**Содержание**

[Введение 2](#_Toc61254638)

[1. Постановка задачи 3](#_Toc61254639)

[2. Логическое проектирование структуры базы данных 4](#_Toc61254640)

[3. Физическое проектирование базы данных 8](#_Toc61254641)

[4. Архитектура приложения 11](#_Toc61254642)

[5. Документация по используемым функциям приложения 15](#_Toc61254643)

[6. Руководство пользователя 19](#_Toc61254644)

[6.1 Назначение приложения 19](#_Toc61254645)

[6.2 Условия выполнения приложения 19](#_Toc61254646)

[6.3 Выполнение приложения 19](#_Toc61254647)

[7. Результаты тестирования разработанного приложения 20](#_Toc61254648)

[Заключение 23](#_Toc61254649)

[Список использованных источников 24](#_Toc61254650)

# Введение

Эффективность функционирования предприятия или организации любой отрасли и сферы деятельности напрямую зависит от скорости, точности и своевременности обмена данными как внутри этого предприятия между его составляющими частями (отделами, подсистемами и т.д.), так и вне его, то есть взаимодействие и обмен данными этой организации с другими (конкурирующими, предприятиями-партнерами и т.д.). И чем больше, масштабнее предприятие, тем серьезнее перед его управляющими встает проблема организации и контроля потоков огромного количества информации предприятия.

Чтобы обеспечить взаимодействия человека с персональным компьютером в интерактивном режиме, стали разрабатываться автоматизированные информационные системы, которые является совокупностью аппаратных и программных средств, которые обеспечивают взаимодействия человека и компьютера, а обеспечивают следующими функциями: возможность ввода в ПК и возможность вывода информации, как на экран, так и на устройства вывода.

Целью данной курсовой работы является разработка приложения учета оплаты коммунальных услуг.

# Постановка задачи

Разработать приложение учета оплаты коммунальных услуг. Учет оплаты ведется по следующим критериям: дата оплаты, ФИО квартиросъемщика, наименование услуги, сумма оплаты, пеня.

Разработать и использовать собственную иерархию классов, расширение базовых классов, предоставляемых JDK.

Использовать сокрытие данных (инкапсуляция), перегрузку методов, переопределение методов, абстрактные типы данных (интерфейсы, абстрактные классы), статические методы, обработку исключительных ситуаций.

Доступ к данным в СУБД должен осуществляться через драйвер, предоставляемый производителем СУБД или через использование специальных технологий.

В разрабатываемом приложении обеспечить добавление, редактирование и удаление записей из базы данных, сохранение табличных результатов в файле (создание текстового отчета).

# Логическое проектирование структуры базы данных

Для логического проектирования структуры базы данных возможно использование одной из нескольких стратегий проектирования:

1. Нормализация схем отношений. Нормализация предусматривает идентификацию требуемых атрибутов с последующим созданием из них нормализованных таблиц, основанных на функциональных зависимостях между этими атрибутами. Процесс проектирования представляет собой процесс нормализации.
2. Концепция модели «сущность-связь». Начинается этот подход с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов.
3. Подход «от общего к частному» напоминает нормализацию, но отличается от него тем, что вначале выявляется набор основных сущностей с последующим расширением круга рассматриваемых сущностей, связей и атрибутов, которые взаимодействуют с первоначально определенными сущностями.
4. «Смешанная» стратегия проектирования предполагает использование первых двух стратегий для создания разных частей модели, после чего все подготовленные фрагменты собираются в единое целое.

Воспользуемся «смешанной» стратегией проектирования.

На первом этапе выделим сущности и опишем их реквизитный состав, на втором – сформируем отношения, уточним количество этих отношений и их атрибутный состав. Затем проверим отношения на соответствие 3 НФ.

* 1. Выделение сущностей и описание их реквизитного состава.

В задаче учета оплаты коммунальных услуг можно выделить следующие сущности:

Сущность «Услуга», которая характеризуется реквизитами: идентификатор сущности, наименование услуги.

Сущность «Квартиросъемщик», которая характеризуется реквизитами: идентификатор сущности, ФИО квартиросъемщика.

Сущность «Расчет», которая характеризуется реквизитами: идентификатор сущности, дата оплаты, идентификатор квартиросъемщика, идентификатор услуги, сумма оплаты, пеня.

* 1. Подготовка к применению ER-метода логического проектирования.

Использование данного метода возможно тогда, когда в результате выполнения концептуального проектирования БД уже выполнены следующие действия:

выделены все сущности, информация о которых должна содержаться в искомой БД;

определены основные атрибуты для каждой сущности;

назначен ключевой атрибут для каждой сущности;

сформулированы связи между выделенными сущностями.

Итак, для применения ER-метода назначим ключевые атрибуты для каждой сущности и сформулируем связи между выделенными сущностями:

*Услуга (****КодУслуги,***Наименование*);*

*Квартиросъемщик* (***КодКвартиросъемщика,*** ФИО квартиросъемщика);

*Расчет* (***КодРасчета,*** Дата оплаты, КодКвартиросъемщика, КодУслуги, Сумма оплаты, Пеня);

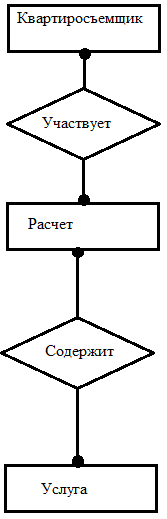
Сущности *«Квартиросъемщик»* и *«Расчет»* соотносятся с помощью *связи* *Платит*.

Сущность «*Расчет»* и сущность «*Услуга*» соотносятся с помощью *связи* *Содержит*.

* 1. Использование ER-метода логического проектирования.

С помощью указанного метода логического проектирования уточним количество таблиц-отношений и их атрибуты.

Построим ER-диаграмму для проектируемой задачи (рис. 2.1).



**Рисунок 2.1 - Диаграмма ER-типа проектируемой базы**

На основании правила 4 генерации отношений, связь *Оплачивает* порождает два отношения по одному для каждой сущности, причем ключевой атрибут *КодКвартиросъемщика* сущности *Квартиросъемщик* должен быть включен в число атрибутов отношения *Расчет*. Получаем следующие отношения:

*Расчет* (***КодСтудента****, Фамилия, Имя, Отчество, Код Группы*);

*Квартиросъемщик* (***КодКвартиросъемщика****, ФИО квартиросъемщика*).

На основании правила 4 генерации отношений связь *Относится*  порождает два отношения по одному для каждой сущности, причем ключевой атрибут *КодФакультета* сущности *Факультет* должен быть включен в число атрибутов отношения *Группа.*

На основании того же правила, связь *Содержит* порождает два отношения для сущностей Услуга и Расчет. Причем ключевой атрибут *КодУслуги* сущности *Услуга* должен быть включен в число атрибутов отношения *Услуга.*

Таким образом,получаем следующие отношения и таблицы:

*Услуга (****КодУслуги,***Наименование*);*

*Квартиросъемщик* (***КодКвартиросъемщика,*** ФИО квартиросъемщика);

*Расчет* (***КодРасчета,*** Дата оплаты, КодКвартиросъемщика, КодУслуги, Сумма оплаты, Пеня);

У таблиц-отношений *Услуга, Квартиросъемщик, Расчет*первичные ключипростые (выделены полужирным шрифтом), они состоят из одногополя.

Все получившиеся таблицы удовлетворяют требованиям 3НФ. Эта степень нормализации вполне достаточна для разрабатываемой базы данных, поэтому будем считать, что процесс проектирования заданной БД завершен.

# 3. Физическое проектирование базы данных

3.1 Опишем структуру таблиц, полученных в ходе логического проектирования, в применении к конкретной СУБД (СУБД MySQL).

Таблица 3.1 – Структура таблиц базы данных

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя таблицы** | **Столбец** | **Описание столбца** | **Комментарий** |
| **tenant** | id | integer, unsigned, AUTO\_INCREMENT,  PRIMARY KEY | код квартиросъемщика, счетчик, первичный ключ |
|  | Fio | varchar(200) | ФИО квартиросъемщика, текстовый |
| **service** | id | integer, unsigned, AUTO\_INCREMENT,  PRIMARY KEY | код услуги, счетчик, первичный ключ |
|  | name | varchar(150) | наименование услуги, текстовый |
| **payment** | id | integer, unsigned, AUTO\_INCREMENT,  PRIMARY KEY | код оплаты, счетчик, первичный ключ |
|  | Dates | Date | дата оплаты |
|  | Tenant | Integer, FOREIGN KEY | идентификатор квартиросъемщика, числовой |
|  | Service | Integer, FOREIGN KEY | идентификатор услуги, числовой |
|  | Sum | Float | сумма оплаты, числовой с точкой |
|  | fine | Float | пеня, числовой с точкой |

**3.2. Создание схемы базы данных payment\_of\_utilities в среде mySQL Workbench**

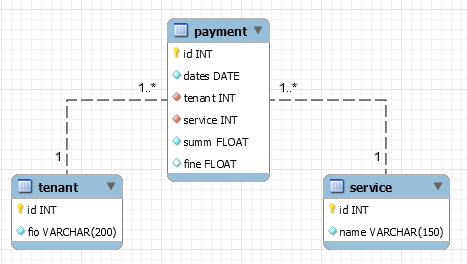
Для создания базы данных payment\_of\_utilities в mySQL Workbench возможно использование одного из двух подходов: прямое проектирование (FORVARD INGINEER) и обратное (REVERSE INGINEER).

В первом случае процесс получения структуры базы данных для выбранной СУБД осуществляется на основе построенной ER-модели.

Другими словами, вначале создается графическая модель базы данных и на ее основе строится физическая модель базы данных: структура таблиц и связи между ними.

Во втором случае – выполняется создание ER-модели на основе уже созданной базы данных. Т.е. вначале описывается структура таблиц и связи, а затем создается графическая модель базы данных.

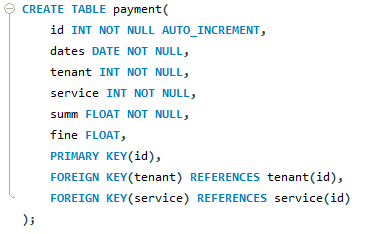
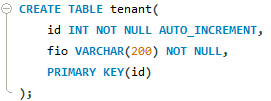
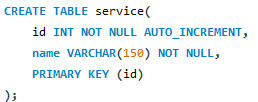
Графическая модель базы данных payment\_of\_utilities представлена на рисунке 3.1.



**Рисунок 3.1 – Графическое представление базы данных в нотации UML**

**3.3. Описание таблиц базы данных payment\_of\_utilities**

Структура таблиц базы данных payment\_of\_utilities представлена на рисунке 3.2.



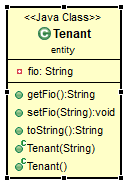
**Рисунок 3.2 – Описание структуры таблиц базы данных** **studentspass**

Содержание файла SQL-запроса на создание таблиц базы данных studentpass представлено в Приложении В (файл database.sql).

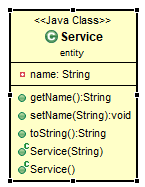
# Архитектура приложения

Каркас приложения выполнен в архитектуре MVC.

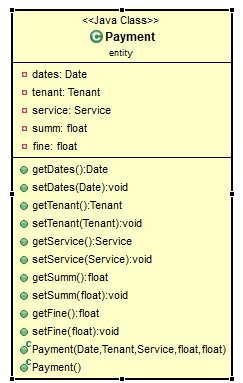
Модель данных представлена в виде набора классов (рисунки 4.1-4.4).



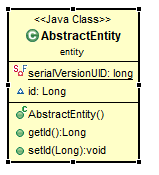
**Рисунок 4.1 – Класс Tenant**



**Рисунок 4.2 – Класс Service**

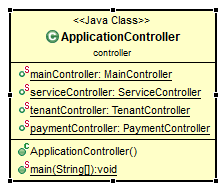


**Рисунок 4.3 – Класс Payment**



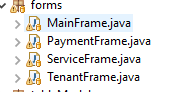
**Рисунок 4.4 – Класс AbstractEntity**

В качестве контроллера всего приложения выступает класс-контроллер ApplicationController, который осуществляет взаимодействие между моделью и представлением и содержит ссылки на контроллеры каждого вида(рисунок 4.5).



**Рисунок 4.5 – Класс ApplicationController**

Представление данных реализовано в виде набора JFrame форм (рисунок 4.6).



**Рисунок 4.6 – Список JFrame форм**

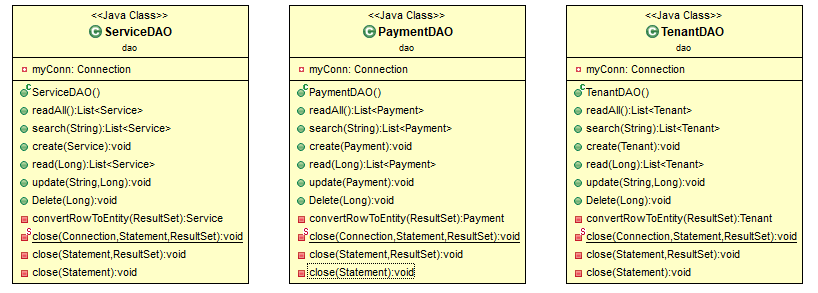
В паттерне «модель – представление – контроллер» модель представляет данные приложения и связанную с ними бизнес-логику.

Данные передаются в БД и из нее в объектах передачи данных, и к ним обращаются с помощью объектов доступа к данным.

Представление – это наглядное отображение содержащихся в модели данных. Подмножество модели содержится в отдельном представлении, таким образом, представление действует в качества фильтра для данных модели. Пользователь взаимодействует с данными модели с помощью предлагаемого представлением наглядного отображения и обращается к бизнес логике, которая, в свою очередь, воздействует на данные модели.

Контроллер связывает представление с моделью и управляет потоками данных приложения. Он выбирает, какое представление визуализировать для пользователя в ответ на вводимые им данные и в соответствии с выполняемой бизнес-логикой. Контроллер получает сообщение от представления и пересылает его модели. Модель, в свою очередь, подготавливает ответ и отправляет его обратно контроллеру, где происходит выбор представления и отправка его пользователю.

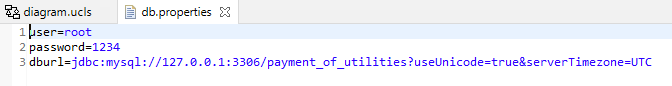
Доступ к данным выделен в отдельный слой приложения (рисунок 4.7). Мы используем паттерн Data Access Object (DAO) для абстрагирования и инкапсулирования доступа к источнику данных. DAO управляет соединением с источником данных для получения и записи данных.



**Рисунок 4.7 – Классы для доступа к данным**

Доступ к данным в СУБД осуществляться через драйвер JDBC jdbc:mysql://localhost:3306/ payment\_of\_utilities.

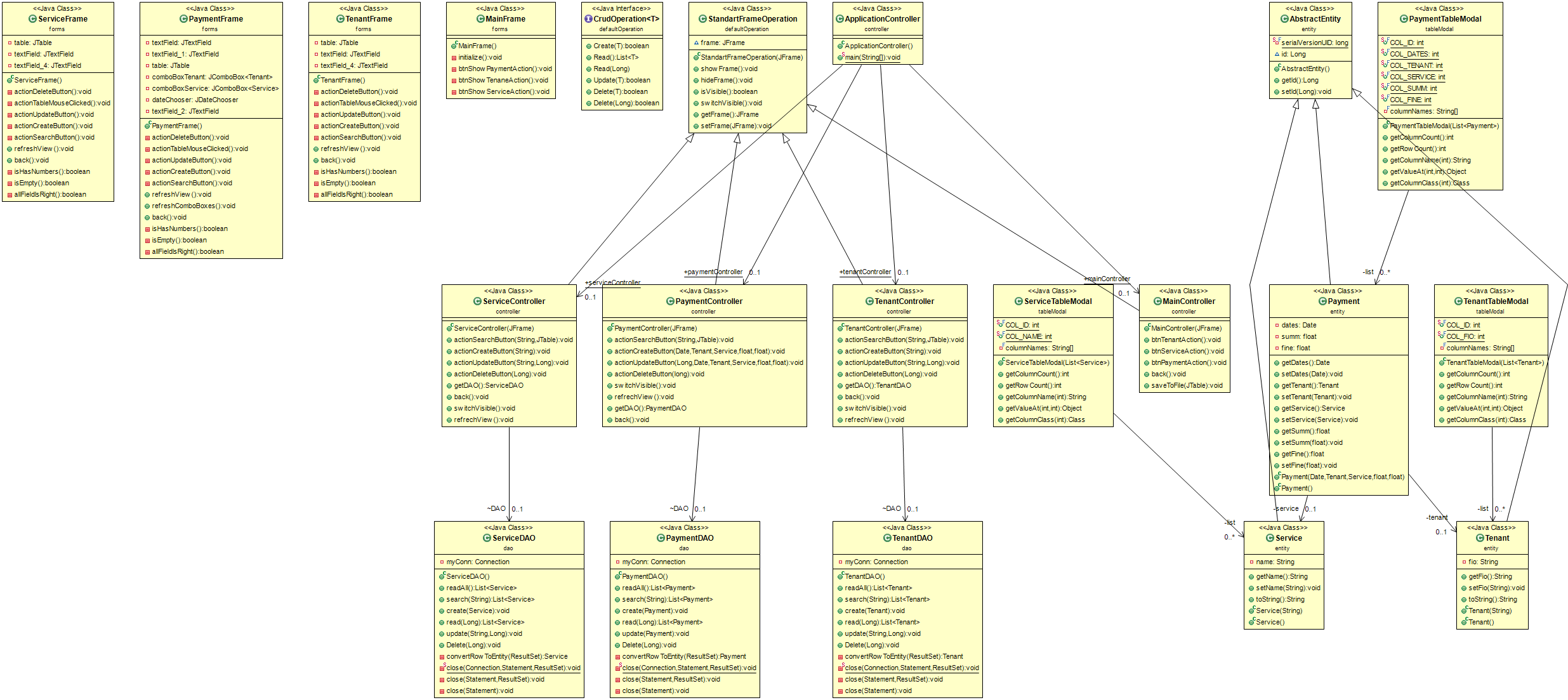
Настройки подключения к базе данных меняются без изменения исходного кода, используется properties-файл (рисунок 4.8).



**Рисунок 4.8 – Файл db.properties**

В разрабатываемом приложении обеспечена возможность добавления, редактирования и удаления записей из базы данных, сохранение табличных результатов в txt-файле.

Структура проекта представлена на рисунке ниже (рисунок 4.9).



**Рисунок 4.9 – Структура проекта**

# 5. Документация по используемым функциям приложения

Опишем методы класса Tenant:

* fio : String – переменная хранения ФИО квартиросъемщика
* getFio() – метод получения ФИО
* setFio(String) – метод установки ФИО
* toString() – метод перевода сущности в строку
* Tenant(String) – конструктор с заданием ФИО
* Tenant() – конструктор без полей

Опишем методы класса Service:

* name : String – переменная хранения наименования
* getName() – метод получения наименования
* setName(String) – метод установки наименования
* toString() – метод перевода сущности в строку
* Service(String) – конструктор с заданием наименования
* Service() – конструктор без полей

Опишем методы класса Payment:

* dates : Date – переменная хранения даты оплаты
* tenant : Tenant – переменная хранения квартиросъемщика
* service : Service – переменная ранения услуги
* summ : float – переменная хранения суммы оплаты
* fine : float – переменная хранения пени
* getDates() - метод поолучения переменной
* setDates(Date) – метод установки переменной
* getTenant()- метод поолучения переменной
* setTenant(Tenant) – метод установки переменной
* getService()- метод поолучения переменной
* setService(Service) – метод установки переменной
* getSumm()- метод поолучения переменной
* setSumm(float) – метод установки переменной
* getFine()- метод поолучения переменной
* setFine(float) – метод установки переменной
* Payment(Date, Tenant, Service, float, float) – конструктор с заполнением полей
* Payment() – конструктор без полей

Листинг 5.1 – Описание сущности Расчет

**package** entity;

**import** java.sql.Date;

/\*\*

\* Сущность Учета

\*

\* **@author** ya

\*

\*/

**public** **class** Payment **extends** AbstractEntity{

/\*\*

\* Дата оплаты

\*/

**private** Date dates;

/\*\*

\* Квартиросъемщик

\*/

**private** Tenant tenant;

/\*\*

\* Услуга

\*/

**private** Service service;

/\*\*

\* Сумма оплаты

\*/

**private** **float** summ;

/\*\*

\* Пеня

\*/

**private** **float** fine;

/\*\*

\* Получение поля

\* **@return** dates

\*/

**public** Date getDates() {

**return** dates;

}

/\*\*

\* Установка поля

\* **@param** dates

\*/

**public** **void** setDates(Date dates) {

**this**.dates = dates;

}

/\*\*

\* Получение поля

\* **@return** dates

\*/

**public** Tenant getTenant() {

**return** tenant;

}

/\*\*

\* Установка поля

\* **@param** tenant

\*/

**public** **void** setTenant(Tenant tenant) {

**this**.tenant = tenant;

}

/\*\*

\* Получение поля

\* **@return** service

\*/

**public** Service getService() {

**return** service;

}

/\*\*

\* Установка поля

\* **@param** service

\*/

**public** **void** setService(Service service) {

**this**.service = service;

}

/\*\*

\* Получение поля

\* **@return** summ

\*/

**public** **float** getSumm() {

**return** summ;

}

/\*\*

\* Установка поля

\* **@param** summ

\*/

**public** **void** setSumm(**float** summ) {

**this**.summ = summ;

}

/\*\*

\* Получение поля

\* **@return** fine

\*/

**public** **float** getFine() {

**return** fine;

}

/\*\*

\* Установка поля

\* **@param** fine

\*/

**public** **void** setFine(**float** fine) {

**this**.fine = fine;

}

/\*\*

\* Конструктор с параметром

\*

\* **@param** fio

\*/

**public** Payment(Date dates, Tenant tenant, Service service, **float** summ, **float** fine) {

**super**();

**this**.dates = dates;

**this**.tenant = tenant;

**this**.service = service;

**this**.summ = summ;

**this**.fine = fine;

}

/\*\*

\* Конструктор без параметров

\*/

**public** Payment() {

**super**();

}

}

# Руководство пользователя

## 6.1 Назначение приложения

Данное приложение предназначено для автоматизации ведения учета оплаты коммунальных услуг. Пользователям доступны операции по добавлению, изменению, удалению данных об услугах, квартиросъемщиках, оплате.

## 6.2 Условия выполнения приложения

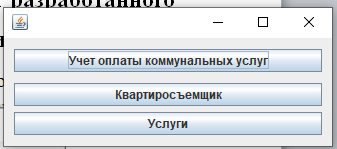
Исполняемый файл runnable.jar занимает 2.8 Мб памяти и может выполняться на любом персональном компьютере, на котором установлено ПО Java 8+, MySQL 5.5+.

## 6.3 Выполнение приложения

Для получения доступа к приложению требуется запустить файл runnable.jar и дождаться появления главного окна приложения.

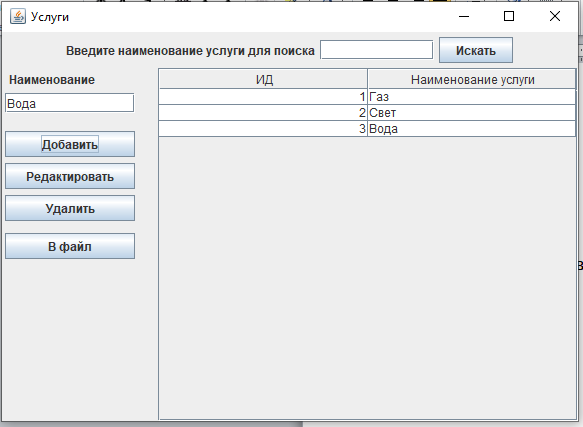
# Результаты тестирования разработанного приложения

При входе в приложения мы видим следующую форму



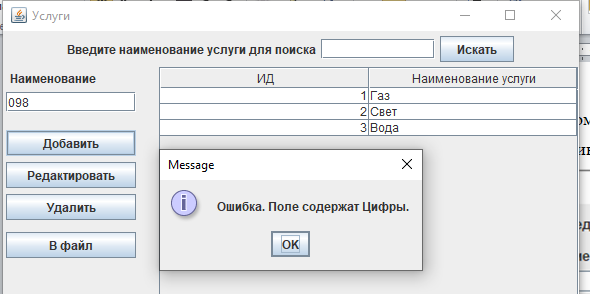
**Рисунок 6.1 – Главная форма**

Первым делом стоит заполнить услуги. Для этого перейдем с помощью последней кнопки в справочник услуг.



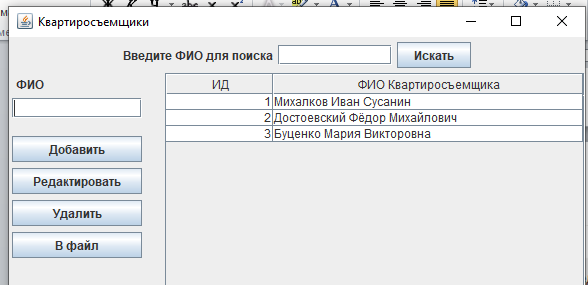
**Рисунок 6.2 – Услуги**

На этой форме можно выполнить основные операции с данными таблицы. При неправильном заполнении поля для добавления, приложение выдаст соответствующую ошибку.



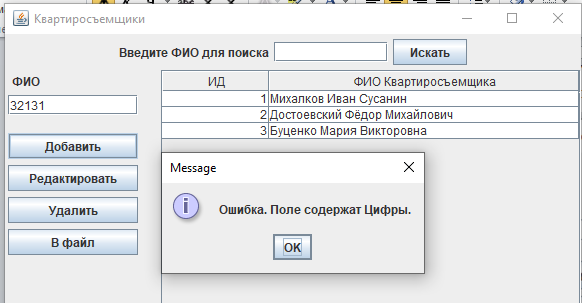
**Рисунок 6.3 – Ошибка при неправильно заполненных полях**

Можно также перейти на страницу квартиросъемщиков, где работа уже пойдет непосредственно с ними.



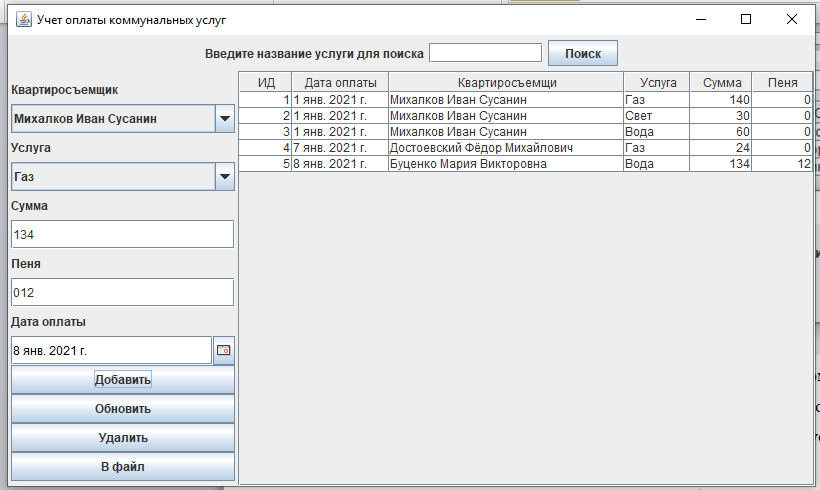
**Рисунок 6.4 – Квартиросъемщик**

На этой странице также идет проверка правильности заполнения поля ФИО квартиросъемщика. При вводе цифр, приложение выведет соответствующую ошибку.



**Рисунокк 6.5 – Ошибка при неправильном заполнении поля**

Последняя работа с формой учета оплаты, на которой работаем сразу с тремя таблицами: квартиросъемщики (выпадающший список), услуги (выпадающий список), расчет.



**Рисунок 6.6 – Форма учета оплаты коммунальных услуг**

# Заключение

В ходе выполнения данной курсовой работы было разработано приложение по учету оплаты коммунальных услуг, закреплены теоретические знания и отработаны практические навыки объектно-ориентированного проектирования и программирования на языке Java, изучена литература по объектно-ориентированному программированию, проведено обучение работе в интегрированной среде программирования IDE Eclipse.

В рамках разработки приложения было проведено логическое проектирование структуры базы данных, физическое проектирование базы данных. Была рассмотрена архитектура MVC.

В практической части работы было разработано приложение для учёта оплаты коммунальных услуг при помощи языка программирования высокого уровня Java.

# Список использованных источников

1. Коваленко, Е.А. Базы и банки данных: практическое пособие для студентов специальности I – 53 01 02 «Автоматизированные системы обработки информации» / Е.А.Коваленко, В.Н.Леванцов. – Гомель: учреждение образования «Гомельский государственный университет имени Франциска Скорины», 2007. – 95 с.

2. Лекция №4 – Проектирование баз данных Подходы к проектированию базы данных // StudFiles [Электронный ресурс]. – 2019. – Режим доступа: https://studfile.net/preview/1705211/. – Дата доступа: 31.12.2020.

3. MySQL/Руководство для начинающих [Электронный ресурс]. – Режим доступа: https://wiki.gentoo.org/wiki/MySQL/Startup\_Guide/ru. – Дата доступа: 31.12.2020.

4. Руководство по MVC [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/mvc5/. – Дата доступа: 31.12.2020.

ПРИЛОЖЕНИЕ А

**Правила генерации отношений по диаграммам ER-типа**

**ПРАВИЛО 1.** Если показатель кардинальности бинарной связи равен **1:1** и класс принадлежности обеих сущностей является обязательным, то требуется только **одно** отношение. Первичным ключом этого отношения может быть ключ любой из двух сущностей.

**ПРАВИЛО 2.** Если показатель кардинальности бинарной связи равен 1:1 и класс принадлежности одной сущности является обязательным, а другой — необязательным, то необходимо построение **двух** отношений. Под каждую сущность необходимо выделение одного отношения, при этом ключ сущности должен служить первичным ключом для соответствующего отношения. Кроме того, ключ сущности, для которой класс принадлежности является необязательным, добавляется в качестве атрибута в отношение, выделенное для сущности с обязательным классом принадлежности.

**ПРАВИЛО 3.** Если показатель кардинальности бинарной связи равен **1:1** и класс принадлежности ни одной сущности не является обязательным, то необходимо использовать **три** отношения: по одному для каждой сущности, ключи которых служат в качестве первичных в соответствующих отношениях, и одного для связи. Среди своих атрибутов отношение, выделяемое связи, будет иметь по одному ключу сущности от каждой сущности.

**ПРАВИЛО 4.** Если показатель кардинальности бинарной связи равен **1:n** и класс принадлежности n-связной сущности является обязательным, то достаточным является использование **двух** отношений, по одному на каждую сущность, при условии, что ключ сущности каждой сущности служит в качестве первичного ключа для соответствующего отношения. Дополнительно ключ 1-связной сущности должен быть добавлен как атрибут в отношение, отводимое n-связной сущности.

**ПРАВИЛО 5.** Если показатель кардинальности бинарной связи равен **1:n** и класс принадлежности n‑связной сущности является необязательным, то необходимо формирование **трех** отношений: по одному для каждой сущности, причем ключ каждой сущности служит первичным ключом соответствующего отношения, и одного отношения для связи. Связь должна иметь среди своих атрибутов ключ сущности от каждой сущности.

**ПРАВИЛО 6.** Если показатель кардинальности бинарной связи равен **m:n**, то для хранения данных необходимо **три** отношения: по одному для каждой сущности, причем ключ каждой сущности используется в качестве первичного ключа соответствующего отношения, и одного отношения для связи. Последнее отношение должно иметь в числе своих атрибутов ключ сущности каждой сущности.

ПРИЛОЖЕНИЕ Б

В теории реляционных баз данных обычно выделяется следующая **последовательность нормальных форм:**

* первая нормальная форма (1NF);
* вторая нормальная форма (2NF);
* третья нормальная форма (3NF);
* нормальная форма Бойса-Кодда (BCNF);
* четвертая нормальная форма (4NF);
* пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

**1НФ - таблица находится в первой нормальной форме (1НФ) тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто.** (Любое поле таблицы содержит неделимую информацию и в таблице определен первичный ключ).

**2НФ - Таблица находится во второй нормальной форме (2NF) в том и только в том** случае**, когда находится в 1NF, и каждый ее неключевой атрибут полностью зависит от первичного ключа.**

Если таблица в 1 НФ имеет простой первичный ключ, то она автоматически находится и во 2 НФ.

**3НФ – Таблица находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый неключевой атрибут** нетранзитивно **зависит от первичного ключа.** (Иными словами, таблица должна находиться во второй нормальной форме и ни одно из ее неключевых полей не должно однозначно идентифицироваться значением другого неключевого поля (полей)).

Теоретики реляционных систем Кодд и Бойс обосновали и предложили более строгое определение для 3НФ, которое учитывает, что в таблице может быть несколько возможных ключей. Таблица, соответствующая этому определению называется таблицей *в улучшенной третьей форме* или таблицей в *нормальной форме Бойса-Кодда*.

Таблица находится в **нормальной форме Бойса-Кодда** (НФБК), если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от возможного ключа.

В следующих нормальных формах (4НФ и 5НФ) учитываются не только функциональные, но и многозначные зависимости между полями таблицы. Для их описания познакомимся с понятием полной декомпозиции таблицы.

**Полной декомпозицией** таблицы называют такую совокупность произвольного числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Теперь можно дать определения высших нормальных форм. И сначала будет дано определение для последней из предложенных - 5НФ.

**5НФ - Таблица находится в пятой нормальной форме тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ.** Таблица, не имеющая ни одной полной декомпозиции, также находится в 5НФ.

**4НФ - Четвертая нормальная форма является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций.** Весьма не просто подобрать реальную таблицу, которая находилась бы в 4НФ, но не была бы в 5НФ.

Третья нормальная форма считается оптимальной для небольших баз данных, при проведении дальнейшей нормализации следует учитывать, что при увеличении количества связанных таблиц увеличивается время обработки информации, хранящейся в них. Поэтому разработчик должен сам принимать решение о том, какая ступень нормализации будет оптимальной для проектируемой базы данных.

ПРИЛОЖЕНИЕ В

Содержание файла database.sql

DROP DATABASE IF EXISTS payment\_of\_utilities;

CREATE DATABASE payment\_of\_utilities;

USE payment\_of\_utilities;

CREATE TABLE service(

id INT NOT NULL AUTO\_INCREMENT,

name VARCHAR(150) NOT NULL,

PRIMARY KEY (id)

);

CREATE TABLE tenant(

id INT NOT NULL AUTO\_INCREMENT,

fio VARCHAR(200) NOT NULL,

PRIMARY KEY(id)

);

CREATE TABLE payment(

id INT NOT NULL AUTO\_INCREMENT,

dates DATE NOT NULL,

tenant INT NOT NULL,

service INT NOT NULL,

summ FLOAT NOT NULL,

fine FLOAT,

PRIMARY KEY(id),

FOREIGN KEY(tenant) REFERENCES tenant(id),

FOREIGN KEY(service) REFERENCES service(id));

ПРИЛОЖЕНИЕ Г

**Содержание файла Payment.java**

**package** entity;

**import** java.sql.Date;

/\*\*

\* Сущность Учета

\*

\* **@author** ya

\*

\*/

**public** **class** Payment **extends** AbstractEntity{

/\*\*

\* Дата оплаты

\*/

**private** Date dates;

/\*\*

\* Квартиросъемщик

\*/

**private** Tenant tenant;

/\*\*

\* Услуга

\*/

**private** Service service;

/\*\*

\* Сумма оплаты

\*/

**private** **float** summ;

/\*\*

\* Пеня

\*/

**private** **float** fine;

/\*\*

\* Получение поля

\* **@return** dates

\*/

**public** Date getDates() {

**return** dates;

}

/\*\*

\* Установка поля

\* **@param** dates

\*/

**public** **void** setDates(Date dates) {

**this**.dates = dates;

}

/\*\*

\* Получение поля

\* **@return** dates

\*/

**public** Tenant getTenant() {

**return** tenant;

}

/\*\*

\* Установка поля

\* **@param** tenant

\*/

**public** **void** setTenant(Tenant tenant) {

**this**.tenant = tenant;

}

/\*\*

\* Получение поля

\* **@return** service

\*/

**public** Service getService() {

**return** service;

}

/\*\*

\* Установка поля

\* **@param** service

\*/

**public** **void** setService(Service service) {

**this**.service = service;

}

/\*\*

\* Получение поля

\* **@return** summ

\*/

**public** **float** getSumm() {

**return** summ;

}

/\*\*

\* Установка поля

\* **@param** summ

\*/

**public** **void** setSumm(**float** summ) {

**this**.summ = summ;

}

/\*\*

\* Получение поля

\* **@return** fine

\*/

**public** **float** getFine() {

**return** fine;

}

/\*\*

\* Установка поля

\* **@param** fine

\*/

**public** **void** setFine(**float** fine) {

**this**.fine = fine;

}

/\*\*

\* Конструктор с параметром

\*

\* **@param** fio

\*/

**public** Payment(Date dates, Tenant tenant, Service service, **float** summ, **float** fine) {

**super**();

**this**.dates = dates;

**this**.tenant = tenant;

**this**.service = service;

**this**.summ = summ;

**this**.fine = fine;

}

/\*\*

\* Конструктор без параметров

\*/

**public** Payment() { **super**(); }}