Министерство образования Республики Беларусь

Учреждение образования

«Гомельский государственный университет   
имени Франциска Скорины»

Заочный факультет

Кафедра автоматизированных систем обработки информации

Разработка системы учета показаний счетчиков и оплаты счетов за коммунальные услуги

Курсовой проект

по дисциплине «Шаблоны проектирования»

Исполнитель

студент группы АСз-41  Павлючков А.В.

Руководитель Грищенко В.В.

Гомель 2018

Содержание

[Введение 3](#_Toc502850311)

[1 Обзор шаблонов проектирования и их классификация 4](#_Toc502850312)

[1.1 Поведенческие шаблоны 5](#_Toc502850313)

[1.2 Порождающие шаблоны 6](#_Toc502850314)

[1.3 Структурные шаблоны 8](#_Toc502850315)

[2 Разработка структуры проекта 10](#_Toc502850316)

[2.1 Требования к системе и логика ее работы 10](#_Toc502850317)

[2.2 Выбор шаблонов проектирования 11](#_Toc502850318)

[2.3 Структура проекта 15](#_Toc502850319)

[3 Реализация приложения 17](#_Toc502850320)

[3.1 Обзор программных средств и применяемых технологий 17](#_Toc502850321)

[3.2 Этапы реализации приложения 18](#_Toc502850322)

[3.3 Описание основных классов и методов проекта 22](#_Toc502850323)

[Заключение 25](#_Toc502850324)

[Список использованных источников 26](#_Toc502850325)

[Приложение А 27](#_Toc502850326)

# Введение

Объектом и предметом исследования являются наиболее распространенные шаблоны проектирования.

Целью курсового проекта является приобретение навыков проектирования и решения возникающих проблем программирования, используя шаблоны проектирования.

Курсовой проект разработан на языке программирования Java с использованием платформы JavaFX и среды разработки Intellij Idea.

Задачи курсового проекта:

* Разработать алгоритм работы и необходимый функционал системы;
* Спроектировать и реализовать макет пользовательских форм;
* Выбрать шаблоны проектирования, максимально облегчающие реализацию логики работы системы;
* Реализовать логику работы системы;
* Протестировать программный продукт;
* Исправить и отладить программный продукт.
* Проанализировать результаты работы системы.

# Обзор шаблонов проектирования и их классификация

В мире постоянно кто-то сталкивается с такими же проблемами программирования, которые возникают и у нас. Многие разработчики решают совершенно идентичные задачи и находят похожие решения. Шаблоны появились, потому что многие разработчики искали пути повышения гибкости и степени повторного использования своих программ. Найденные решения воплощены в краткой и легко применяемой на практике форме.

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения – повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код – это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и поэтому не универсальные.

На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы.

Основная польза от использования шаблонов состоит в снижении сложности разработки за счет готовых абстракций. Шаблон дает решению свое имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова.

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, с применением могут быть связанны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Все шаблоны проектирования бывают следующих трех видов:

* Поведенческие (behavioral);
* Порождающие (creational);
* Структурные (structural);

## Поведенческие шаблоны

Поведенческие шаблоны (Behavioral) определяют взаимодействие между объектами, увеличивая таким образом их гибкость. Они связаны с распределением обязанностей между объектами. Их отличие от структурных шаблонов заключается в том, что они не просто описывают структуру, но также описывают шаблоны для передачи сообщений (связи между ними). Другими словами, они помогают ответить на вопрос «Как запустить поведение в программном компоненте?».

К поведенческим шаблонам относятся:

* Цепочка обязанностей (Chain of Responsibility);
* Команда (Command);
* Итератор (Iterator);
* Посредник (Mediator);
* Хранитель (Memento);
* Наблюдатель (Observer);
* Посетитель (Visitor);
* Стратегия (Strategy);
* Состояние (State);
* Шаблонный метод (Template Method);

Цепочка обязанностей (Chain of Responsibility) – шаблон проектирования, предназначенный для организации в системе уровней ответственности. Избегает связывания отправителя запроса с его получателем, давая возможность обработать запрос более чем одному объекту. Связывает объекты получатели и передает запрос по цепочке пока объект не обработает его.

Команда (Command) – шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себя само действие и его параметры. Инкапсулирует запросы в виде объекта, позволяя передавать их клиентам в качестве параметров, ставить в очередь, логировать, а также поддерживает отмену операций. Шаблон может быть использован для реализации системы, основанной на транзакциях, где вы сохраняете историю команд, как только их выполняете. Если окончательная команда выполнена успешно, то все хорошо, иначе алгоритм перебирает историю и продолжает отмену для всех выполненных команд.

Итератор (Iterator) – шаблон проектирования, представляющий собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из агрегированных объектов. Простыми словами – представляет способ последовательного доступа к элементам множества, независимо от его внутреннего устройства.

Посредник (Mediator) – шаблон проектирования, обеспечивающий взаимодействие множества объектов, формируя при этом слабую связанность, и избавляя объекты, от необходимости явно ссылаться друг на друга и дает возможность независимо изменять их взаимодействие.

Хранитель (Memento) – шаблон проектирования, позволяющий, не нарушая инкапсуляцию, зафиксировать и сохранить внутреннее состояние объекта так, чтобы позднее восстановить его в этом состоянии.

Наблюдатель (Observer) – шаблон проектирования, создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними. Т.е. шаблон определяет зависимость между объектами, чтобы при изменении состояния одного из них зависимые от него узнали об этом.

Посетитель (Visitor) – шаблон проектирования, описывающий операцию, которая выполняется над объектами других классов. При изменении visitor нет необходимости изменять обслуживаемые классы. Т.е. шаблон посетитель позволяет добавлять будущие операции для объектов без их модифицирования.

Стратегия (Strategy) – шаблон, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путем определения соответствующего класса. Позволяет менять выбранный алгоритм независимо от объектов – клиентов, которые его используют.

Состояние (State) – шаблон проектирования. Используется в тех случаях, когда во время выполнения программы объект должен менять свое поведение в зависимости от своего состояния.

Шаблонный метод (Template Method) – шаблон проектирования, определяющий основу алгоритма и позволяющий наследникам переопределять некоторые шаги алгоритма, не изменяя его структуру в целом.

## Порождающие шаблоны

Порождающие шаблоны – шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять наследуемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту.

Существуют следующие порождающие шаблоны:

* Простая фабрика (Simple Factory);
* Фабричный метод (Factory Method);
* Абстрактная фабрика (Abstract Factory);
* Строитель (Builder);
* Прототип (Prototype);
* Одиночка (Singleton).

Простая фабрика, в ООП, - это объект для создания других объектов. Формально фабрика – это функция или метод, который возвращает объекты изменяющегося прототипа или класса из некоторого вызова метода, который считается новым. Т.е. простая фабрика генерирует экземпляр для клиента, не раскрывая никакой логики.

Фабричный метод – шаблон проектирования, представляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне. Полезен, когда есть некоторая общая обработка в классе, но необходимый подкласс динамически определяется во время выполнения (когда клиент не знает, какой именно подкласс ему может понадобиться).

Абстрактная фабрика – шаблон проектирования, представляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфируя их конкретных классов. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы. Затем пишутся классы, реализующие этот интерфейс. Используется, когда есть взаимосвязанные зависимости с не очень простой логикой создания.

Строитель (Builder) – шаблон, который представляет способ создания составного объекта. Используется тогда, когда может быть несколько видов объекта и нужно избежать «телескопического конструктора». Главное отличие от «фабрики» - это то, что «фабрика» используется, когда создание занимает один шаг, а «строитель» применяется при множестве шагов.

Прототип (Prototype) – шаблон проектирования, который задает виды создаваемых объектов с помощью экземпляра-прототипа и создает новые объекты путем копирования этого прототипа. Он позволяет уйти от реализации и позволяет следовать принципу «программирование через интерфейсы». В качестве возвращаемого типа указывается интерфейс или абстрактный класс на вершине иерархии, а классы наследники могут подставить туда наследника, реализующего этот тип. Используется тогда, когда необходим объект, похожий на существующий объект, либо, когда создание будет дороже клонирования.

Одиночка (Singleton) – шаблон проектирования, гарантирующий, что в одно процессном приложении будет единственный экземпляр некоторого класса, и представляющий глобальную точку доступа к этому экземпляру. Т.е. гарантирует, что создаваемый объект является единственным объектом своего класса.

## Структурные шаблоны

Структурные шаблоны – шаблоны проектирования, в которых рассматривается вопрос о том, как из классов и объектов образуются более крупные структуры.

К структурным шаблонам проектирования относятся:

* Адаптер (Adapter);
* Мост (Bridge);
* Компоновщик (Composite);
* Декоратор (Decorator);
* Фасад (Facade);
* Приспособленец (Flyweight);
* Заместитель (Proxy);

Адаптер (Adapter) – шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс. Другими словами, шаблон позволяет обернуть несовместимые объекты в адаптер, чтобы сделать их совместимыми с другим классом.

Мост (Bridge) – структурный шаблон, используемый в проектировании программного обеспечения чтобы разделять абстракцию и реализацию так, чтобы они могли изменяться независимо. Шаблон использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами. Шаблон «мост» - это предпочтение композиции над наследованием. Детали реализации передаются из одной иерархии в другой объект с отдельной иерархией.

Компоновщик (Composite) – шаблон, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому. Позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково. Шаблон определяет иерархию классов, которые одновременно могут состоять из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.

Декоратор (Decorator) – структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Представляет собой гибкую альтернативу практике создания подклассов с целью расширения функциональности.

Фасад (Facade) – шаблон, позволяющий скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы. Т.е. шаблон «фасад» представляет упрощенный интерфейс для сложной системы.

Приспособленец (Flyweight) – шаблон, при котором объект, представляющий себя как уникальный экземпляр в разных местах программы, по факту не является таковым. Используется для минимизации использования памяти или вычислительной стоимости путем разделения ресурсов с наибольшим количеством похожих объектов.

Заместитель (Proxy) – шаблон, который представляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера). Используя шаблон «заместитель», класс отображает функциональность другого класса.

# Разработка структуры проекта

## Требования к системе и логика ее работы

Для начала определим некоторые требования, предъявляемые к нашей системе учета.

Разрабатываемая система учета показаний счетчиков и оплаты счетов за коммунальные услуги должна иметь:

1. Удобный и понятный графический пользовательский интерфейс.
2. Возможность работы с системой множества пользователей.
3. Возможность добавления, удаления и редактирования данных: объекта учета, услуг и счетчиков.
4. Возможность выбора формы оплаты.
5. Возможность совершения и удаления платежей.
6. Возможность предоставления отчетов по совершенным платежам в графической форме.

Хранение данных системы должно осуществляться посредствам СУБД MySQL.

Логика работы системы.

После запуска, приложение попытается подключиться к MySQL – серверу (логин и пароль для подключения должны быть «root») и базе данных communalpayments. Сдесь возможны 2-а варианта развития событий:

1. Приложение сможет подключиться к серверу, но не сможет подключиться к базе данных.
2. Приложение не сможет подключиться к серверу.

В первом случае – будет создана база данных с названием communalpayments и все необходимые таблицы в ней.

Во втором случае пользователь получит сообщение об ошибке.

После успешного запуска приложения и подключения к БД (база данных) пользователю в указанной последовательности необходимо будет сделать следующее:

1. Находясь на вкладке «Объект учета» добавить объект за которым будем вести наблюдение посредствам нажатия кнопки «Добавить» и заполнения всех данных в открывшемся диалоговом окне.
2. Перейти во вкладку «Услуги» и добавить все необходимые пользователю услуги, действуя при этом также, как в пункте 1.
3. Если добавленные ранее услуги имеют форму оплаты «по счетчику», то необходимо добавить счетчики к соответствующим услугам. Сделать это можно во вкладке «Счетчики».

После внесения всех необходимых данных, система готова к работе. Все внесенные данные можно редактировать или, при необходимости, удалить.

Для совершения платежа, необходимо перейти на вкладку «Платежи» и нажать кнопку добавить. После этого появится диалоговое окно с данными по услуге и рассчитанной, по данным пользователя, суммой которую необходимо заплатить за услугу. Если услуга имеет форму оплаты «по счетчику», то необходимо будет внести показания всех счетчиков, закрепленных за данной услугой. Платежи можно только добавить, либо удалить.

Во вкладке «Отчет», пользователь может увидеть годовой и месячный отчет о произведенных платежах по услугам в виде графических диаграмм.

Все внесенные пользователем данные будут располагаться в таблицах соответствующих вкладок. Есть возможность отключения столбцов таблиц.

## Выбор шаблонов проектирования

Согласно требований, предъявляемых к системе, наше приложение будет представлять собой некий графический интерфейс, посредствам которого пользователь будет работать с базой данных.

Для отделения бизнес-логики от ее визуализации и возможности дальнейшего масштабирования системы мы воспользуемся шаблоном проектирования MVC (Модель – представление - контроллер) диаграмма которого представлена на рис.1.

MVC (Model - View - Controller) – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

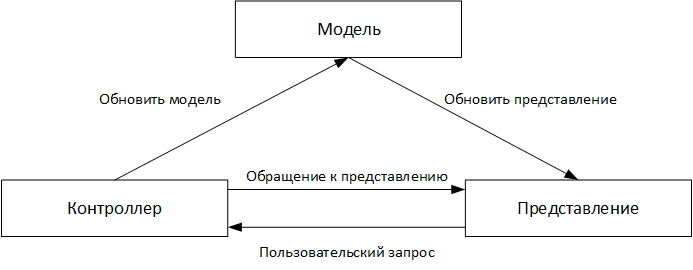


Рисунок 1 – Диаграмма шаблона MVC

Модель, представляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Реагирует на команды контроллера, изменяя свое состояние. Не зависит от представления – не знает, как данные визуализировать – и контроллера – не имеет точек взаимодействия с пользователем – просто предоставляет доступ к данным и управлению ими. За счет независимости от визуального представления, может иметь несколько различных представлений для одной модели.

Представление – отвечает за отображение данных модели пользователю, реагируя на изменение модели. Представление не обрабатывает введенные данные пользователя. Может влиять на состояние модели, сообщая модели об этом.

Контроллер – обеспечивает связи между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия. Интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Используя данный шаблон, мы достигаем следующих целей:

1. Получаем возможность представления данных одновременно в разном виде (таблица, круговая диаграмма и гистограмма).
2. Свободно можем изменять реакции на действия пользователя (ввод данных, нажатие кнопки мыши и т.д.) – для этого достаточно использовать другой контроллер.
3. Разделяем бизнес-логику приложения и графический интерфейс.

Следующим шаблоном, который который нам понадобится в проекте, будет шаблон проектирования «Наблюдатель» (Observer). Диаграмма шаблона представлена на рис.2.

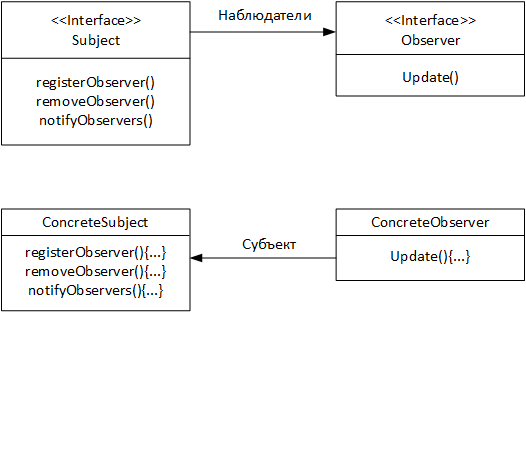


Рисунок 2 – Диаграмма шаблона «Наблюдатель» (Observer)

Интерфейс субъекта используется объектами для регистрации в качестве наблюдателя, а также исключения из списка таковых. Каждый субъект может иметь много наблюдателей. Субъект реализует интерфейс Subject. Кроме методов регистрации (registerObserver ()) и исключения (removeObserver ()), субъект также реализует метод notifyObservers (), оповещающий всех текущих наблюдателей об изменении состояния.

Каждый потенциальный наблюдатель должен реализовать интерфейс Observer. Интерфейс содержит единственный метод update (), который вызывается при изменении состояния субъекта. Наблюдатели могут относиться к любому классу, реализующему интерфейс Observer. Каждый наблюдатель регистрируется у конкретного субъекта для получения обновлений.

В нашем проекте с помощью этого шаблона мы реализуем оповещение «представления» об изменении «модели».

Т.к. данные для работы нашего приложения мы планируем хранить, используя СУБД, то нам желательно избавиться от жестких связей с механизмом хранения данных. Для этой цели нам подойдет шаблон проектирования DAO (Data Access Object).

DAO – это объект, который предоставляет интерфейс к какому-либо типу баз данных или механизму хранения (например файл). Определенные возможности предоставляются независимо от того, какой механизм хранения используется и без необходимости специальным образом соответствовать этому механизму хранения.

Используя этот шаблон в нашем приложении, мы абстрагируем и инкапсулируем доступ к источнику данных. DAO управляет соединением с источником данных для получения и записи данных. Это дает нам гибкость в смене этих механизмов хранения данных. Например, при смене базы данных, нам достаточно будет реализовать интерфейс (предоставляемый DAO) нужным нам способом, а все оставшееся приложение будет без изменений. Диаграмма классов, в общем виде, представляющая взаимоотношения в шаблоне DAO представлена на рис. 3.

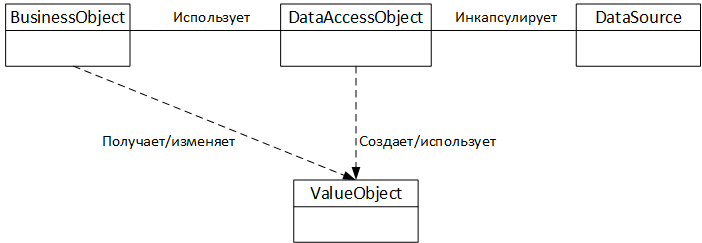


Рисунок 3 – Общая диаграмма классов DAO

BusinessObject – представляет клиента данных. Это объект, который нуждается в доступе к источнику данных для получения и сохранения данных. BusinessObject передает также ответственность за выполнение операций загрузки и сохранения данных объекту DataAccessObject.

DataAccessObject – является первичным объектом данного шаблона. Он абстрагирует используемую реализацию доступа к данным для BusinessObject, обеспечивая прозрачный доступ к источнику данных.

DataSource – представляет собой реализацию источника данных.

ValueObject – представляет собой объект, используемый для передачи данных. DataAccessObject может использовать его для возврата данных клиенту и может также принимать данные от клиента, для их обновления в источнике данных.

В нашем проекте в купе с DAO мы будем использовать шаблон проектирования «Фабричный метод» (Factory Method) для генерации нескольких объектов DAO, которые нужны приложению. Это позволит сделать DAO еще гибче. Такая стратегия называется Factory for Data Access Object, она использует шаблон Factory Method для реализации конкретных генераторов и их продуктов (объектов DAO). Общая диаграмма классов DAO с использованием Factory Method имеет вид, показанный на рис. 4.

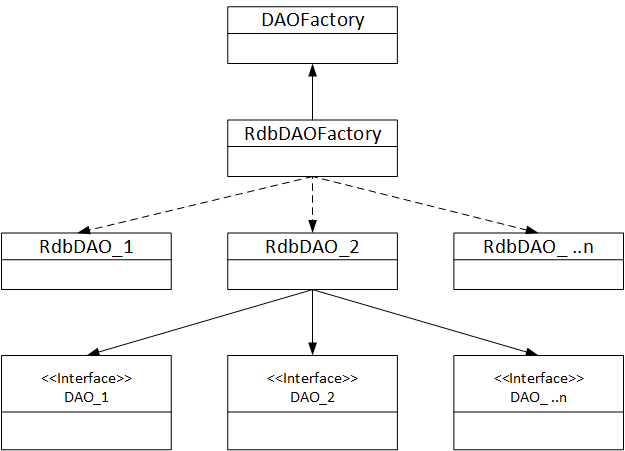


Рисунок 4 – Стратегия Factory for DAO использующая Factory Method

Что дает нам использование паттерна DAO в нашем проекте:

* Разрешает прозрачность. Бизнес-объекты могут использовать источник данных, не имея знаний о конкретных деталях его реализации. Доступ является прозрачным, поскольку детали реализации скрыты внутри DAO.
* Облегчает миграцию. Уровень объектов DAO облегчает приложению миграцию на другую реализацию базы данных. Бизнес-объекты не знают о деталях реализации используемых данных. Следовательно, процесс миграции требует изменений только в уровне DAO. Более того при использовании стратегии генератора, мы предоставляем конкретную реализацию генератора для каждой реализации хранилища данных.
* Уменьшает сложность кода в бизнес-объектах. Поскольку объекты DAO управляют всеми сложностями доступа к данным, упрощается код бизнес-компонентов и других клиентов данных, использующих DAO. Весь зависящий от реализации код (например, SQL- команды) содержится в DAO, а не в бизнес-объекте. Это улучшает читаемость кода и производительность разработки.
* Централизует весь доступ к данным в отдельном уровне. Поскольку все операции доступа к данным реализованы в объектах DAO, отдельный уровень доступа к данным может рассматриваться как уровень, изолирующий остальную часть приложения от реализации доступа к данным. Такая централизация облегчает поддержку и управление приложением.

Для инкапсуляции взаимодействий между представлением и базой данных мы реализуем шаблон проектирования посредник (Mediator). Mediator будет выступать в качестве посредника в общении между контроллерами и классами DAO. Т.е. все обращения к базе данных из контроллеров будут осуществляться через объект – посредник.

## Структура проекта

Архитектура проекта выполнена согласно шаблону проектирования MVC (Модель-представление-контроллер). Все классы проекта распределены по пакетам, название которых говорит о их содержимом.

Общая структура пакетов проекта представлена на рис.5. Она отражает структуру программного комплекса, организацию диалога и показывает связь модулей программы между собой.

Пакет representationObjects – содержит в себе набор классов представления и класс-посредник Database.

В пакете controllers содержатся классы контроллеров (классов бизнес-логики), отвечающих за взаимодействие представления (графического интерфейса пользователя) и модели (база данных), расположенных в соответствующих пакетах.

Пакеты daoInterfaces и daoImplements содержат интерфейсы и классы реализации модели DAO. Пакеты connectionInterfaces и connectionImplements содержат интерфейсы установки соединения с БД и классы их реализующие.

Пакет observerInterfaces содержит интерфейсы, необходимые для реализации шаблона проектирования Observer (Наблюдатель).

Кроме того, в структуру проекта входят пакеты: lib (хранит JDBC драйвер для MySQL), ico (хранит файлы иконок проекта), fxml (хранит fxml- файлы диалоговых окон и главного окна приложения).

|  |  |
| --- | --- |
|  | Рисунок 5 – Структура пакетов проекта |

# Реализация приложения

## Обзор программных средств и применяемых технологий

Для разработки данного программного продукта были выбраны следующие программные средства:

* Интегрированная среда разработки Intellij Idea;
* Язык программирования Java;
* Платформа для создания графического интерфейса – JavaFX;
* JavaFX Scene Builder – редактор форм под JavaFX;
* Система управления базами данных MySQL;
* MySQL Workbench – инструмент для визуального проектирования баз данных;
* Moqups – онлайн приложение для создания макета.

Интегрированная среда разработки (IDE) - это среда, в которой разработчику предоставляется всё необходимое для написания, отладки, запуска и тестирования приложений.

В состав IDE входит несколько элементов: редактор кода, отладчик, набор панелей инструментов, обширная библиотека компонентов, редактор изображений, инструментарий баз данных.

Intellij Idea – это интегрированная среда разработки программного обеспечения на языке Java от компании JetBrains. Среда с успехом используется и для других языков программирования, например, Scala.

Первая версия программы появилась в 2001 г. и с тех пор программа неуклонно повышает свой рейтинг популярности.

Выпускается в двух редакциях:

* Community Edition – является полностью бесплатной.
* Ultimate Edition – распространяется под различными лицензиями.

Компания JetBrains поддерживает также и студенческую программу, т.е. есть возможность получения студенческой лицензии на все продукты компании.

JavaFx – платформа на основе Java для создания приложений с насыщенным графическим интерфейсом. Может использоваться как для создания настольных приложений, запускаемых непосредственно из –под операционных систем, так и для интернет-приложений (RIA), работающих в браузерах, и для приложений на мобильных устройствах. JavaFX призвана заменить использующуюся ранее библиотеку Swing. Платформа конкурирует с Microsoft Silverlight, Adobe Flash и аналогичными системами.

Приложения JavaFX создаются с помощью декларативного языка программирования JavaFX Script. Из кода, написанного на JavaFX Script, можно обращаться к любым библиотекам Java. Поэтому совместное использование JavaFX Script и Java позволяет решать разнообразные задачи, например, бизнес-логика приложения может быть написана на Java, а графический интерфейс пользователя на JavaFX Script.

JavaFX Scene Builder – редактор предназначенный для редактирования графических форм под JavaFX.

Выбор СУБД MySQL обусловлен правилами ее распространения - за нее не надо платить деньги, и она соответствует требованиям приложения.

MySQL Workbench – инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для систем баз данных MySQL. Является приемником DBDesigner от FabForce.

Moqups – это HTML-5 – приложение для создания макетов, концептов пользовательского интерфейса.

## Этапы реализации приложения

Первым этапом реализации проекта является проектирование и создание базы данных приложения. Для проектирования модели базы данных воспользуемся программой MySQL Workbench. Получившаяся модель базы данных представлена на рис. 6.

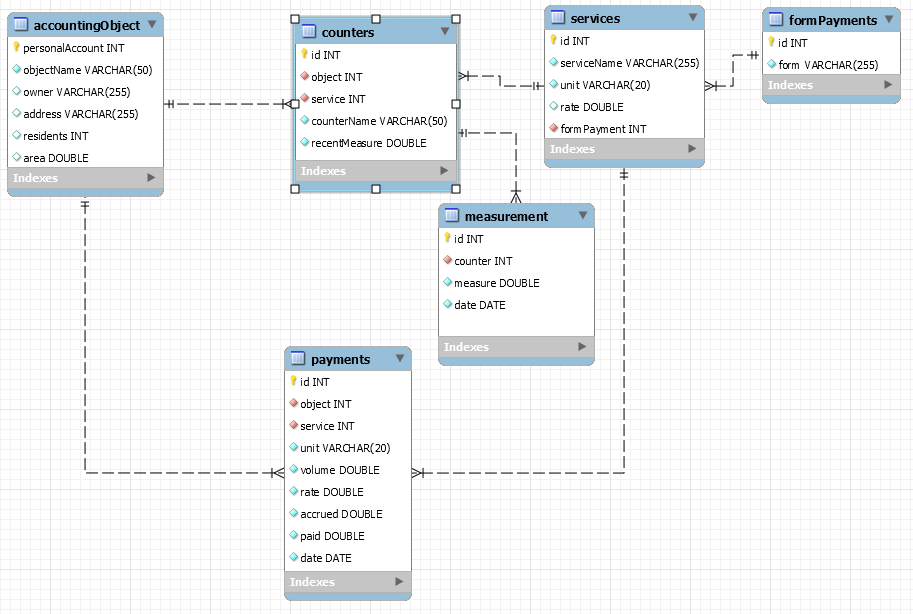


Рисунок 6 – Модель базы данных приложения

У нас получилось шесть таблиц:

1. AccountingObject – таблица, хранящая данные по всем «Объектам учета», имеющимся в системе.
2. Counters – таблица, в которой хранятся данные всех счетчиков системы.
3. Services – таблица, хранящая данные по всем услугам системы.
4. FormPayments – таблица, в которой хранятся все доступные формы платежей.
5. Measurement – таблица, хранящая данные измерений по счетчикам.
6. Payments – таблица, хранящая данные по совершенным платежам.

Названия столбцов таблиц, их тип и связи между таблицами видны на рис. 6.

В дальнейшем для работы нашего приложения будет необходимо наличие на машине клиента, развернутого MySQL-server с пользователем root и паролем доступа – root. Наше приложение при запуске попытается подключиться к серверу MySQL используя указанные имя пользователя и пароль, проверит наличие базы данных «communalpayments», при отсутствии таковой – создаст ее.

Вторым этапом является проектирование визуальной формы программы, а именно создание макета главной формы приложения. При создании макета нужно учесть множество факторов, например, таких как: удобство, понятность, легкодоступность и т.д. Для создания макета воспользуемся онлайн сервисом [moqups.com](http://moqups.com) – это HTML-5 – приложение для создания макетов, концептов пользовательского интерфейса. Результат создания макета отображен на рисунке 6.

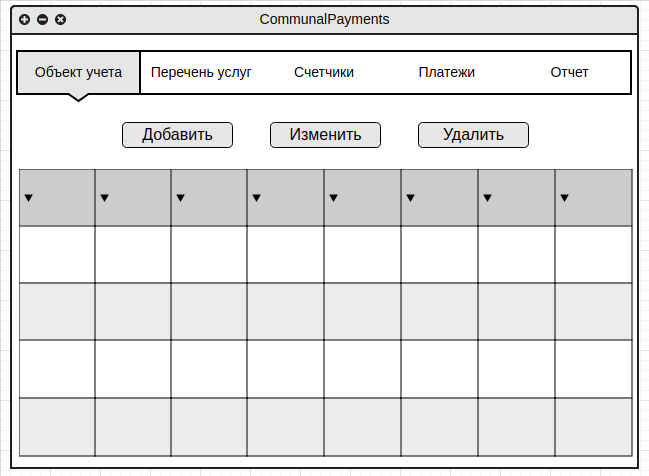


Рисунок 6 – Макет главной формы

Макет главного окна приложения состоит из пяти вкладок. Все вкладки, кроме вкладки «Отчет», содержат по три функциональных кнопки («Добавить», «Изменить», «Удалить») и таблицу, в которую выводится соответствующая информация. Информация, выводимая в таблицы, соответствует данным, находящимся в соответствующей таблице базы данных.

Далее нужно реализовать данный макет в визуальном редакторе в виде формы. Для этого мы воспользуемся JavaFX Scene builder. Отредактированный макет мы сохраняем в файл FXML, чтобы далее использовать его в своей программе.

Результат реализации формы изображен на рисунке 7.

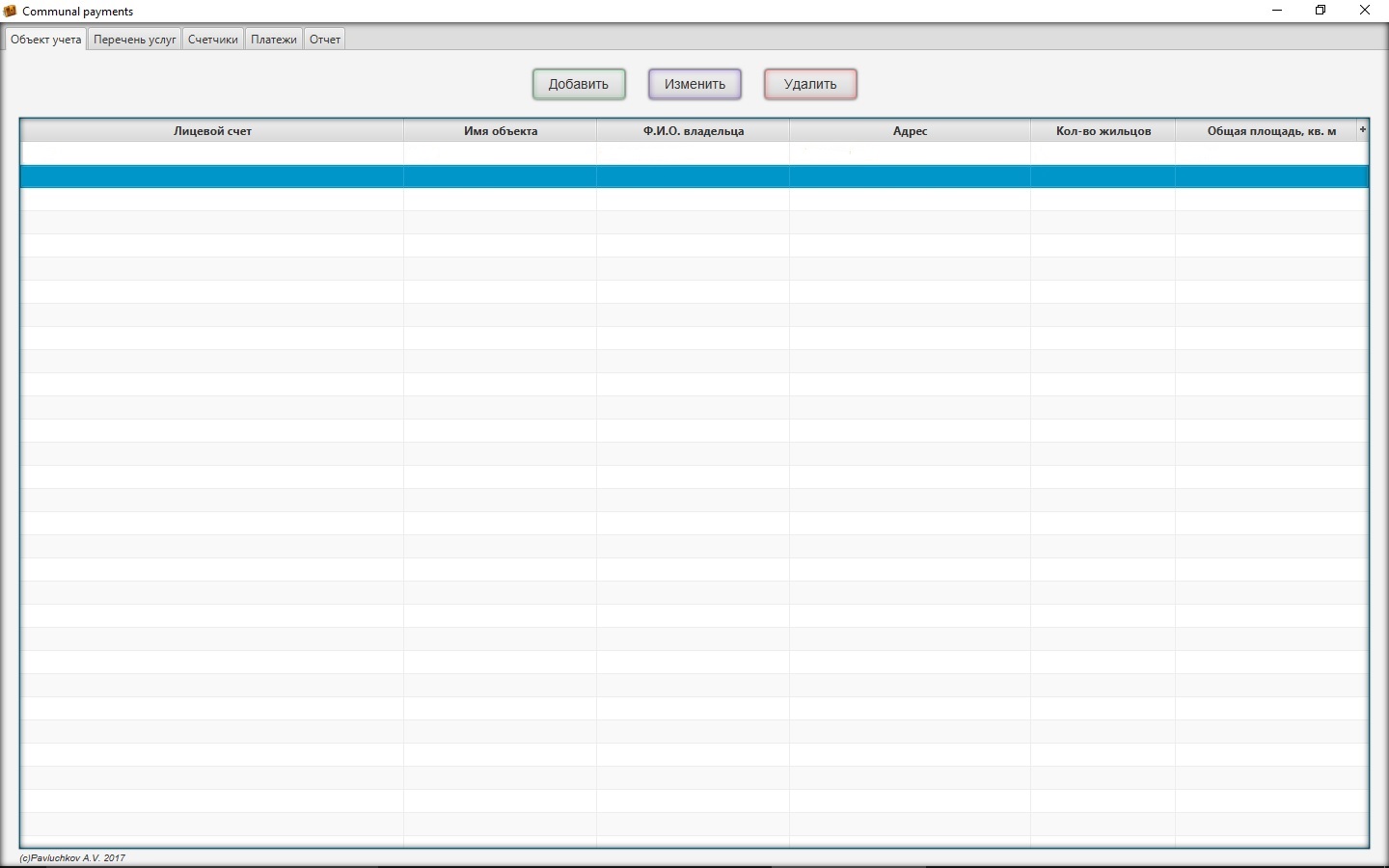


Рисунок 7 – Главное окно приложения

Таким же образом мы создаем все диалоговые окна приложения (окна ввода параметров при добавлении и изменение счетчиков, услуг и объектов учета, а также окно подтверждения удаления). Вид получившихся диалоговых окон показан на рис. 8.

Следует немного остановиться на реализации вкладки «Отчет», т.к. она в значительной степени отличается от всех остальных. Вкладка представляет из себя окно, разделенное на две половины и имеющее в верхней части три элемента ComboBox рис. 9.

Элементы comboBox позволяют выбрать «Объект учета», по которому предоставляется отчет, а также месяц и год, за которые хотим получить отчет.

В левой части окна отображается pieChart – диаграмма (круговая диаграмма), представляющая собой окружность, разделенную на некоторое количество долей, окрашенных разным цветом, соответствующих имеющимся у выбранного «объекта учета» услугам. Диаграмма показывает месячный расход по услугам, за выбранный год и месяц.

При щелчке левой кнопкой мыши по одной из долей диаграммы, мы получим информацию о процентном и денежном эквиваленте платежа по выбранной услуге в общем объеме месячных платежей.

В правой части окна отображается barChart – диаграмма (гистограмма), представляющая собой некоторое количество прямоугольных столбиков, окрашенных в разный цвет. Эта диаграмма показывает годовой расход по каждой услуге, для выбранного объекта. Каждый столбик соответствует годовому расходу по услуге.

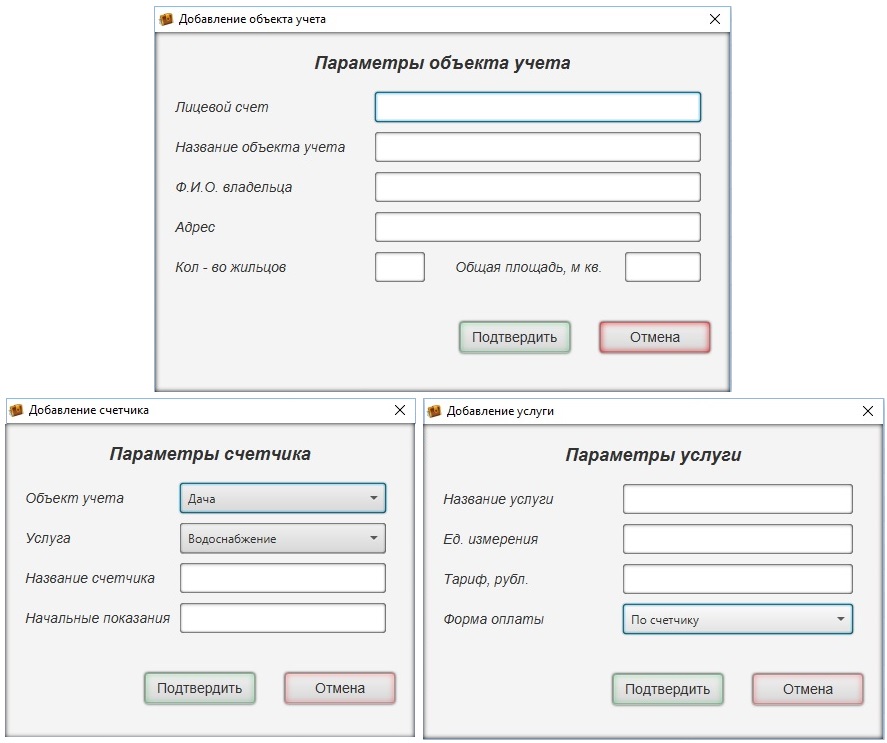


Рисунок 8 – Диалоговые окна

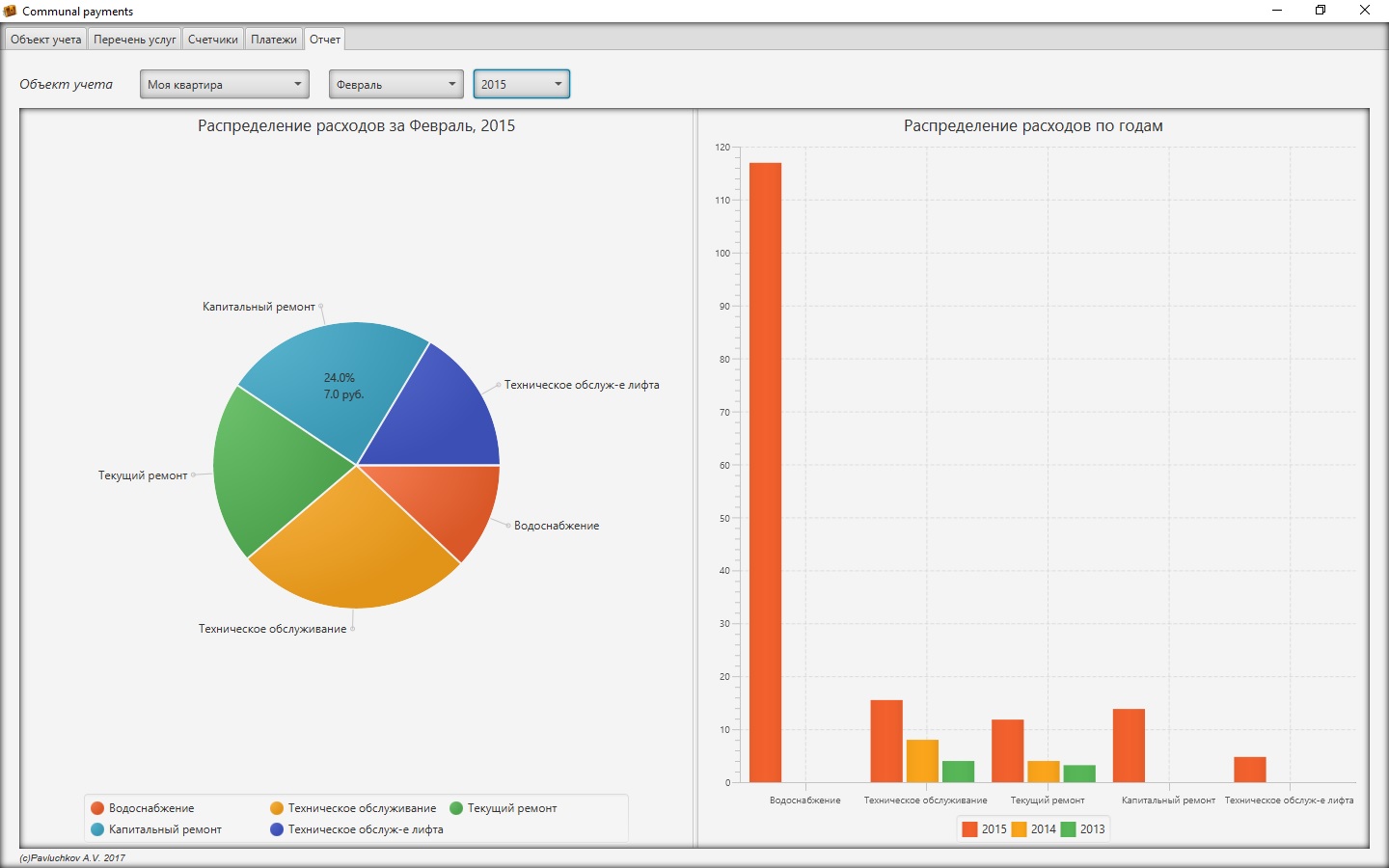


Рисунок 9 – Вид вкладки «Отчет»

Третьим этапом реализации проекта является программирование всех управляющих элементов, находящихся на форме, а также тестирование, исправление и отладка программного продукта.

## Описание основных классов и методов проекта

Общая диаграмма классов и public-методов проекта представлена на рис. 10. Данная диаграмма отражает структуру классов проета и связи между ними.

Основным классом приложения является класс Main.java из пакета main. Этот класс расширяет класс Application и содержит два метода. В общем-то это базовая структура, которая необходима для запуска JavaFX-приложения.

Метод start (Stage primaryStage) автоматически вызывается при вызове метода launch (…) из метода main. Stage - является основным контейнером, который, как правило, представляет собой обрамленное окно со стандартными кнопками: закрыть, свернуть, развернуть. Внутрь Stage добавляется Scene, а внутрь Scene – стандартные компоненты (textbox, comboBox и т.д.). Так, в методе start (…) мы загружаем макет главного окна приложения из файла mainWindow.fxml, устанавливаем размеры сцены (scene), иконку окна приложения, устанавливаем контроллер и отображаем все это вызовом метода show ().

Класс MainController – является контроллером главного окна приложения. Класс реализует интерфейс Observer и устанавливает наблюдение за объектом Database, который, в свою очередь, реализует интерфейс Subject. В этом классе реализуется логика обработки событий нажатия трех функциональных кнопок приложения: «Добавить», «Изменить», «Удалить». После нажатия одной из функциональных кнопок, при помощи метода dialogWindow (…), вызывается нужное диалоговое окно и управление передается соответствующему контроллеру. Каждому диалоговому окну соответствует свой контроллер. Все контроллеры диалоговых окон (находятся в пакете controllers) наследуются от MainController и используют его объект Database, для доступа к базе данных. По сути, для каждой вкладки главного окна приложения (кроме вкладки «Отчет» - она имеет отдельный контроллер) есть три контроллера – по одному на каждую функциональную кнопку.

Объект Database осуществляет роль посредника между объектами базы данных и контроллерами. Он является наблюдаемым для класса MainController. Т.е. при изменении объекта Database (происходит какое-либо изменение в БД) - оповещается класс MainController и вызывается метод update () (происходит обновление таблиц главного окна приложения). У класса Database есть методы:

* setConnection () – для установки соединения с сервером БД;
* addObject () – для добавления объекта учета в БД;
* changeObject () – для изменения объекта учета;
* deleteObject () – для удаления объекта учета;
* getTableObject () – возвращает весь список объектов из БД;
* Аналогичный набор методов для работы с услугами, счетчиками и платежами;
* registerObserver () – для регистрации наблюдателя;
* removeObserver () – для удаления наблюдателя;
* notifyObserver () – для оповещения наблюдателей.

Класс MyObjects – абстрактный класс от которого наследуются все классы представления: ObjectAccounting, Counters, Services, Measurement, Payments, FormPayments. Поля этих классов повторяют собой сигнатуру полей соответствующих таблиц базы данных. Они имеют стандартные методы для установки и получения значений поля (get и set - методы). Объектами этих классов заполняются таблицы главного окна приложения.

ConnectionBuilderFactory – класс-фабрика, с помощью которого мы устанавливаем соединение с БД, имеющий один единственный метод – getSimpleConnection (). Метод возвращает объект интерфейса ConnectionBuilder. Таким образом образом мы не привязываемся жестко к конкретному классу реализации соединения с БД и свободно можем менять реализацию. Например, можем реализовать получение соединения с БД из пула-соединений. В нашем проекте за реализацию интерфейса ConnectionBuilder отвечает класс MySQLSimpleConnection у которого есть два метода: getConnection () и createDatabase (). Первый метод возвращает объект типа Connection, а второй – создает нужную нам базу данных, в случае ее отсутствия.

MySQLDaoFactory – класс-фабрика, реализующий интерфейс DaoFactory. Эта фабрика, занимается созданием объектов типа MyObjects. Используя методы этой фабрики, мы можем получать объекты нужного нам типа из базы данных. Для каждого такого объекта есть соответствующий интерфейс и класс его реализующий (в названии класса реализации присутствует слово MySQL, например, MySQLCounters).

Вся бизнес-логика представления данных в виде графических диаграмм, реализована в классе-контроллере ReportController. Класс имеет всего два публичных метода: drawPieChart () и drawBarChart (). Методы выводят соответствующую графическую диаграмму во вкладке «Отчет».

Весь исходный код приведен в приложении А.

|  |  |
| --- | --- |
|  | Рисунок 10 – Общая диаграмма классов проекта |

# Заключение

При выполнении курсового проекта были закреплены навыки программирования с использованием шаблонов проектирования. Изучены способы применения их на практике для решения вычислительных задач на языке программирования Java.

В процессе работы выполнены следующие исследования и разработки:

1. Изучены основы работы в среде разработки Intellij Idea;
2. Изучена технология создания графических интерфейсов в JavaFX.
3. Изучены основные шаблоны проектирования и стратегии их применения при проектировании программного продукта;
4. Изучены методы составления алгоритмов решения задач;
5. Разработан алгоритм работы системы;
6. Исходя из поставленной задачи, создан программный продукт;
7. Проведены все виды тестов над программой;

Поставленные задачи в ходе выполнения курсового проекта выполнены в полном объеме, что означает - данный программный продукт обладает всеми качествами надежного программного продукта. Областью возможного практического применения является любой персональный компьютер, нуждающийся в данном приложении.

# Список использованных источников

1. Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. Паттерны проектирования. - СПб.: Питер, 2011. — 656 с.: ил.
2. Герберт Шилд. Java 8. Полное руководство, 9-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2015. – 1376 с.: ил. – Парал. тит. англ.
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2015. — 368 с.: ил. - (Серия «Библиотека программиста»).
4. Официальная документация по JavaFX от Oracle: <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>
5. Общедоступная мультиязычная универсальная интернет -энциклопедия со свободным контентом: <http://www.wikipedia.org> .

# Приложение А

package by.javafx.communalPayments.main;

import by.javafx.communalPayments.controllers.MainController;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.image.Image;

import javafx.stage.Stage;

public class Main extends Application {

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) throws Exception {

FXMLLoader fxmlLoader = new FXMLLoader();

fxmlLoader.setLocation(getClass().getResource("/by/javafx/communalPayments/fxml/mainWindow.fxml"));

fxmlLoader.setController(new MainController());

Parent root = fxmlLoader.load();

primaryStage.setTitle("Communal payments");

primaryStage.setScene(new Scene(root, 1000, 600));

primaryStage.setMinWidth(1000);

primaryStage.setMinHeight(600);

Image ico = new Image("/by/javafx/communalPayments/ico/icon.png");

primaryStage.getIcons().add(ico);

primaryStage.show();

}

}

package by.javafx.communalPayments.controllers.counters;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.Counters;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class CounterAddController extends MainController {

private ObservableList<ObjectAccounting> tableObject = FXCollections.observableArrayList();

private ObservableList<Services> tableService = FXCollections.observableArrayList();

@FXML

private ComboBox<String> objectCombo;

@FXML

private ComboBox<String> serviceCombo;

@FXML

private Label measureLabel;

@FXML

private TextField nameField;

@FXML

private TextField recentField;

@FXML

private Button btnCancel;

@FXML

public void initialize() {

tableObject = database.getTableObject();

tableService = database.getTableServices();

ObservableList<String> listObjects = FXCollections.observableArrayList();

ObservableList<String> listServices = FXCollections.observableArrayList();

for (ObjectAccounting obj : tableObject) {

listObjects.add(obj.getObjectName());

}

for (Services obj : tableService) {

if (obj.getFormPayments() == 1) {

listServices.add(obj.getServiceName());

}

}

objectCombo.setItems(listObjects);

serviceCombo.setItems(listServices);

objectCombo.setValue(listObjects.get(0));

serviceCombo.setValue(listServices.get(0));

}

@FXML

void btnOkClicked() {

String counterName = nameField.getText();

if (counterName.isEmpty()) {

printDialogError("Ввод данных", "Ошибка ввода данных !", "Введите имя счетчика.");

return;

}

double recentMeasure;

try {

recentMeasure = Double.parseDouble(recentField.getText());

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

String selectedItemObj = objectCombo.getValue();

String selectedItemService = serviceCombo.getValue();

int objectId = 0;

for (ObjectAccounting obj : tableObject) {

if (selectedItemObj.equals(obj.getObjectName())) {

objectId = obj.getId();

}

}

int serviceId = 0;

for (Services obj : tableService) {

if (selectedItemService.equals(obj.getServiceName())) {

serviceId = obj.getId();

}

}

Counters counter = new Counters(0, objectId, serviceId, counterName, recentMeasure);

if (!database.addCounter(counter)) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.counters;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.Counters;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class CounterChangeController extends MainController {

private Counters object;

private MainController mainController;

private ObservableList<ObjectAccounting> tableObject = FXCollections.observableArrayList();

private ObservableList<Services> tableService = FXCollections.observableArrayList();

@FXML

private ComboBox<String> objectCombo;

@FXML

private ComboBox<String> serviceCombo;

@FXML

private TextField nameField;

@FXML

private TextField recentField;

@FXML

private Label measureLabel;

@FXML

private Button btnCancel;

public CounterChangeController(MainController mainController) {

this.mainController = mainController;

}

@FXML

public void initialize() {

object = (Counters) mainController.getSelectedObject();

measureLabel.setText("Последние показания");

tableObject = database.getTableObject();

tableService = database.getTableServices();

ObservableList<String> listObjects = FXCollections.observableArrayList();

ObservableList<String> listServices = FXCollections.observableArrayList();

for (ObjectAccounting obj : tableObject) {

listObjects.add(obj.getObjectName());

if (obj.getId() == object.getObject()) {

objectCombo.setValue(obj.getObjectName());

}

}

for (Services obj : tableService) {

if (obj.getFormPayments() == 1) {

listServices.add(obj.getServiceName());

}

if (obj.getId() == object.getService()) {

serviceCombo.setValue(obj.getServiceName());

}

}

nameField.setText(object.getCounterName());

recentField.setText(String.valueOf(object.getRecentMeasure()));

objectCombo.setItems(listObjects);

serviceCombo.setItems(listServices);

}

@FXML

void btnOkClicked() {

String counterName = nameField.getText();

if (counterName.isEmpty()) {

printDialogError("Ввод данных", "Ошибка ввода данных !", "Введите имя счетчика.");

return;

}

double recentMeasure;

try {

recentMeasure = Double.parseDouble(recentField.getText());

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

String selectedItemObj = objectCombo.getValue();

String selectedItemService = serviceCombo.getValue();

int objectId = 0;

for (ObjectAccounting obj : tableObject) {

if (selectedItemObj.equals(obj.getObjectName())) {

objectId = obj.getId();

}

}

int serviceId = 0;

for (Services obj : tableService) {

if (selectedItemService.equals(obj.getServiceName())) {

serviceId = obj.getId();

}

}

int id = object.getId();

Counters counter = new Counters(id, objectId, serviceId, counterName, recentMeasure);

if (!database.changeCounter(counter)) {

return;

}

if (!database.lastMeasureChange(counter, object.getRecentMeasure())) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.counters;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.Counters;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.stage.Stage;

public class CounterDeleteController extends MainController {

private MainController mainController;

private Counters object;

@FXML

private Label nameDeleteObject;

@FXML

private Button btnCancel;

public CounterDeleteController(MainController mainController) {

this.mainController = mainController;

}

@FXML

public void initialize() {

object = (Counters) mainController.getSelectedObject();

nameDeleteObject.setText("\"" + object.getCounterName() + "\"");

}

@FXML

public void btnOkClicked() {

if (!database.deleteCounter(object)) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.counters;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.controllers.payments.PaymentAddController;

import by.javafx.communalPayments.representationObjects.Counters;

import by.javafx.communalPayments.representationObjects.Measurement;

import by.javafx.communalPayments.representationObjects.Payments;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.DatePicker;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import java.sql.Date;

import java.time.LocalDate;

public class MeasureController extends MainController {

private Counters object;

private PaymentAddController paymentAddController;

@FXML

private Label textLabel;

@FXML

private TextField previousMeasureField;

@FXML

private TextField measureField;

@FXML

private DatePicker datePicker;

@FXML

private Button btnCancel;

public MeasureController(PaymentAddController paymentAddController, Counters counter) {

this.object = counter;

this.paymentAddController = paymentAddController;

}

@FXML

public void initialize() {

textLabel.setText(object.getCounterName());

previousMeasureField.setText(String.valueOf(object.getRecentMeasure()));

previousMeasureField.setEditable(false);

datePicker.setValue(LocalDate.now());

}

@FXML

public void btnOkClicked() {

if ((measureField.getText().isEmpty())) {

printDialogError("Внесение показаний счетчика", "Ошибка внесения показаний!",

"Поле <Текущие показания> должно быть заполнено");

return;

}

double rate = paymentAddController.getRate(object.getService());

double volume;

try {

volume = Double.parseDouble(measureField.getText()) - object.getRecentMeasure();

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

double sum = volume \* rate;

if (sum < 0) {

printDialogError("Ввод данных", "Ошибка ввода данных !", "Текущие показания меньше предыдущих.");

return;

}

Payments payment = new Payments();

payment.setId(0);

payment.setObject(object.getObject());

payment.setService(object.getService());

payment.setUnit("");

payment.setVolume(Math.rint(volume \* 1000) / 1000);

payment.setRate(rate);

payment.setAccrued(0);

payment.setPaid(0);

payment.setDate(Date.valueOf(LocalDate.now()));

paymentAddController.setPayment(payment);

object.setRecentMeasure(Double.parseDouble(measureField.getText()));

paymentAddController.setCounter(object);

Measurement measure = new Measurement(0, object.getId(), Double.parseDouble(measureField.getText()),

Date.valueOf(datePicker.getValue()));

paymentAddController.setMeasure(measure);

paymentAddController.setLayout(true);

paymentAddController.setSum(sum);

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.objectAccounting;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class ObjAddController extends MainController {

@FXML

private TextField personalAccount;

@FXML

private TextField nameObject;

@FXML

private TextField owner;

@FXML

private TextField address;

@FXML

private TextField residents;

@FXML

private TextField area;

@FXML

private Button btnCancel;

@FXML

public void initialize() {

}

@FXML

public void btnOkClicked() {

ObjectAccounting object = new ObjectAccounting();

try {

object.setId(Integer.parseInt(personalAccount.getText()));

object.setObjectName(nameObject.getText());

object.setOwner(owner.getText());

object.setAddress(address.getText());

object.setResidents(Integer.parseInt(residents.getText()));

object.setArea(Double.parseDouble(area.getText()));

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

if (!database.addObject(object)) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.objectAccounting;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class ObjChangeController extends MainController {

private ObjectAccounting objectAccounting;

private MainController mainController;

@FXML

private TextField personalAccount;

@FXML

private TextField nameObject;

@FXML

private TextField owner;

@FXML

private TextField address;

@FXML

private TextField residents;

@FXML

private TextField area;

@FXML

private Button btnCancel;

public ObjChangeController(MainController mainController) {

this.mainController = mainController;

}

@FXML

public void initialize() {

objectAccounting = (ObjectAccounting) mainController.getSelectedObject();

personalAccount.setText(String.valueOf(objectAccounting.getId()));

nameObject.setText(objectAccounting.getObjectName());

owner.setText(objectAccounting.getOwner());

address.setText(objectAccounting.getAddress());

residents.setText(String.valueOf(objectAccounting.getResidents()));

area.setText(String.valueOf(objectAccounting.getArea()));

}

@FXML

public void btnOkClicked() {

ObjectAccounting object = new ObjectAccounting();

try {

object.setId(Integer.parseInt(personalAccount.getText()));

object.setObjectName(nameObject.getText());

object.setOwner(owner.getText());

object.setAddress(address.getText());

object.setResidents(Integer.parseInt(residents.getText()));

object.setArea(Double.parseDouble(area.getText()));

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

if (!database.changeObject(object, objectAccounting.getId())) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.objectAccounting;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.stage.Stage;

public class ObjDeleteController extends MainController {

private ObjectAccounting objectAccounting;

private MainController mainController;

@FXML

private Label nameDeleteObject;

@FXML

private Button btnCancel;

public ObjDeleteController(MainController mainController) {

this.mainController = mainController;

}

@FXML

public void initialize() {

objectAccounting = (ObjectAccounting) mainController.getSelectedObject();

nameDeleteObject.setText("\"" + objectAccounting.getObjectName() + "\"");

}

@FXML

public void btnOkClicked() {

if (!database.deleteObject(objectAccounting)) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.payments;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.controllers.counters.MeasureController;

import by.javafx.communalPayments.representationObjects.\*;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.TextField;

import javafx.scene.layout.HBox;

import javafx.stage.Stage;

import java.sql.Date;

import java.time.LocalDate;

import java.util.ArrayList;

public class PaymentAddController extends MainController {

private Payments payment = new Payments();

private ObservableList<ObjectAccounting> tableObject = FXCollections.observableArrayList();

private ObservableList<Services> tableService = FXCollections.observableArrayList();

private ObservableList<String> listObjects = FXCollections.observableArrayList();

private ObservableList<String> listServices = FXCollections.observableArrayList();

private ArrayList<Counters> listCounters = new ArrayList<>();

private ArrayList<Measurement> listMeasure = new ArrayList<>();

private ArrayList<Counters> newListCounters = new ArrayList<>();

private double sum;

private double area;

private int residents;

private String serviceComboValue;

@FXML

private ComboBox<String> objectCombo;

@FXML

private ComboBox<String> serviceCombo;

@FXML

private TextField sumField;

@FXML

private Button btnMeasure;

@FXML

private Button btnOk;

@FXML

private Button btnCancel;

@FXML

private Button btnCancelMeasure;

@FXML

private HBox layout\_1;

@FXML

private HBox layout\_2;

@FXML

public void initialize() {

tableObject = database.getTableObject();

tableService = database.getTableServices();

for (ObjectAccounting obj : tableObject) {

listObjects.add(obj.getObjectName());

}

for (Services obj : tableService) {

listServices.add(obj.getServiceName());

}

objectCombo.valueProperty().addListener((observable, oldValue, newValue) -> checkCombo(serviceComboValue));

serviceCombo.valueProperty().addListener((observable, oldValue, newValue) -> {

serviceComboValue = newValue;

checkCombo(serviceComboValue);

});

objectCombo.setItems(listObjects);

serviceCombo.setItems(listServices);

objectCombo.setValue(listObjects.get(0));

serviceCombo.setValue(listServices.get(0));

}

@FXML

public void btnOkClicked() {

for (Counters obj : newListCounters) {

if (!database.changeCounter(obj)) {

return;

}

}

for (Measurement obj : listMeasure) {

if (!database.addMeasurement(obj)) {

return;

}

}

if (listMeasure.size() != 0) {

payment.setAccrued(Math.rint(sum \* 100) / 100);

try {

payment.setPaid(Double.parseDouble(sumField.getText()));

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

for (Services obj : tableService) {

if (payment.getService() == obj.getId()) {

payment.setUnit(obj.getUnit());

}

}

} else {

try {

payment.setPaid(Double.parseDouble(sumField.getText()));

payment.setDate(Date.valueOf(LocalDate.now()));

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

}

if (!database.addPayment(payment)) {

return;

}

sum = 0;

btnCancelClicked();

}

@FXML

public void inputMeasureClicked() {

fillListCounters();

if (listCounters.size() != 0) {

for (Counters obj : listCounters) {

dialogWindow(serviceCombo.getScene().getWindow(), new MeasureController(this, obj), "/by/javafx/communalPayments/fxml/countersDialog/inputMeasure.fxml",

"Внесение показаний счетчика", 430, 255);

}

} else {

printDialogError("Счетчики", "Ошибка внесения показаний!",

"Отсутствуют счетчики для данного объекта учета.\nВ разделе <Счетчики> необходимо добавить счетчик.");

}

listCounters.clear();

}

@FXML

public void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

private void checkCombo(String newValue) {

for (Services obj : tableService) {

if (obj.getServiceName().equals(newValue)) {

double rate = obj.getRate();

payment.setRate(obj.getRate());

payment.setService(obj.getId());

payment.setUnit(obj.getUnit());

if (obj.getFormPayments() == 1) {

sum = 0;

sumField.setText("");

setLayout(false);

} else {

setLayout(true);

for (ObjectAccounting objectAccounting : tableObject) {

if (objectAccounting.getObjectName().equals(objectCombo.getValue())) {

area = objectAccounting.getArea();

residents = objectAccounting.getResidents();

payment.setObject(objectAccounting.getId());

}

}

if (obj.getFormPayments() == 2) {

sum = Math.rint(rate \* area \* 100) / 100;

payment.setVolume(area);

payment.setAccrued(sum);

sumField.setText(String.valueOf(sum));

}

if (obj.getFormPayments() == 3) {

sum = Math.rint(rate \* residents \* 100) / 100;

payment.setVolume(residents);

payment.setAccrued(sum);

sumField.setText(String.valueOf(sum));

}

}

}

}

}

private void fillListCounters() {

ObjectAccounting objectAccounting = new ObjectAccounting();

for (ObjectAccounting obj : tableObject) {

if (objectCombo.getValue().equals(obj.getObjectName())) {

objectAccounting = obj;

}

}

Services services = new Services();

for (Services obj : tableService) {

if (serviceCombo.getValue().equals(obj.getServiceName())) {

services = obj;

}

}

ObservableList<Counters> tableCounters;

tableCounters = database.getTableCounters();

for (Counters obj : tableCounters) {

if ((obj.getObject() == objectAccounting.getId()) && (obj.getService() == services.getId())) {

listCounters.add(obj);

}

}

}

public void setCounter(Counters counter) {

newListCounters.add(counter);

}

public void setMeasure(Measurement measure) {

listMeasure.add(measure);

}

public void setLayout(boolean b) {

if (b) {

layout\_1.setVisible(true);

layout\_2.setVisible(false);

sumField.setDisable(false);

} else {

layout\_1.setVisible(false);

layout\_2.setVisible(true);

sumField.setDisable(true);

}

}

public void setSum(double summa) {

this.sum += summa;

sumField.setText(String.valueOf(Math.rint(sum \* 100) / 100));

}

public double getRate(int serviceId) {

double rate = 0;

for (Services obj : tableService) {

if (serviceId == obj.getId()) {

rate = obj.getRate();

}

}

return rate;

}

public void setPayment(Payments payment) {

this.payment = payment;

}

}

package by.javafx.communalPayments.controllers.payments;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.Payments;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.stage.Stage;

public class PaymentsDeleteController extends MainController {

private MainController mainController;

private Payments object;

@FXML

private Label nameDeleteObject;

@FXML

private Button btnCancel;

public PaymentsDeleteController(MainController mainController) {

this.mainController = mainController;

}

@FXML

public void initialize() {

object = (Payments) mainController.getSelectedObject();

nameDeleteObject.setText("\"Платеж № " + object.getId() + "\"");

}

@FXML

public void btnOkClicked() {

if (!database.deletePayment(object)) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.services;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.FormPayments;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class ServiceAddController extends MainController {

private ObservableList<FormPayments> tableForm = FXCollections.observableArrayList();

@FXML

private TextField nameField;

@FXML

private TextField unitField;

@FXML

private TextField rateField;

@FXML

private ComboBox<String> formPaymentCmb;

@FXML

private Button btnCancel;

@FXML

public void initialize() {

tableForm = database.getTableFormPayments();

ObservableList<String> formPayments = FXCollections.observableArrayList();

for (FormPayments obj : tableForm) {

formPayments.add(obj.getForm());

}

formPaymentCmb.setItems(formPayments);

formPaymentCmb.setValue(formPayments.get(0));

}

@FXML

public void btnOkClicked() {

String serviceName = nameField.getText();

String unit = unitField.getText();

double rate;

if (serviceName.isEmpty()) {

printDialogError("Ввод данных", "Ошибка ввода данных !", "Введите имя услуги.");

return;

}

try {

rate = Double.parseDouble(rateField.getText());

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

int formId = 0;

for (FormPayments obj : tableForm) {

if (obj.getForm().equals(formPaymentCmb.getSelectionModel().getSelectedItem())) {

formId = obj.getId();

}

}

if (!database.addService(new Services(0, serviceName, unit, rate, formId))) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.services;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.FormPayments;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class ServiceChangeController extends MainController {

private Services object;

private MainController mainController;

private ObservableList<FormPayments> tableForm = FXCollections.observableArrayList();

@FXML

private TextField nameField;

@FXML

private TextField unitField;

@FXML

private TextField rateField;

@FXML

private ComboBox<String> formPaymentCmb;

@FXML

private Button btnCancel;

public ServiceChangeController(MainController mainController) {

this.mainController = mainController;

}

public ServiceChangeController() {

}

@FXML

public void initialize() {

object = (Services) mainController.getSelectedObject();

tableForm = database.getTableFormPayments();

ObservableList<String> formPayments = FXCollections.observableArrayList();

for (FormPayments obj : tableForm) {

formPayments.add(obj.getForm());

if (object.getFormPayments() == obj.getId()) {

formPaymentCmb.setValue(obj.getForm());

}

}

nameField.setText(object.getServiceName());

unitField.setText(object.getUnit());

rateField.setText(String.valueOf(object.getRate()));

formPaymentCmb.setItems(formPayments);

}

@FXML

public void btnOkClicked() {

String serviceName = nameField.getText();

String unit = unitField.getText();

double rate;

if (serviceName.isEmpty()) {

printDialogError("Ввод данных", "Ошибка ввода данных !", "Введите имя услуги.");

return;

}

try {

rate = Double.parseDouble(rateField.getText());

} catch (NumberFormatException e) {

printDialogError("Ввод данных", "Ошибка ввода данных !", e.getMessage());

return;

}

int formId = 0;

int serviceId = object.getId();

for (FormPayments obj : tableForm) {

if (obj.getForm().equals(formPaymentCmb.getSelectionModel().getSelectedItem())) {

formId = obj.getId();

}

}

if (!database.changeService(new Services(serviceId, serviceName, unit, rate, formId))) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.services;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.stage.Stage;

public class ServiceDeleteController extends MainController {

private Services object;

private MainController mainController;

@FXML

private Label nameDeleteObject;

@FXML

private Button btnCancel;

public ServiceDeleteController(MainController mainController) {

this.mainController = mainController;

}

public ServiceDeleteController() {

}

@FXML

public void initialize() {

object = (Services) mainController.getSelectedObject();

nameDeleteObject.setText("\"" + object.getServiceName() + "\"");

}

@FXML

public void btnOkClicked() {

if (!database.deleteService(object)) {

return;

}

btnCancelClicked();

}

@FXML

private void btnCancelClicked() {

Stage stage = (Stage) btnCancel.getScene().getWindow();

stage.close();

}

}

package by.javafx.communalPayments.controllers.report;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import by.javafx.communalPayments.representationObjects.Payments;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.scene.chart.CategoryAxis;

import javafx.scene.chart.NumberAxis;

import javafx.scene.chart.PieChart;

import javafx.scene.chart.XYChart;

import javafx.scene.control.Label;

import javafx.scene.input.MouseEvent;

import javafx.stage.Popup;

import java.time.LocalDate;

import java.time.format.TextStyle;

import java.util.ArrayList;

import java.util.Locale;

public class ReportController extends MainController {

private double sumServiceMonth;

private MainController mainController;

public ReportController(MainController mainController) {

this.mainController = mainController;

}

public void drawPieChart() {

ObservableList<Payments> tablePayments = database.getTablePayments();

ObservableList<Services> tableServices = database.getTableServices();

int objectId = 0;

ObservableList<ObjectAccounting> tableObject = database.getTableObject();

for (ObjectAccounting obj : tableObject) {

if (obj.getObjectName().equals(mainController.getReportObjCombo().getValue())) {

objectId = obj.getId();

}

}

ArrayList<Payments> payments = new ArrayList<>();

String monthCombo = mainController.getReportMonthCombo().getValue();

String yearCombo = mainController.getReportYearCombo().getValue();

double sum = 0;

Locale local = new Locale("ru", "RU");

if ((monthCombo != null) && (yearCombo != null)) {

for (Payments obj : tablePayments) {

if (obj.getObject() == objectId) {

LocalDate date = obj.getDate().toLocalDate();

String month = String.valueOf(date.getMonth().getDisplayName(TextStyle.FULL\_STANDALONE, local));

if (monthCombo.equals(month)) {

if (yearCombo.equals(String.valueOf(obj.getDate().toLocalDate().getYear()))) {

sum += obj.getPaid();

payments.add(obj);

}

}

}

}

sumServiceMonth = sum;

}

ObservableList<PieChart.Data> pieChartData = FXCollections.observableArrayList();

if (!payments.isEmpty()) {

for (Services services : tableServices) {

double temp = 0;

for (Payments obj : payments) {

if (services.getId() == obj.getService()) {

temp += obj.getPaid();

}

}

if (temp != 0) {

pieChartData.add(new PieChart.Data(services.getServiceName(), temp \* 100 / sum));

}

}

}

mainController.getPieChart().setData(pieChartData);

if (pieChartData.isEmpty()) {

mainController.getPieChart().setTitle("Нет данных за " + monthCombo + ", " + yearCombo);

} else {

mainController.getPieChart().setTitle("Распределение расходов за " + monthCombo + ", " + yearCombo);

final Popup popup = new Popup();

popup.setAutoHide(true);

final Label label = new Label("");

popup.getContent().addAll(label);

for (final PieChart.Data data : pieChartData) {

data.getNode().addEventHandler(MouseEvent.MOUSE\_PRESSED,

e -> {

label.setText(String.valueOf(Math.rint(data.getPieValue())) + "%" +

"\n" + Math.rint(data.getPieValue() / 100 \* sumServiceMonth \* 100) / 100 + " руб.");

popup.setX(e.getScreenX());

popup.setY(e.getScreenY());

popup.show(mainController.getTabPane().getScene().getWindow());

});

}

}

}

public void drawBarChart() {

ObservableList<String> listYear = FXCollections.observableArrayList();

ObservableList<Payments> tablePayments = database.getTablePayments();

ObservableList<Services> tableServices = database.getTableServices();

int objectId = 0;

ObservableList<ObjectAccounting> tableObject = database.getTableObject();

for (ObjectAccounting obj : tableObject) {

if (obj.getObjectName().equals(mainController.getReportObjCombo().getValue())) {

objectId = obj.getId();

}

}

for (Payments obj : tablePayments) {

if (obj.getObject() == objectId) {

if (listYear.indexOf(String.valueOf(obj.getDate().toLocalDate().getYear())) == -1) {

listYear.add(String.valueOf(obj.getDate().toLocalDate().getYear()));

}

}

}

ObservableList<XYChart.Series<String, Double>> barChartData = FXCollections.observableArrayList();

final CategoryAxis xAxis = new CategoryAxis();

final NumberAxis yAxis = new NumberAxis();

xAxis.setLabel("Услуга");

yAxis.setLabel("Сумма");

for (String year : listYear) {

double sum = 0;

XYChart.Series<String, Double> series = new XYChart.Series<>();

series.setName(year);

for (Services service : tableServices) {

for (Payments obj : tablePayments) {

if (year.equals(String.valueOf(obj.getDate().toLocalDate().getYear()))) {

if (obj.getService() == service.getId()) {

if (obj.getObject() == objectId) {

sum += obj.getPaid();

}

}

}

}

if (sum != 0) {

series.getData().add(new XYChart.Data<>(service.getServiceName(), sum));

sum = 0;

}

}

barChartData.add(series);

}

if (barChartData.isEmpty()) {

mainController.getBarChart().setTitle("Нет данных");

} else {

mainController.getBarChart().setTitle("Распределение расходов по годам");

}

mainController.getBarChart().setData(barChartData);

}

}

package by.javafx.communalPayments.controllers;

import by.javafx.communalPayments.controllers.counters.CounterAddController;

import by.javafx.communalPayments.controllers.counters.CounterChangeController;

import by.javafx.communalPayments.controllers.counters.CounterDeleteController;

import by.javafx.communalPayments.controllers.objectAccounting.ObjAddController;

import by.javafx.communalPayments.controllers.objectAccounting.ObjChangeController;

import by.javafx.communalPayments.controllers.objectAccounting.ObjDeleteController;

import by.javafx.communalPayments.controllers.payments.PaymentAddController;

import by.javafx.communalPayments.controllers.payments.PaymentsDeleteController;

import by.javafx.communalPayments.controllers.report.ReportController;

import by.javafx.communalPayments.controllers.services.ServiceAddController;

import by.javafx.communalPayments.controllers.services.ServiceChangeController;

import by.javafx.communalPayments.controllers.services.ServiceDeleteController;

import by.javafx.communalPayments.interfaces.observerInterfaces.Observer;

import by.javafx.communalPayments.representationObjects.\*;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.chart.BarChart;

import javafx.scene.chart.PieChart;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.image.Image;

import javafx.stage.Modality;

import javafx.stage.Stage;

import javafx.stage.Window;

import java.sql.Date;

import java.time.LocalDate;

import java.time.format.TextStyle;

import java.util.Collections;

import java.util.Locale;

public class MainController implements Observer {

protected Database database = new Database(this);

private MyObjects selectedObject;

private ReportController report;

@FXML

private TabPane tabPane;

@FXML

private PieChart pieChart;

@FXML

private BarChart<String, Double> barChart;

@FXML

private ComboBox<String> reportObjCombo;

@FXML

private ComboBox<String> reportMonthCombo;

@FXML

private ComboBox<String> reportYearCombo;

@FXML

private TableView<ObjectAccounting> T1\_objAccounting;

@FXML

private TableColumn<ObjectAccounting, Integer> T1\_personalAccountColumn;

@FXML

private TableColumn<ObjectAccounting, String> T1\_nameObjColumn;

@FXML

private TableColumn<ObjectAccounting, String> T1\_ownerColumn;

@FXML

private TableColumn<ObjectAccounting, String> T1\_addressColumn;

@FXML

private TableColumn<ObjectAccounting, Integer> T1\_residentsColumn;

@FXML

private TableColumn<ObjectAccounting, Double> T1\_areaColumn;

@FXML

private TableView<Counters> T2\_counters;

@FXML

private TableColumn<Counters, Integer> T2\_id\_counterColumn;

@FXML

private TableColumn<Counters, Integer> T2\_nameObjColumn;

@FXML

private TableColumn<Counters, Integer> T2\_serviceColumn;

@FXML

private TableColumn<Counters, String> T2\_counterNameColumn;

@FXML

private TableColumn<Counters, Double> T2\_recentMeasureColumn;

@FXML

private TableView<Services> T3\_service;

@FXML

private TableColumn<Services, Integer> T3\_id\_serviceColumn;

@FXML

private TableColumn<Services, String> T3\_serviceNameColumn;

@FXML

private TableColumn<Services, String> T3\_unitColumn;

@FXML

private TableColumn<Services, Double> T3\_rateColumn;

@FXML

private TableColumn<Services, Integer> T3\_formPaymentsColumn;

@FXML

private TableView<Payments> T4\_payments;

@FXML

private TableColumn<Payments, Integer> T4\_id\_paymentsColumn;

@FXML

private TableColumn<Payments, Integer> T4\_objectColumn;

@FXML

private TableColumn<Payments, Integer> T4\_serviceColumn;

@FXML

private TableColumn<Payments, String> T4\_unitColumn;

@FXML

private TableColumn<Payments, Double> T4\_volumeColumn;

@FXML

private TableColumn<Payments, Double> T4\_rateColumn;

@FXML

private TableColumn<Payments, Double> T4\_accruedColumn;

@FXML

private TableColumn<Payments, Double> T4\_paidColumn;

@FXML

private TableColumn<Payments, Date> T4\_dateColumn;

@FXML

private void initialize() {

database.registerObserver(this);

this.report = new ReportController(this);

T1\_personalAccountColumn.setCellValueFactory(new PropertyValueFactory<>("id"));

T1\_nameObjColumn.setCellValueFactory(new PropertyValueFactory<>("objectName"));

T1\_ownerColumn.setCellValueFactory(new PropertyValueFactory<>("owner"));

T1\_addressColumn.setCellValueFactory(new PropertyValueFactory<>("address"));

T1\_residentsColumn.setCellValueFactory(new PropertyValueFactory<>("residents"));

T1\_areaColumn.setCellValueFactory(new PropertyValueFactory<>("area"));

T2\_id\_counterColumn.setCellValueFactory(new PropertyValueFactory<>("id"));

T2\_nameObjColumn.setCellValueFactory(new PropertyValueFactory<>("object"));

T2\_serviceColumn.setCellValueFactory(new PropertyValueFactory<>("service"));

T2\_counterNameColumn.setCellValueFactory(new PropertyValueFactory<>("counterName"));

T2\_recentMeasureColumn.setCellValueFactory(new PropertyValueFactory<>("recentMeasure"));

T3\_id\_serviceColumn.setCellValueFactory(new PropertyValueFactory<>("id"));

T3\_serviceNameColumn.setCellValueFactory(new PropertyValueFactory<>("serviceName"));

T3\_unitColumn.setCellValueFactory(new PropertyValueFactory<>("unit"));

T3\_rateColumn.setCellValueFactory(new PropertyValueFactory<>("rate"));

T3\_formPaymentsColumn.setCellValueFactory(new PropertyValueFactory<>("formPayments"));

T4\_id\_paymentsColumn.setCellValueFactory(new PropertyValueFactory<>("id"));

T4\_objectColumn.setCellValueFactory(new PropertyValueFactory<>("object"));

T4\_serviceColumn.setCellValueFactory(new PropertyValueFactory<>("service"));

T4\_unitColumn.setCellValueFactory(new PropertyValueFactory<>("unit"));

T4\_volumeColumn.setCellValueFactory(new PropertyValueFactory<>("volume"));

T4\_rateColumn.setCellValueFactory(new PropertyValueFactory<>("rate"));

T4\_accruedColumn.setCellValueFactory(new PropertyValueFactory<>("accrued"));

T4\_paidColumn.setCellValueFactory(new PropertyValueFactory<>("paid"));

T4\_dateColumn.setCellValueFactory(new PropertyValueFactory<>("date"));

reportObjCombo.valueProperty().addListener((observable, oldValue, newValue) -> drawCharts());

reportMonthCombo.valueProperty().addListener((observable, oldValue, newValue) -> drawCharts());

reportYearCombo.valueProperty().addListener((observable, oldValue, newValue) -> drawCharts());

database.setConnection();

fillTables();

chartsInitialize();

}

@FXML

public void objAccountAdd() {

dialogWindow(tabPane.getScene().getWindow(), new ObjAddController(), "/by/javafx/communalPayments/fxml/objAccountDialog/objectAccountAdd.fxml",

"Добавление объекта учета", 565, 350);

}

@FXML

public void objAccountChange() {

ObjectAccounting object = T1\_objAccounting.getSelectionModel().getSelectedItem();

if (object != null) {

setSelectedObject(object);

dialogWindow(tabPane.getScene().getWindow(), new ObjChangeController(this), "/by/javafx/communalPayments/fxml/objAccountDialog/objectAccountAdd.fxml",

"Изменение объекта учета", 565, 350);

} else {

printDialogError("Изменение объекта", "Не выбран изменяемый объект !", "Выберите объект для изменения.");

}

}

@FXML

public void objAccountDelete() {

ObjectAccounting object = T1\_objAccounting.getSelectionModel().getSelectedItem();

if (object != null) {

setSelectedObject(object);

dialogWindow(tabPane.getScene().getWindow(), new ObjDeleteController(this), "/by/javafx/communalPayments/fxml/deleteObject.fxml",

"Удаление объекта", 380, 190);

} else {

printDialogError("Удаление объекта", "Ошибка удаления объекта !", "Не выбран удаляемый объект ! ");

}

}

@FXML

public void countersAdd() {

if (checkAvailabilityObjects() && checkAvailabilityServices() && checkAvailabilityServicesForm()) {

dialogWindow(tabPane.getScene().getWindow(), new CounterAddController(), "/by/javafx/communalPayments/fxml/countersDialog/CountersAdd.fxml",

"Добавление счетчика", 400, 305);

}

}

@FXML

public void countersChange() {

Counters object = T2\_counters.getSelectionModel().getSelectedItem();

if (object != null) {

setSelectedObject(object);

dialogWindow(tabPane.getScene().getWindow(), new CounterChangeController(this), "/by/javafx/communalPayments/fxml/countersDialog/CountersAdd.fxml",

"Изменение счетчика", 400, 305);

} else {

printDialogError("Изменение объекта", "Ошибка изменения объекта !", "Не выбран изменяемый объект ! ");

}

}

@FXML

public void countersDelete() {

Counters object = T2\_counters.getSelectionModel().getSelectedItem();

if (object != null) {

setSelectedObject(object);

dialogWindow(tabPane.getScene().getWindow(), new CounterDeleteController(this), "/by/javafx/communalPayments/fxml/deleteObject.fxml",

"Удаление объекта", 380, 190);

} else {

printDialogError("Удаление объекта", "Ошибка удаления объекта !", "Не выбран удаляемый объект ! ");

}

}

@FXML

public void serviceAdd() {

dialogWindow(tabPane.getScene().getWindow(), new ServiceAddController(), "/by/javafx/communalPayments/fxml/servicesDialog/servicesAdd.fxml",

"Добавление услуги", 450, 305);

}

@FXML

public void serviceChange() {

Services object = T3\_service.getSelectionModel().getSelectedItem();

if (object != null) {

setSelectedObject(object);

dialogWindow(tabPane.getScene().getWindow(), new ServiceChangeController(this), "/by/javafx/communalPayments/fxml/servicesDialog/servicesAdd.fxml",

"Изменение услуги", 450, 305);

} else {

printDialogError("Изменение объекта", "Ошибка изменения объекта!", "Не выбран изменяемый объект ! ");

}

}

@FXML

public void serviceDelete() {

Services object = T3\_service.getSelectionModel().getSelectedItem();

if (object != null) {

setSelectedObject(object);

dialogWindow(tabPane.getScene().getWindow(), new ServiceDeleteController(this), "/by/javafx/communalPayments/fxml/deleteObject.fxml",

"Удаление объекта", 380, 190);

} else {

printDialogError("Удаление объекта", "Ошибка удаления объекта!", "Не выбран удаляемый объект ! ");

}

}

@FXML

public void paymentAdd() {

if (checkAvailabilityObjects() && checkAvailabilityServices()) {

dialogWindow(tabPane.getScene().getWindow(), new PaymentAddController(), "/by/javafx/communalPayments/fxml/paymentsDialog/PaymentsAdd.fxml",

"Добавление платежа", 400, 260);

}

}

@FXML

public void paymentDelete() {

Payments object = T4\_payments.getSelectionModel().getSelectedItem();

if (object != null) {

setSelectedObject(object);

dialogWindow(tabPane.getScene().getWindow(), new PaymentsDeleteController(this), "/by/javafx/communalPayments/fxml/deleteObject.fxml",

"Удаление объекта", 380, 190);

} else {

printDialogError("Удаление объекта", "Ошибка удаления объекта!", "Не выбран удаляемый объект ! ");

}

}

private void fillTables() {

T1\_objAccounting.setItems(database.getTableObject());

T2\_counters.setItems(database.getTableCounters());

T3\_service.setItems(database.getTableServices());

T4\_payments.setItems(database.getTablePayments());

}

private void chartsInitialize() {

ObservableList<ObjectAccounting> tableObject = database.getTableObject();

ObservableList<String> listObjects = FXCollections.observableArrayList();

for (ObjectAccounting obj : tableObject) {

listObjects.add(obj.getObjectName());

}

if (!listObjects.isEmpty()) {

reportObjCombo.setItems(listObjects);

reportObjCombo.setValue(listObjects.get(0));

}

ObservableList<Payments> tablePayments = database.getTablePayments();

ObservableList<String> listMonth = FXCollections.observableArrayList();

ObservableList<String> listYear = FXCollections.observableArrayList();

Locale local = new Locale("ru", "RU");

for (Payments obj : tablePayments) {

LocalDate date = obj.getDate().toLocalDate();

String month = String.valueOf(date.getMonth().getDisplayName(TextStyle.FULL\_STANDALONE, local));

if (listMonth.indexOf(month) == -1) {

listMonth.add(month);

}

if (listYear.indexOf(String.valueOf(date.getYear())) == -1) {

listYear.add(String.valueOf(date.getYear()));

}

}

if (!listMonth.isEmpty()) {

listMonth.sort(Collections.reverseOrder());

reportMonthCombo.setItems(listMonth);

reportMonthCombo.setValue(listMonth.get(0));

}

if (!listYear.isEmpty()) {

Collections.sort(listYear);

reportYearCombo.setItems(listYear);

reportYearCombo.setValue(listYear.get(0));

}

}

private void drawCharts() {

report.drawPieChart();

report.drawBarChart();

}

private boolean checkAvailabilityObjects() {

if (database.getTableObject().isEmpty()) {

printDialogError("Добавление объекта", "Объект не может быть добавлен !",

" Отсутствуют объекты учета.");

return false;

}

return true;

}

private boolean checkAvailabilityServices() {

if (database.getTableServices().isEmpty()) {

printDialogError("Добавление объекта", "Объект не может быть добавлен !",

" Для добавления необходимо наличие услуг.");

return false;

}

return true;

}

private boolean checkAvailabilityServicesForm() {

boolean flag = false;

for (Services obj : database.getTableServices()) {

if (obj.getFormPayments() == 1) {

flag = true;

}

}

if (!flag) {

printDialogError("Добавление объекта", "Объект не может быть добавлен !",

"Отсутствуют услуги с формой оплаты по счетчику.");

return false;

}

return true;

}

protected void dialogWindow(Window window, MainController controller, String resource, String title, int width, int height) {

Stage stage = new Stage();

FXMLLoader fxmlLoader = new FXMLLoader();

fxmlLoader.setLocation(getClass().getResource(resource));

fxmlLoader.setController(controller);

try {

Parent root = fxmlLoader.load();

stage.setTitle(title);

stage.setScene(new Scene(root, width, height));

stage.initModality(Modality.WINDOW\_MODAL);

stage.initOwner(window);

javafx.scene.image.Image ico = new Image("/by/javafx/communalPayments/ico/icon.png");

stage.getIcons().add(ico);

stage.setResizable(false);

stage.show();

} catch (Throwable e) {

printDialogError("Диалоговые окна", "Ошибка создания диалогового окна !", e.getMessage());

}

}

public void printDialogError(String title, String headerText, String contentText) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle(title);

alert.setHeaderText(headerText);

alert.setContentText(contentText);

Stage stage = (Stage) alert.getDialogPane().getScene().getWindow();

stage.getIcons().add(new Image("/by/javafx/communalPayments/ico/icon.png"));

alert.showAndWait();

alert.close();

}

public MyObjects getSelectedObject() {

return selectedObject;

}

private void setSelectedObject(MyObjects object) {

selectedObject = object;

}

public TabPane getTabPane() {

return tabPane;

}

public PieChart getPieChart() {

return pieChart;

}

public BarChart<String, Double> getBarChart() {

return barChart;

}

public ComboBox<String> getReportObjCombo() {

return reportObjCombo;

}

public ComboBox<String> getReportMonthCombo() {

return reportMonthCombo;

}

public ComboBox<String> getReportYearCombo() {

return reportYearCombo;

}

@Override

public void update() {

fillTables();

chartsInitialize();

}

}

package by.javafx.communalPayments.representationObjects;

public class Counters extends MyObjects {

private int id;

private int object;

private int service;

private String counterName;

private double recentMeasure;

public Counters(int id, int object, int service, String counterName, double recentMeasure) {

this.id = id;

this.object = object;

this.service = service;

this.counterName = counterName;

this.recentMeasure = recentMeasure;

}

public Counters() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getObject() {

return object;

}

public void setObject(int object) {

this.object = object;

}

public int getService() {

return service;

}

public void setService(int service) {

this.service = service;

}

public String getCounterName() {

return counterName;

}

public void setCounterName(String counterName) {

this.counterName = counterName;

}

public double getRecentMeasure() {

return recentMeasure;

}

public void setRecentMeasure(double recentMeasure) {

this.recentMeasure = recentMeasure;

}

}

package by.javafx.communalPayments.representationObjects;

import java.sql.Date;

public class Measurement extends MyObjects {

private int id;

private int counter;

private double measure;

private Date date;

public Measurement(int id, int counter, double measure, Date date) {

this.id = id;

this.counter = counter;

this.measure = measure;

this.date = date;

}

public Measurement() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getCounter() {

return counter;

}

public void setCounter(int counter) {

this.counter = counter;

}

public double getMeasure() {

return measure;

}

public void setMeasure(double measure) {

this.measure = measure;

}

public Date getDate() {

return date;

}

public void setDate(Date date) {

this.date = date;

}

}

package by.javafx.communalPayments.representationObjects;

import java.sql.Date;

public class Payments extends MyObjects {

private int id;

private int object;

private int service;

private String unit;

private double volume;

private double rate;

private double accrued;

private double paid;

private Date date;

public Payments(int id, int object, int service, String unit, double volume, double rate, double accrued, double paid, Date date) {

this.id = id;

this.object = object;

this.service = service;

this.unit = unit;

this.volume = volume;

this.rate = rate;

this.accrued = accrued;

this.paid = paid;

this.date = date;

}

public Payments() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getObject() {

return object;

}

public void setObject(int object) {

this.object = object;

}

public int getService() {

return service;

}

public void setService(int service) {

this.service = service;

}

public String getUnit() {

return unit;

}

public void setUnit(String unit) {

this.unit = unit;

}

public double getVolume() {

return volume;

}

public void setVolume(double volume) {

this.volume = volume;

}

public double getRate() {

return rate;

}

public void setRate(double rate) {

this.rate = rate;

}

public double getAccrued() {

return accrued;

}

public void setAccrued(double accrued) {

this.accrued = accrued;

}

public double getPaid() {

return paid;

}

public void setPaid(double paid) {

this.paid = paid;

}

public Date getDate() {

return date;

}

public void setDate(Date date) {

this.date = date;

}

}

package by.javafx.communalPayments.representationObjects;

public class Services extends MyObjects {

private int id;

private String serviceName;

private String unit;

private double rate;

private int formPayments;

public Services(int id, String serviceName, String unit, double rate, int formPayments) {

this.id = id;

this.serviceName = serviceName;

this.unit = unit;

this.rate = rate;

this.formPayments = formPayments;

}

public Services() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getServiceName() {

return serviceName;

}

public void setServiceName(String serviceName) {

this.serviceName = serviceName;

}

public String getUnit() {

return unit;

}

public void setUnit(String unit) {

this.unit = unit;

}

public double getRate() {

return rate;

}

public void setRate(double rate) {

this.rate = rate;

}

public int getFormPayments() {

return formPayments;

}

public void setFormPayments(int formPayments) {

this.formPayments = formPayments;

}

}

package by.javafx.communalPayments.representationObjects;

public class ObjectAccounting extends MyObjects {

private int id;

private String objectName;

private String owner;

private String address;

private int residents;

private double area;

public ObjectAccounting(int id, String objectName, String owner, String address, int residents, double area) {

this.id = id;

this.objectName = objectName;

this.owner = owner;

this.address = address;

this.residents = residents;

this.area = area;

}

public ObjectAccounting() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getObjectName() {

return objectName;

}

public void setObjectName(String objectName) {

this.objectName = objectName;

}

public String getOwner() {

return owner;

}

public void setOwner(String owner) {

this.owner = owner;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public int getResidents() {

return residents;

}

public void setResidents(int residents) {

this.residents = residents;

}

public double getArea() {

return area;

}

public void setArea(double area) {

this.area = area;

}

}

package by.javafx.communalPayments.representationObjects;

public class FormPayments extends MyObjects {

private int id;

private String form;

public FormPayments(int id, String form) {

this.id = id;

this.form = form;

}

public FormPayments() {

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getForm() {

return form;

}

public void setForm(String form) {

this.form = form;

}

}

package by.javafx.communalPayments.representationObjects;

public abstract class MyObjects {

}

package by.javafx.communalPayments.representationObjects;

import by.javafx.communalPayments.connectionImplements.ConnectionBuilderFactory;

import by.javafx.communalPayments.controllers.MainController;

import by.javafx.communalPayments.daoImplements.MySQLDaoFactory;

import by.javafx.communalPayments.interfaces.connectionInterfaces.ConnectionBuilder;

import by.javafx.communalPayments.interfaces.daoInterfaces.\*;

import by.javafx.communalPayments.interfaces.observerInterfaces.Observer;

import by.javafx.communalPayments.interfaces.observerInterfaces.Subject;

import javafx.collections.ObservableList;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.ArrayList;

public class Database implements Subject {

private static ArrayList<Observer> observers = new ArrayList<>();

private static Connection connection;

private MainController mainController;

private DaoFactory daoFactory = new MySQLDaoFactory();

public Database(MainController mainController) {

this.mainController = mainController;

}

public void setConnection() {

ConnectionBuilder connectionBuilder = ConnectionBuilderFactory.getSimpleConnection();

try {

connection = connectionBuilder.getConnection();

} catch (SQLException | ClassNotFoundException e) {

mainController.printDialogError("Ошибка подключения", "Не удалось подключиться к серверу MySQL !", e.getMessage());

System.exit(0);

}

if (!connectionBuilder.createDatabase(connection)) {

mainController.printDialogError("Ошибка подключения", "Не удалось подключиться к БД !", "Ошибка при создании БД !");

System.exit(0);

}

}

public boolean addObject(ObjectAccounting object) {

try {

ObjectAccountDao objectAccountDao = daoFactory.getObjectAccountDao(connection);

objectAccountDao.add(object);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка добавления объекта учета в БД !", e.getMessage());

return false;

}

return true;

}

public boolean addCounter(Counters counter) {

try {

CountersDao countersDao = daoFactory.getCountersDao(connection);

countersDao.add(counter);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка добавления счетчика в БД !", e.getMessage());

return false;

}

return true;

}

public boolean addService(Services service) {

try {

ServicesDao servicesDao = daoFactory.getServicesDao(connection);

servicesDao.add(service);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка добавления услуги в БД !", e.getMessage());

return false;

}

return true;

}

public boolean addPayment(Payments payment) {

try {

PaymentsDao paymentsDao = daoFactory.getPaymentsDao(connection);

paymentsDao.add(payment);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка добавления платежа в БД !", e.getMessage());

return false;

}

return true;

}

public boolean addMeasurement(Measurement measurement) {

try {

MeasurementDao measurementDao = daoFactory.getMeasurementDao(connection);

measurementDao.add(measurement);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка добавления показаний в БД !", e.getMessage());

return false;

}

return true;

}

public boolean changeObject(ObjectAccounting object, int id) {

try {

ObjectAccountDao objectAccountDao = daoFactory.getObjectAccountDao(connection);

objectAccountDao.update(object, id);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка изменения данных в objectAccounting !", e.getMessage());

return false;

}

return true;

}

public boolean changeCounter(Counters counter) {

try {

CountersDao countersDao = daoFactory.getCountersDao(connection);

countersDao.update(counter);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка изменения данных в counters !", e.getMessage());

return false;

}

return true;

}

public boolean changeService(Services service) {

try {

ServicesDao servicesDao = daoFactory.getServicesDao(connection);

servicesDao.update(service);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка изменения данных в services !", e.getMessage());

return false;

}

return true;

}

public boolean lastMeasureChange(Counters object, double lastMeasure) {

try {

MeasurementDao measurementDao = daoFactory.getMeasurementDao(connection);

measurementDao.changeLastMeasure(object, lastMeasure);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка изменения данных в measurement !", e.getMessage());

return false;

}

return true;

}

public boolean deleteObject(ObjectAccounting object) {

try {

ObjectAccountDao objectAccountDao = daoFactory.getObjectAccountDao(connection);

objectAccountDao.delete(object);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка удаления объекта учета из БД !", e.getMessage());

return false;

}

return true;

}

public boolean deleteCounter(Counters counter) {

try {

CountersDao countersDao = daoFactory.getCountersDao(connection);

countersDao.delete(counter);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка удаления счетчика из БД !", e.getMessage());

return false;

}

return true;

}

public boolean deleteService(Services service) {

try {

ServicesDao servicesDao = daoFactory.getServicesDao(connection);

servicesDao.delete(service);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка удаления услуги из БД !", e.getMessage());

return false;

}

return true;

}

public boolean deletePayment(Payments payment) {

try {

PaymentsDao paymentsDao = daoFactory.getPaymentsDao(connection);

paymentsDao.delete(payment);

dataChange();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка удаления платежа из БД !", e.getMessage());

return false;

}

return true;

}

public ObservableList<ObjectAccounting> getTableObject() {

try {

ObjectAccountDao objectAccountDao = daoFactory.getObjectAccountDao(connection);

return objectAccountDao.getAll();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка чтения данных из objectAccounting !", e.getMessage());

return null;

}

}

public ObservableList<Counters> getTableCounters() {

try {

CountersDao countersDao = daoFactory.getCountersDao(connection);

return countersDao.getAll();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка чтения данных из counters !", e.getMessage());

return null;

}

}

public ObservableList<Services> getTableServices() {

try {

ServicesDao servicesDao = daoFactory.getServicesDao(connection);

return servicesDao.getAll();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка чтения данных из services !", e.getMessage());

return null;

}

}

public ObservableList<Payments> getTablePayments() {

try {

PaymentsDao paymentsDao = daoFactory.getPaymentsDao(connection);

return paymentsDao.getAll();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка чтения данных из payments !", e.getMessage());

return null;

}

}

public ObservableList<FormPayments> getTableFormPayments() {

try {

FormPaymentsDao formPaymentsDao = daoFactory.getFormPaymentsDao(connection);

return formPaymentsDao.getAll();

} catch (SQLException e) {

mainController.printDialogError("Работа с базой данных", "Ошибка чтения данных из formPayments !", e.getMessage());

return null;

}

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void removeObserver(Observer o) {

int i = observers.indexOf(o);

if (i >= 0) {

observers.remove(i);

}

}

@Override

public void notifyObserver() {

for (Observer o : observers) {

o.update();

}

}

private void dataChange() {

notifyObserver();

}

}

package by.javafx.communalPayments.daoImplements;

import by.javafx.communalPayments.interfaces.daoInterfaces.CountersDao;

import by.javafx.communalPayments.representationObjects.Counters;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class MySQLCounters implements CountersDao {

private Connection connection;

public MySQLCounters(Connection connection) {

this.connection = connection;

}

@Override

public void add(Counters object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("INSERT INTO counters" +

"(object, service, counterName, recentMeasure) VALUES (?, ?, ?, ?)");

stmt.setInt(1, object.getObject());

stmt.setInt(2, object.getService());

stmt.setString(3, object.getCounterName());

stmt.setDouble(4, object.getRecentMeasure());

stmt.execute();

stmt.close();

}

@Override

public void update(Counters object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("UPDATE counters SET" +

" id=?, object=?, service=?, counterName=?, recentMeasure=?" +

" WHERE id=?");

stmt.setInt(1, object.getId());

stmt.setInt(2, object.getObject());

stmt.setInt(3, object.getService());

stmt.setString(4, object.getCounterName());

stmt.setDouble(5, object.getRecentMeasure());

stmt.setInt(6, object.getId());

stmt.execute();

}

@Override

public void delete(Counters object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("DELETE FROM counters WHERE id=?");

stmt.setInt(1, object.getId());

stmt.execute();

stmt.close();

}

@Override

public ObservableList<Counters> getAll() throws SQLException {

ObservableList<Counters> objectList = FXCollections.observableArrayList();

PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM " + "counters");

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

int id = rs.getInt(1);

int objectId = rs.getInt(2);

int service = rs.getInt(3);

String counterName = rs.getString(4);

double recentMeasure = rs.getDouble(5);

objectList.add(new Counters(id, objectId, service, counterName, recentMeasure));

}

rs.close();

stmt.close();

return objectList;

}

}

package by.javafx.communalPayments.daoImplements;

import by.javafx.communalPayments.interfaces.daoInterfaces.\*;

import java.sql.Connection;

public class MySQLDaoFactory implements DaoFactory {

@Override

public ObjectAccountDao getObjectAccountDao(Connection connection) {

return new MySQLObject(connection);

}

@Override

public CountersDao getCountersDao(Connection connection) {

return new MySQLCounters(connection);

}

@Override

public ServicesDao getServicesDao(Connection connection) {

return new MySQLServices(connection);

}

@Override

public PaymentsDao getPaymentsDao(Connection connection) {

return new MySQLPayments(connection);

}

@Override

public MeasurementDao getMeasurementDao(Connection connection) {

return new MySQLMeasurement(connection);

}

@Override

public FormPaymentsDao getFormPaymentsDao(Connection connection) {

return new MySQLFormPayments(connection);

}

}

package by.javafx.communalPayments.daoImplements;

import by.javafx.communalPayments.interfaces.daoInterfaces.FormPaymentsDao;

import by.javafx.communalPayments.representationObjects.FormPayments;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class MySQLFormPayments implements FormPaymentsDao {

private Connection connection;

public MySQLFormPayments(Connection connection) {

this.connection = connection;

}

@Override

public ObservableList<FormPayments> getAll() throws SQLException {

ObservableList<FormPayments> objectList = FXCollections.observableArrayList();

PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM " + "formpayments");

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

int id = rs.getInt(1);

String form = rs.getString(2);

objectList.add(new FormPayments(id, form));

}

rs.close();

stmt.close();

return objectList;

}

}

package by.javafx.communalPayments.daoImplements;

import by.javafx.communalPayments.interfaces.daoInterfaces.MeasurementDao;

import by.javafx.communalPayments.representationObjects.Counters;

import by.javafx.communalPayments.representationObjects.Measurement;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class MySQLMeasurement implements MeasurementDao {

private Connection connection;

public MySQLMeasurement(Connection connection) {

this.connection = connection;

}

@Override

public void add(Measurement object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("INSERT INTO measurement" +

"(counter, measure, date) VALUES (?, ?, ?)");

stmt.setInt(1, object.getCounter());

stmt.setDouble(2, object.getMeasure());

stmt.setDate(3, object.getDate());

stmt.execute();

stmt.close();

}

@Override

public void changeLastMeasure(Counters object, double lastMeasure) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("UPDATE measurement SET" +

" measure=?" + " WHERE counter=?" + " AND measure=?");

stmt.setDouble(1, object.getRecentMeasure());

stmt.setInt(2, object.getId());

stmt.setDouble(3, lastMeasure);

stmt.execute();

stmt.close();

}

}

package by.javafx.communalPayments.daoImplements;

import by.javafx.communalPayments.interfaces.daoInterfaces.ObjectAccountDao;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class MySQLObject implements ObjectAccountDao {

private Connection connection;

public MySQLObject(Connection connection) {

this.connection = connection;

}

@Override

public void add(ObjectAccounting object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("INSERT INTO accountingobject" +

"(personalAccount, objectName, owner, address, residents, area) VALUES (?, ?, ?, ?, ?, ?)");

stmt.setInt(1, object.getId());

stmt.setString(2, object.getObjectName());

stmt.setString(3, object.getOwner());

stmt.setString(4, object.getAddress());

stmt.setInt(5, object.getResidents());

stmt.setDouble(6, object.getArea());

stmt.execute();

stmt.close();

}

@Override

public void update(ObjectAccounting object, int id) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("UPDATE accountingobject SET" +

" personalAccount=?, objectName=?, owner=?, address=?, residents=?, area=?" +

" WHERE personalAccount=?");

stmt.setInt(1, object.getId());

stmt.setString(2, object.getObjectName());

stmt.setString(3, object.getOwner());

stmt.setString(4, object.getAddress());

stmt.setInt(5, object.getResidents());

stmt.setDouble(6, object.getArea());

stmt.setInt(7, id);

stmt.execute();

stmt.close();

}

@Override

public void delete(ObjectAccounting object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("DELETE FROM accountingobject WHERE personalAccount=?");

stmt.setInt(1, object.getId());

stmt.execute();

stmt.close();

}

@Override

public ObservableList<ObjectAccounting> getAll() throws SQLException {

ObservableList<ObjectAccounting> objectList = FXCollections.observableArrayList();

PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM " + "accountingobject");

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

int id = rs.getInt(1);

String objectName = rs.getString(2);

String owner = rs.getString(3);

String address = rs.getString(4);

int residents = rs.getInt(5);

double area = rs.getDouble(6);

objectList.add(new ObjectAccounting(id, objectName, owner, address, residents, area));

}

rs.close();

stmt.close();

return objectList;

}

}

package by.javafx.communalPayments.daoImplements;

import by.javafx.communalPayments.interfaces.daoInterfaces.PaymentsDao;

import by.javafx.communalPayments.representationObjects.Payments;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.sql.\*;

public class MySQLPayments implements PaymentsDao {

private Connection connection;

public MySQLPayments(Connection connection) {

this.connection = connection;

}

@Override

public void add(Payments object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("INSERT INTO payments" +

"(object, service, unit, volume, rate, accrued, paid, date) VALUES (?, ?, ?, ?, ?, ?, ?, ?)");

stmt.setInt(1, object.getObject());

stmt.setInt(2, object.getService());

stmt.setString(3, object.getUnit());

stmt.setDouble(4, object.getVolume());

stmt.setDouble(5, object.getRate());

stmt.setDouble(6, object.getAccrued());

stmt.setDouble(7, object.getPaid());

stmt.setDate(8, object.getDate());

stmt.execute();

stmt.close();

}

@Override

public void delete(Payments object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("DELETE FROM payments WHERE id=?");

stmt.setInt(1, object.getId());

stmt.execute();

stmt.close();

}

@Override

public ObservableList<Payments> getAll() throws SQLException {

ObservableList<Payments> objectList = FXCollections.observableArrayList();

PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM " + "payments");

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

int id = rs.getInt(1);

int objectId = rs.getInt(2);

int service = rs.getInt(3);

String unit = rs.getString(4);

double volume = rs.getDouble(5);

double rate = rs.getDouble(6);

double accrued = rs.getDouble(7);

double paid = rs.getDouble(8);

Date date = rs.getDate(9);

objectList.add(new Payments(id, objectId, service, unit, volume, rate, accrued, paid, date));

}

rs.close();

stmt.close();

return objectList;

}

}

package by.javafx.communalPayments.daoImplements;

import by.javafx.communalPayments.interfaces.daoInterfaces.ServicesDao;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class MySQLServices implements ServicesDao {

private Connection connection;

public MySQLServices(Connection connection) {

this.connection = connection;

}

@Override

public void add(Services object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("INSERT INTO services" +

"(serviceName, unit, rate, formPayment) VALUES (?, ?, ?, ?)");

stmt.setString(1, object.getServiceName());

stmt.setString(2, object.getUnit());

stmt.setDouble(3, object.getRate());

stmt.setInt(4, object.getFormPayments());

stmt.execute();

stmt.close();

}

@Override

public void update(Services object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("UPDATE services SET" +

" id=?, serviceName=?, unit=?, rate=?, formPayment=?" +

" WHERE id=?");

stmt.setInt(1, object.getId());

stmt.setString(2, object.getServiceName());

stmt.setString(3, object.getUnit());

stmt.setDouble(4, object.getRate());

stmt.setInt(5, object.getFormPayments());

stmt.setInt(6, object.getId());

stmt.execute();

stmt.close();

}

@Override

public void delete(Services object) throws SQLException {

PreparedStatement stmt = connection.prepareStatement("DELETE FROM services WHERE id=?");

stmt.setInt(1, object.getId());

stmt.execute();

stmt.close();

}

@Override

public ObservableList<Services> getAll() throws SQLException {

ObservableList<Services> objectList = FXCollections.observableArrayList();

PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM " + "services");

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

int idService = rs.getInt(1);

String serviceName = rs.getString(2);

String unit = rs.getString(3);

double rate = rs.getDouble(4);

int formPayments = rs.getInt(5);

objectList.add(new Services(idService, serviceName, unit, rate, formPayments));

}

rs.close();

stmt.close();

return objectList;

}

}

package by.javafx.communalPayments.interfaces.daoInterfaces;

import by.javafx.communalPayments.representationObjects.Counters;

import javafx.collections.ObservableList;

import java.sql.SQLException;

public interface CountersDao {

void add(Counters object) throws SQLException;

void update(Counters object) throws SQLException;

void delete(Counters object) throws SQLException;

ObservableList<Counters> getAll() throws SQLException;

}

package by.javafx.communalPayments.interfaces.daoInterfaces;

import by.javafx.communalPayments.representationObjects.FormPayments;

import javafx.collections.ObservableList;

import java.sql.SQLException;

public interface FormPaymentsDao {

ObservableList<FormPayments> getAll() throws SQLException;

}

package by.javafx.communalPayments.interfaces.daoInterfaces;

import by.javafx.communalPayments.representationObjects.ObjectAccounting;

import javafx.collections.ObservableList;

import java.sql.SQLException;

public interface ObjectAccountDao {

void add(ObjectAccounting object) throws SQLException;

void update(ObjectAccounting object, int id) throws SQLException;

void delete(ObjectAccounting object) throws SQLException;

ObservableList<ObjectAccounting> getAll() throws SQLException;

}

package by.javafx.communalPayments.interfaces.daoInterfaces;

import by.javafx.communalPayments.representationObjects.Services;

import javafx.collections.ObservableList;

import java.sql.SQLException;

public interface ServicesDao {

void add(Services object) throws SQLException;

void update(Services object) throws SQLException;

void delete(Services object) throws SQLException;

ObservableList<Services> getAll() throws SQLException;

}

package by.javafx.communalPayments.interfaces.daoInterfaces;

import by.javafx.communalPayments.representationObjects.Payments;

import javafx.collections.ObservableList;

import java.sql.SQLException;

public interface PaymentsDao {

void add(Payments object) throws SQLException;

void delete(Payments object) throws SQLException;

ObservableList<Payments> getAll() throws SQLException;

}

package by.javafx.communalPayments.interfaces.daoInterfaces;

import by.javafx.communalPayments.representationObjects.Counters;

import by.javafx.communalPayments.representationObjects.Measurement;

import java.sql.SQLException;

public interface MeasurementDao {

void add(Measurement object) throws SQLException;

void changeLastMeasure(Counters object, double lastMeasure) throws SQLException;

}

package by.javafx.communalPayments.interfaces.daoInterfaces;

import java.sql.Connection;

public interface DaoFactory {

ObjectAccountDao getObjectAccountDao(Connection connection);

CountersDao getCountersDao(Connection connection);

ServicesDao getServicesDao(Connection connection);

PaymentsDao getPaymentsDao(Connection connection);

MeasurementDao getMeasurementDao(Connection connection);

FormPaymentsDao getFormPaymentsDao(Connection connection);

}

package by.javafx.communalPayments.interfaces.observerInterfaces;

public interface Observer {

void update();

}

package by.javafx.communalPayments.interfaces.observerInterfaces;

public interface Subject {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObserver();

}

package by.javafx.communalPayments.interfaces.connectionInterfaces;

import java.sql.Connection;

import java.sql.SQLException;

public interface ConnectionBuilder {

Connection getConnection() throws SQLException, ClassNotFoundException;

boolean createDatabase(Connection connection);

}

package by.javafx.communalPayments.connectionImplements;

import by.javafx.communalPayments.interfaces.connectionInterfaces.ConnectionBuilder;

public class ConnectionBuilderFactory {

public static ConnectionBuilder getSimpleConnection() {

return new MySQLSimpleConnection();

}

}

package by.javafx.communalPayments.connectionImplements;

import by.javafx.communalPayments.interfaces.connectionInterfaces.ConnectionBuilder;

import java.sql.\*;

import java.util.Properties;

public class MySQLSimpleConnection implements ConnectionBuilder {

@Override

public Connection getConnection() throws SQLException, ClassNotFoundException {

Class.forName("com.mysql.jdbc.Driver");

Properties properties = new Properties();

properties.setProperty("user", "root");

properties.setProperty("password", "root");

properties.setProperty("useUnicode", "true");

properties.setProperty("characterEncoding", "UTF-8");

return DriverManager.getConnection("jdbc:mysql://localhost:3306", properties);

}

@Override

public boolean createDatabase(Connection connection) {

try {

connection.setAutoCommit(false);

PreparedStatement stmt = connection.prepareStatement("CREATE SCHEMA IF NOT EXISTS `communalpayments`");

stmt.execute();

stmt = connection.prepareStatement("USE `communalpayments`");

stmt.execute();

stmt = connection.prepareStatement("CREATE TABLE IF NOT EXISTS `accountingobject` (" +

"`personalAccount` INT(11) NOT NULL," +

"`objectName` VARCHAR(50) NOT NULL," +

"`owner` VARCHAR(255) DEFAULT NULL," +

"`address` VARCHAR(255) DEFAULT NULL," +

"`residents` INT(11) DEFAULT NULL," +

"`area` DOUBLE DEFAULT NULL," +

"PRIMARY KEY (`personalAccount`)," +

"UNIQUE KEY `objectName\_UNIQUE` (`objectName`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;");

stmt.execute();

stmt = connection.prepareStatement("CREATE TABLE IF NOT EXISTS `formpayments` (" +

"`id` INT(11) NOT NULL AUTO\_INCREMENT," +

"`form` VARCHAR(255) NOT NULL," +

"PRIMARY KEY (`id`)) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;");

stmt.execute();

stmt = connection.prepareStatement("SELECT 1 FROM formpayments;");

ResultSet rs = stmt.executeQuery();

if (!rs.next()) {

stmt = connection.prepareStatement("INSERT INTO formpayments(id, form) VALUES (?, ?) ");

stmt.setInt(1, 1);

stmt.setString(2, "По счетчику");

stmt.execute();

stmt = connection.prepareStatement("INSERT INTO formpayments(id, form) VALUES (?, ?)");

stmt.setInt(1, 2);

stmt.setString(2, "По площади");

stmt.execute();

stmt = connection.prepareStatement("INSERT INTO formpayments(id, form) VALUES (?, ?)");

stmt.setInt(1, 3);

stmt.setString(2, "По количеству жильцов");

stmt.execute();

}

stmt = connection.prepareStatement("CREATE TABLE IF NOT EXISTS `services` (" +

"`id` INT(11) NOT NULL AUTO\_INCREMENT," +

"`serviceName` VARCHAR(255) NOT NULL," +

"`unit` VARCHAR(20) NOT NULL," +

"`rate` DOUBLE DEFAULT NULL," +

"`formPayment` INT(11) NOT NULL," +

"PRIMARY KEY (`id`)," +

"UNIQUE KEY `serviceName\_UNIQUE` (`serviceName`)," +

"KEY `services-formPayments\_idx` (`formPayment`)," +

"CONSTRAINT `services-formPayments` FOREIGN KEY (`formPayment`) REFERENCES `formpayments` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) " +

"ENGINE=InnoDB AUTO\_INCREMENT=14 DEFAULT CHARSET=utf8;");

stmt.execute();

stmt = connection.prepareStatement("CREATE TABLE IF NOT EXISTS `counters` (" +

"`id` INT(11) NOT NULL AUTO\_INCREMENT," +

"`object` INT(11) NOT NULL," +

"`service` INT(11) NOT NULL," +

"`counterName` VARCHAR(50) NOT NULL," +

"`recentMeasure` DOUBLE NOT NULL," +

"PRIMARY KEY (`id`)," +

"UNIQUE KEY `counterName\_UNIQUE` (`counterName`)," +

"KEY `counters-object\_idx` (`object`)," +

"KEY `counters-services\_idx` (`service`)," +

"CONSTRAINT `counters-object` FOREIGN KEY (`object`) REFERENCES `accountingobject` (`personalAccount`) ON DELETE CASCADE ON UPDATE CASCADE," +

"CONSTRAINT `counters-services` FOREIGN KEY (`service`) REFERENCES `services` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) " +

"ENGINE=InnoDB AUTO\_INCREMENT=23 DEFAULT CHARSET=utf8;");

stmt.execute();

stmt = connection.prepareStatement("CREATE TABLE IF NOT EXISTS `measurement` (" +

"`id` INT(11) NOT NULL AUTO\_INCREMENT," +

"`counter` INT(11) NOT NULL," +

"`measure` DOUBLE NOT NULL," +

"`date` DATE NOT NULL," +

"PRIMARY KEY (`id`)," +

"KEY `measurement-counters\_idx` (`counter`)," +

"CONSTRAINT `measurement-counters` FOREIGN KEY (`counter`) REFERENCES `counters` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) " +

"ENGINE=InnoDB AUTO\_INCREMENT=75 DEFAULT CHARSET=utf8;");

stmt.execute();

stmt = connection.prepareStatement("CREATE TABLE IF NOT EXISTS `payments` (" +

"`id` INT(11) NOT NULL AUTO\_INCREMENT," +

"`object` INT(11) NOT NULL," +

"`service` INT(11) NOT NULL," +

"`unit` VARCHAR(20) NOT NULL," +

"`volume` DOUBLE NOT NULL," +

"`rate` DOUBLE NOT NULL," +

"`accrued` DOUBLE NOT NULL," +

"`paid` DOUBLE NOT NULL," +

"`date` DATE NOT NULL," +

"PRIMARY KEY (`id`)," +

"KEY `payments-object\_idx` (`object`)," +

"KEY `payments-service\_idx` (`service`)," +

"CONSTRAINT `payments-object` FOREIGN KEY (`object`) REFERENCES `accountingobject` (`personalAccount`) ON DELETE CASCADE ON UPDATE CASCADE," +

"CONSTRAINT `payments-service` FOREIGN KEY (`service`) REFERENCES `services` (`id`) ON DELETE CASCADE ON UPDATE CASCADE) " +

"ENGINE=InnoDB AUTO\_INCREMENT=36 DEFAULT CHARSET=utf8;");

stmt.execute();

connection.commit();

connection.setAutoCommit(true);

stmt.close();

} catch (SQLException e) {

return false;

}

return true;

}

}