {

setDebit(nextBalanceTable, diff);

setCredit(nextBalanceTable, null);

saveResult("Недооплачено",

format("уйдет в дебет %.2f", diff));

}

}

private void calcIfOverpaid(Double credit, Double debit, Double diff) {

diff = Math.abs(diff);

if (isExist(debit)) {

calcWithDebit(debit, diff);

} else if (isExist(credit)) {

setDebit(nextBalanceTable, null);

setCredit(nextBalanceTable, diff + credit);

saveResult("Переоплачено",

format("текущий кредит: %.2f + новый %.2f", credit, diff));

} else {

setCredit(nextBalanceTable, diff);

setDebit(nextBalanceTable, null);

saveResult("Переоплачено",

format("уйдет в кредит %.2f", diff));

}

}

private void calculateValuesAfter(Double diff) {

Double prevCredit = getCredit(balanceTable);

Double prevDebit = getDebit(balanceTable);

if (diff == 0.0) {

calcIfZero(prevCredit, prevDebit);

} else if (diff > 0) {

calcIfNotEnought(prevCredit, prevDebit, diff);

} else {

calcIfOverpaid(prevCredit, prevDebit, diff);

}

}

private StringProperty state;

private StringProperty info;

public String getState() {

return state.get();

}

public void setState(String state) {

this.state.set(state);

}

public String getInfo() {

return info.get();

}

public void setInfo(String info) {

this.info.set(info);

}

}

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc10018092)

[1. ОБЩИЕ ПРИНЦИПЫ ВЕДЕНИЯ УЧЁТА АРЕНДЫ 6](#_Toc10018093)

[1.1. Структура организации бухгалтерского учета аренды 6](#_Toc10018094)

[1.2. Информационные потребности пользователей 6](#_Toc10018095)

[1.3. Процесс учета аренды 7](#_Toc10018096)

[1.3.1. Описание ручного процесса 7](#_Toc10018097)

[1.3.2. Преимущества автоматизации 8](#_Toc10018098)

[1.4. Анализ существующих программных аналогов 8](#_Toc10018099)

[1.5. Преимущества разработки отдельной программы 11](#_Toc10018100)

[2. ТРЕБОВАНИЯ К ПРОГРАММЕ 13](#_Toc10018101)

[2.1. Входные данные 13](#_Toc10018102)

[2.2. Процессы обработки договоров аренды 13](#_Toc10018103)

[2.3. Выходные данные 14](#_Toc10018104)

[2.4. Функциональные возможности программы 14](#_Toc10018105)

[2.5. Подробное описание требований к функционалу 15](#_Toc10018106)

[3. ОБОСНОВАНИЕ ВЫБОРА ИНСТРУМЕНТОВ РАЗРАБОТКИ 17](#_Toc10018107)

[3.1. Целесообразность написания десктоп приложения 17](#_Toc10018108)

[3.2. Недостатки применения Microsoft Excel и макросов VBA 17](#_Toc10018109)

[3.3. Описание стека технологий 18](#_Toc10018110)

[3.3.1. Язык программирования Java 18](#_Toc10018111)

[3.3.2. Библиотека JavaFX 18](#_Toc10018112)

[3.3.3. Язык разметки FXML и конструктор JavaFX Scene Builder 19](#_Toc10018113)

[3.4. Выбор библиотеки Apache POI для работы с Microsoft Excel 19](#_Toc10018114)

[3.5. Аргументация в пользу системы управления базами данных h2 20](#_Toc10018115)

[4. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ И ПРИЛОЖЕНИЯ 21](#_Toc10018116)

[4.1. Проектирование базы данных 21](#_Toc10018117)

[4.1.1. Сущности базы данных 21](#_Toc10018118)

[4.1.2. Модель и архитектура базы данных 22](#_Toc10018119)

[4.1.3. Нормализация отношений 22](#_Toc10018120)

[4.1.4. Концептуальная модель базы данных 23](#_Toc10018121)

[4.1.5. Логическая модель базы данных 24](#_Toc10018122)

[4.1.6. Физическая модель базы данных 25](#_Toc10018123)

[4.2. Проектирование приложения 26](#_Toc10018124)

[4.2.1. Диаграмма вариантов использования 26](#_Toc10018125)

[4.2.2. Диаграмма видов деятельности 27](#_Toc10018126)

[4.2.3. Диаграмма состояний для счетов 28](#_Toc10018127)

[4.2.4. Диаграмма состояний для договоров 29](#_Toc10018128)

[4.2.5. Диаграмма развертывания 29](#_Toc10018129)

[5. РАЗРАБОТКА ПРОГРАММЫ 31](#_Toc10018130)

[5.1. Потоки данных 31](#_Toc10018131)

[5.2. Рабочий процесс пользователя и программы 31](#_Toc10018132)

[5.3. Диаграммы классов 38](#_Toc10018133)

[5.4. Классы и методы 41](#_Toc10018134)

[6. ОХРАНА ТРУДА 55](#_Toc10018135)

[6.1. Искусственное освещение производственных помещений 55](#_Toc10018136)

[6.2. Вентиляция производственных помещений. Её виды 57](#_Toc10018137)

[6.3. Основы пожарной безопасности 60](#_Toc10018138)

[ВЫВОД 64](#_Toc10018139)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 66](#_Toc10018140)

[ПРИЛОЖЕНИЯ 69](#_Toc10018141)

[Приложение А. Скрипт генерации базы данных 69](#_Toc10018142)

[Приложение Б. Точка входа в приложение 73](#_Toc10018143)

[Приложение В. Класс базы данных Database 75](#_Toc10018144)

[Приложение Г. Листинг базовых классов 76](#_Toc10018145)