|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

**КУРСОВАЯ РАБОТА**

по дисциплине «Объектно-ориентированное программирование»

**Тема курсовой работы** Проектирование и реализация программной системы с использованием объектного подхода

**Студент группы** ИКБО-07-17 Акжигитов Радмир Русланович

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

(подпись студента)

**Руководитель курсовой работы** ст. преподаватель Гусев К.В.

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

(подпись руководителя)

Работа представлена к защите «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Допущен к защите «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Москва 2018

|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

Утверждаю

Заведующий кафедрой МОСИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Головин С.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2018 г.

**ЗАДАНИЕ**

**на выполнение курсовой работы по дисциплине**

«Объектно-ориентированное программирование»

Студент Акжигитов Радмир Русланович Группа ИКБО-07-17

**Тема работы:** Проектирование и реализация программной системы с использованием объектного подхода

**Исходные данные:** задание на курсовую работу,тема №1, вариант № 10. Разработать систему Личный кабинет квартиросъемщика. Служба госуслуг каждый месяц выставляет счет по оплате коммунальных услуг: за расход горячей и холодной воды, за эксплуатацию помещения (площади), взнос на капитальный ремонт. Объем расхода горячей и холодной воды рассчитываются на основе показателей счетчиков учета расхода воды, которые квартиросъемщик обязан отправить до 20 числа текущего месяца в единый расчетный центр госуслуг.

**Функционал:** регистрация квартиросъемщика, с указанием логина, пароля, Фамилии, Имени, Отчества, полного адреса, общую площадь квартиры, количество проживающих в квартире, вид льгот по оплате коммунальных слуг квартиросемщика; вход в созданный личный кабинет по логину и паролю; ввод показаний счетчиков воды за текущий месяц; формирование службой единого центра счета к плате за текущий месяц; отображение всех плат за текущий год; сохранение данных истекшего года в файле с именем Номер года; ведение нормативов на оплату каждого вида услуг.

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

Моделирование объектной системы с использованием обозначений языка моделирования UML

Реализация программной системы средствами языка С++

Отчет по курсовой работе в виде расчетно-пояснительной записки .

**Срок представления к защите курсовой работы:** до «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

**Задание на курсовую работу выдал** \_\_\_\_\_\_\_\_\_\_\_\_ (Гусев К.В.)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

**Задание на курсовую работу получил**  \_\_\_\_\_\_\_\_\_\_\_\_(Акжигитов Р. Р.)

СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 4](#_Toc515919591)

[РЕФЕРАТ 6](#_Toc515919592)

[Общее количество картинок, таблиц, разделов 6](#_Toc515919593)

[Краткий обзор содержания разделов 6](#_Toc515919594)

[ВВЕДЕНИЕ 7](#_Toc515919595)

[ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ 8](#_Toc515919596)

[Объектно-ориентированное программирование 8](#_Toc515919597)

[Основные понятия 8](#_Toc515919598)

[Описание UML 10](#_Toc515919599)

[Что такое UML 10](#_Toc515919600)

[Структура диаграммы класса 10](#_Toc515919601)

[Атрибуты, методы и обозначение доступа 11](#_Toc515919602)

[Наследование и интерфейсы 11](#_Toc515919603)

[Композиция: агрегация и ассоциация 12](#_Toc515919604)

[Кардинальность 14](#_Toc515919605)

[Метод создания интерфейсов 14](#_Toc515919606)

[JSON 15](#_Toc515919607)

[ПРОЕКТНАЯ ЧАСТЬ 17](#_Toc515919608)

[Постановка задачи 17](#_Toc515919609)

[Необходимый функционал 17](#_Toc515919610)

[Список выявленных понятий 17](#_Toc515919611)

[Список понятий, претендующих на класс 18](#_Toc515919612)

[Проектирование программного обеспечения 18](#_Toc515919613)

[Отношения между классами 18](#_Toc515919614)

[Атрибуты и методы класса 19](#_Toc515919615)

[Проектирование методов класса 20](#_Toc515919616)

[Диаграмма объектов UML 21](#_Toc515919617)

[ЭКСПЕРЕМЕНТАЛЬНАЯ ЧАСТЬ 23](#_Toc515919618)

[Решение функциональных задач 23](#_Toc515919619)

[Графический интерфейс 25](#_Toc515919620)

[Руководство пользователя 25](#_Toc515919621)

[Тестирование 26](#_Toc515919622)

[ЗАКЛЮЧЕНИЕ 30](#_Toc515919623)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 31](#_Toc515919624)

[ПРИЛОЖЕНИЯ 32](#_Toc515919625)

[Global.h 32](#_Toc515919626)

[Tenant.h 32](#_Toc515919627)

[Tenant.cpp 33](#_Toc515919628)

[Landlord.h 34](#_Toc515919629)

[Landlord.cpp 35](#_Toc515919630)

[Database.h 37](#_Toc515919631)

[Database.cpp 37](#_Toc515919632)

[Account.h 38](#_Toc515919633)

[Account.cpp 39](#_Toc515919634)

[MyForm.h 40](#_Toc515919635)

[MyForm.cpp 52](#_Toc515919636)

РЕФЕРАТ

# Общее количество картинок, таблиц, разделов

Общее количество картинок: 9

Общее количество таблиц: 1

Количество разделов: 7

# Краткий обзор содержания разделов

**Введение**. В данном разделе описаны цели, задачи для данной курсовой работы, актуальность выбранной проблемы.

**Теоретическое введение**. В этой части описаны программные решения, библиотеки и парадигмы программирования, которые были использованы в работе, описан визуальный унифицированный язык UML.

**Проектная часть**. В этом разделе ведется объяснение принятым решениям по объектной декомпозиции программы. Выделяются основные понятия, функции. Описывается концептуальная модель.

**Экспериментальная часть**. Здесь проводится тестирование написанной программы. Расписаны основные исключения. Описывается графический интерфейс. Содержится краткое руководство пользователя.

**Заключение**. Приводятся итоги работы, описаны достигнутые цели.

**Список использованной литературы**. Обращения к литературным и интернет источникам.

**Приложения**. Приводится код программы без авто-генерированной части.

ВВЕДЕНИЕ

Тема, выбранная мной, актуальна, так как ее реализация позволит соединить центры гос. услуг и квартирантов (квартиросъемщиков). Людей, состоящих в этих отношениях, очень много, ведь все жильцы платят коммунальные услуги, оплачивают ремонт. Автоматизация этого процесса – востребованная и тяжелая задача.

Цель данной курсовой работы состоит в разработке программного продукта, способного решить или упростить задачу сбора данных с собственников квартир, вести временную статистику, отправленных платежей, дать возможность вносить правки в услуги (название, тариф и т.д.).

Реализация программной части на языке C++ в среде разработки Microsoft Visual Studio 2017.

В ходе курсовой работы были спроектированы модули и их функции, отвечающие за каждую часть программы. Было спроектировано взаимодействие классов между собой.

Данная курсовая работа преследует за собой цели:

1. Обучение элементам технологии создания программного продукта и применение их на практике;
2. Использование объектно-ориентированного подхода в ходе решения прикладной задачи;
3. Оформление документации для программного обеспечения.

Данная работа демонстрирует создание и работу с собственными программными объектами на примере взаимоотношений собственников и портала гос. услуг.

ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

# Объектно-ориентированное программирование

Объектно-ориентированное программирование – один из подходов программирования, который включает в себя такие понятия как: класс, объект, взаимоотношения между классами и объектами, основной принцип: разбиение сложного механизма (всего алгоритма программы) на логические составные части (классы) и наделение объектов класса методами, присущими только им.

Основные понятия

Объект (экземпляр класса) - сущность, которая обладает набором свойств, определенными в классе и поведением, описанным в классе. Объектно-ориентированная программа включает себя множество взаимодействующих определенным образом объектов.

Класс – трафарет (прототип) объекта, содержащий все методы и атрибуты, которые получат будущие объекты. Класс можно назвать типом данных более высокого уровня, так как он будет в объявлении своем содержать базовые типы и функции взаимодействия.

Инкапсуляция – свойство системы, состоящее в том, что объект скрывает внутренние данные и алгоритмы поведения (детали) от других объектов, предоставляет их через интерфейсы взаимодействия. Надежные классы всегда проектируются с учетом инкапсуляции.

Интерфейс – основные средства коммуникации между объектами. Любой метод, который содержится в объекте, должен вызываться через интерфейс. Объект, содержащий атрибут, управляет доступом к нему. Это вопрос безопасности данных. Если мы предусмотрим установку атрибутов, только в одном месте, с помощью сеттера, то нам не придется отслеживать каждый фрагмент кода, который мог изменить соответствующее поле. Особое значение следует уделить таким моментам, как передача личной информации и паролей.

Наследование – свойство системы, позволяющее описать новый «внутренний» (дочерний) класс на базе существующего «внешнего» класса (базового, родительского). Дочерний класс обладает полями и функциями производного (полностью или частично, используя модификаторы доступа), а также может иметь свои собственные атрибуты и методы. Этот принцип позволяет создавать абсолютно новые классы путем абстрагирования общих атрибутов и поведений. Создав подкласс, мы можем использовать его в качестве суперкласса и создавать классы наследников уже от него.

У класса может иметься только один базовый, но много дочерних – в таком случае, это простое наследование, если родителей класса может быть несколько, то это множественное наследование.

Пример использования наследования классов (Рисунок 1): базовый класс Mammal (млекопитающее) имеет 2 подкласса: Dog (собака) и Cat (кошка), в свою очередь класс Dog может включать в себя подвиды собак: German Shepherd и Poodle. Каждый уровень ниже наследует все параметры (атрибуты и методы) высших уровней.

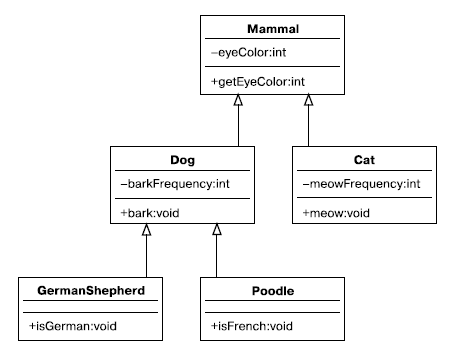


Рисунок - Пример наследования

Полиморфизм – свойство системы, которое позволяет разным объектам по-разному «реагировать» на одно и то же сообщение. Множественное поведение на один и тот же интерфейс у объектов, у которых нам не обязательно знать их внутреннюю структуру и типы данных.

Композиция – возможность одних объектов содержать множество других объектов. Например, компьютер – это сложный объект, он содержит жесткий диск, который тоже является объектом.

# Описание UML

Что такое UML

UML – унифицированный графический язык для моделирования, визуализации и конструирования главных сущностей (абстрактной модели) и связей в архитектурной части программного обеспечения.

Объектное моделирование – является частью языка UML.

Структура диаграммы класса

Диаграмма класса включает в себя 3 части: имя класса, атрибуты и методы (конструкторы – тоже методы). Класс, отображенный на диаграмме является прямоугольником, который разделен горизонтальными линиями на 3 вышеописанные части. Пример ниже (Рисунок 2).

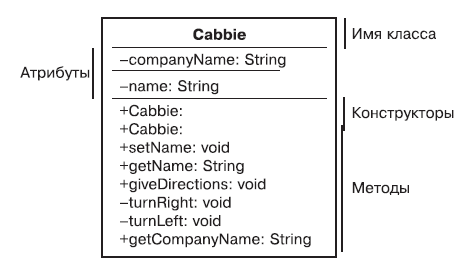


Рисунок – Компоненты диаграммы класса

Атрибуты, методы и обозначение доступа

Для отображения атрибутов и методов на диаграмме классов необходимо написать:

1. Модификатор доступа, в виде «+» - public (поле видно в классе, вне класса, доступно для наследников), «-» - private (к полю можно обращаться только в классе, недоступно для наследников), «#» - protected (поле видно в классе и для наследников).
2. Название поля (атрибута, функции), также можно включить список параметров.
3. Тип поля (тип свойства или возвращаемый тип метода). Для конструкторов тип не указывают.

Пример:

+Cabbie – конструктор

+giveDirections(param1, param2) : void – public метод

-processSecretInformation() : String – private метод, возвращающий String.

Наследование и интерфейсы

Рассмотрим различные виды возможных связей между несколькими объектами. Так как наследование и интерфейсы тесно связаны и их отображение сильно похоже рассмотрим их вместе.

На диаграмме классов наследование представляется пустой сплошной стрелкой, указывающей от дочернего класса к родительскому (Рисунок 3).

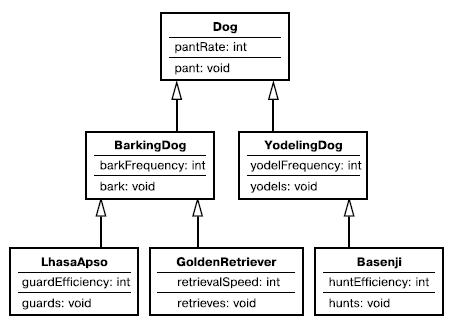


Рисунок – Пример наследования UML

Отношение интерфейс объекта обозначается пустой штриховой стрелкой (Рисунок 4).

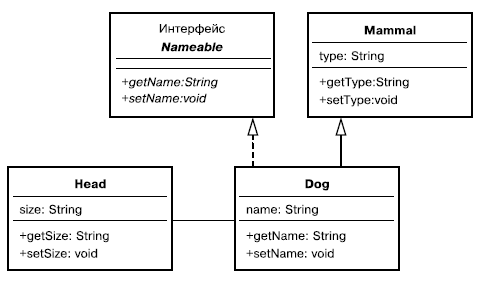


Рисунок – UML-отношение интерфейса

Композиция: агрегация и ассоциация

Композиция – вариант использования отношения «содержит как часть» («является экземпляром» (наследование) не подходит при проектировании).

При композиции класс создается с применением других классов.

Есть два варианта композиции – агрегация и ассоциация. При агрегации класс является частью другого класса (покрышка – автомобиль, покрышка существует только совместно с классом автомобиль (Рисунок 5)). На диаграмме отмечается сплошной линией с пустым «ромбиком» на конце (в направлении совокупляющего класса). Существует строгий вид агрегации – композиция, при которой один объект жестко связан с другим, объект не может существовать без своего класса, к которому он прикреплен. Также ограничен временем жизни внешнем классом. На схеме отображается «закрашенным» ромбиком.

При ассоциации классы взаимодействуют между собой, но могут существовать друг без друга (не являются частью друг друга). Пример, клиент – сервер, существуют отдельно, но обмениваются друг с другом информацией (Рисунок 6). Отображается сплошной линией. Направление отсутствует.

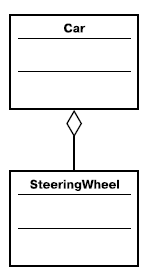


Рисунок – UML-отношение агрегации

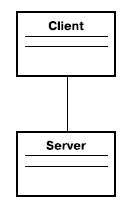


Рисунок – UML-отношение ассоциации

Кардинальность

Кардинальность – диапазон объектов (количество возможных связей), соответствующих определенному классу. Например, в состав компьютера должна входить одна и только одна материнская плата (кардинальность 1 к 1). Однако планок оперативной памяти можно установить несколько, минимум одну (1 компьютер к 1..\* или + хотя бы одной планке памяти).

Предельные значения кардинальности. Если известно, что в данной плане предусмотрено 4 слота для установки памяти, но вместо \* или N, у нас будет число 4 (1 компьютер к 1..4 планок).

Кардинальность может иметь оба «направления» действия. Пример: 1 человек может работать на 0..\* компаний, однако на 1 компанию могут работать 1..\* человек.

# Метод создания интерфейсов

Существует два разных способа создания интерфейсов:

1. Описание графических элементов на языке верстки. Соединение написанного макета с кодом программы. Пример: python-библиотека Kivy App с языком верстки интерфейса YAML. Этот способ менее нагляден, но позволяет гибко настроить пользовательские элементы, имеет множество вариантов комбинирования элементов в контейнерах.
2. Конструирование интерфейса в графическом редакторе. Данный метод позволяет сразу увидеть будущий результат. Пример: winForms на С# и QT C++.

В данной работе я буду использовать WinForms на С++.

Редактор состоит из множества элементов управления, такие как кнопки, текстовые поля, файловый проводник и др. Каждый элемент имеет свои свойства такие как: размер, положение на форме, цвет, текст, отображаемый на элементе.

Также любой элемент имеет свои события: нажатие ЛКМ и ПКМ, наведение курсора на элемент, перетаскивание файла и др. На событие можно установить пользовательский обработчик (функцию), который выполнится при происшествии события. В функции обработчике можно изменять свойства любого элемента управления на расположенного на форме.

Эта среда вполне подходит для данной задачи.

# JSON

JSON – формат, используемый для передачи данных в виде текста, в основе лежит нотация объектов языка JavaScript, не зависит от языка, так как имеет парсеры почти на любой язык, является легким для понимания и простым для прочтения.

Его синтаксис очень прост (Рисунок 7). Объект заключается в фигурные скобки. Содержимое – множество ключей, значения которых могут быть: строкой, числом, логической константой, массивом, другим объектом. Формат поддерживает сколь угодно большую вложенность, что позволяет описывать и передавать объекты любой сложности. Представляет собой ассоциативный массив.

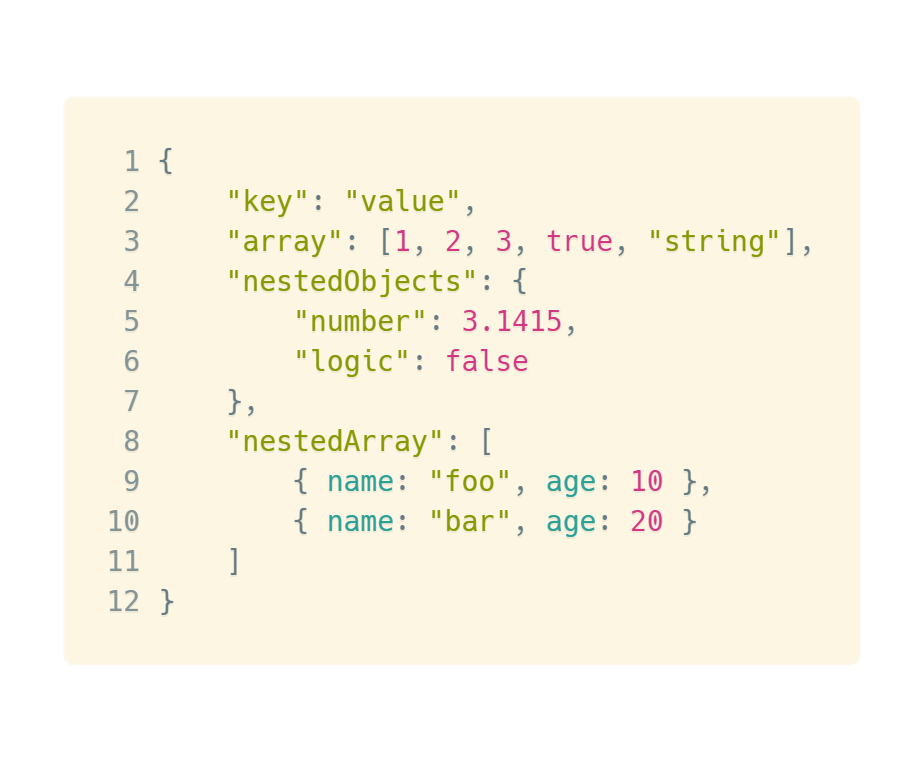


Рисунок – Синтаксис JSON

ПРОЕКТНАЯ ЧАСТЬ

# Постановка задачи

Квартиросъемщик должен платить за услуги каждый месяц (вода, газ, ремонт, аренда). Ему приходит запрос от портала гос. услуг, на который он должен до 20 числа каждого месяца отправить показания счетчиков, если они нужны. Центр гос. услуг в ответ отправляет квитанцию на оплату, посчитанную специальным образом, в зависимости по какому критерию рассчитывается стоимость услуги: по показаниям счетчика, по нормативу, по площади жилья. Квартирант должен оплатить все услуги, только затем он может переключиться на следующий месяц, и цикл повторяется.

Необходимый функционал

1. Квартиросъемщик ОТПРАВЛЯЕТ показания в портал услуг, ОПЛАЧИВАЕТ счета, полученные от портала.
2. Портал услуг ПОЛУЧАЕТ показания собственника, ВЫСЧИТЫВАЕТ стоимости и ОТПРАВЛЯЕТ запрос на оплату обратно.
3. Портал услуг может также ИЗМЕНЯТЬ, ДОБАВЛЯТЬ и УДАЛЯТЬ услуги к оплате.
4. ВЕДЕНИЕ порталом услуг статистики платежей, СОХРАНЕНИЕ устаревших платежей в файл.
5. РЕГИСТРАЦИЯ и АВТОРИЗАЦИЯ в базе данных новых пользователей.

Список выявленных понятий

1. Квартиросъемщик (пользователь, клиент)
2. Портал гос. услуг (сервер)
3. Отчет по показаниям
4. Платеж пользователя
5. База данных пользователей
6. Личный кабинет пользователя

Список понятий, претендующих на класс

1. Пользователь (квартиросъемщик), так как он объединяет в себе множество полей личной информации и должен исполнять набор функции взаимодействия с порталом.
2. Портал (арендодатель), так как он взаимодействует с пользователем, принимает платежи, контролирует нормативы оплаты.
3. База данных пользователей, так как это интерфейс работы с множеством пользователей, получение пользователя, удаление, сохранение в файл и т.д.
4. Личный кабинет, объединяет в себе авторизованного в данный момент времени пользователя, единый портал и базу данных. Является посредником между графической формой и классами. Подключается к базе данных, создает запросы на регистрацию и авторизацию. Работает с учетной записью.

# Проектирование программного обеспечения

Отношения между классами

Личный кабинет ВКЛЮЧАЕТ в себя: пользователя, портал услуг, базу данных.

Пользователь ОПРЕДЕЛЯЕТСЯ из базы данных (через логин и пароль) и СОХРАНЯЕТСЯ в нее при регистрации.

Пользователь и портал услуг ОБМЕН информацией.

Атрибуты и методы класса

**Пользователь (Tenant):**

**Атрибуты:** json пользователя, который содержит все данные о нем: авторизационные данные, личная информация, какие услуги собственнику необходимо платить.

**Методы:** получение массива услуг, представленных к оплате, отправка показаний счетчиков, оплата, полученного от портала счета.

**Портал услуг (Landlord):**

**Атрибуты:** json, хранящий все платежи.

**Методы:** получение счета к оплате, подтверждение оплаты (сохранение платежа), получение следующего неоплаченного месяца для пользователя, сохранение всех платежей, сохранение просроченных платежей.

**База данных пользователя (Database):**

**Атрибуты:** json, в нем находится вся база

**Методы:** считывание JSON в базу данных (файла по имени), получение пользователя по логину и паролю (авторизация), удаление пользователя, добавление нового пользователя в базу данных, сохранение всех базы в файл.

**Личный кабинет (Account):**

**Атрибуты:** пользователь, портал услуг, база данных.

**Методы:** подключение к БД, вход в систему, регистрация нового пользователя.

Проектирование методов класса

**База данных:**

После того, как пользователь авторизовался, мы отдаем ему ссылку на его часть JSON-а от большого файла. Сделано это для того, чтобы при любом изменении JSON-а в классе пользователя, JSON общей базы данных тоже изменялся.

Алгоритм **авторизации**: итеративный поиск по объекту базы по логину и паролю. Возвращение объекта пользователя, если не найден соответствующее исключение

Алгоритм **регистрации**: добавление «призрачного» объекта пользователя (JSON) в конец общей базы. При регистрации создается другой объект пользователя, нежели при авторизации, так как у него нету части общей базы, а он сам пока является уникальной частью, которую нужно добавить в базу.

Алгоритм **удаления**: сравниваем части общего базы пользователя со всеми частями в общей базе, если совпадение есть, то удаляем эту часть из базы, если нет, то такого пользователя в базе нет (или он уже удален).

**Портал услуг:**

Алгоритмы получение и передачи счетов и платежей между пользователем и порталом услуг максимально простое, идет сбор данных из полей структуры элемента графического интерфейса и отправка другому объекту.

Алгоритм **вычисления стоимости услуг:** после изучения предметной области был составлен алгоритм. Оплачиваемая услуга имеет:

1. Тариф оплаты – цена за единицу услуги
2. Норматив – удельное потребление услуги одним человеком (среднее)
3. Тип услуги: зависит только от показаний счетчика, зависит только от кол-ва квадратных метров жилплощади. Например, для услуги ремонта или аренды.

**Алгоритм**:

**Если** (тип услуги – показания счетчика), то

**Если** (показания счетчика неравны 0), то

счет за оплату = показания \* тариф

**Иначе**

счет за оплату = норматив \* кол-во проживающих \* тариф \* повышающий коэффициент

**Иначе** **если** (тип услуги – площадь жилплощади), то

счет за оплату = площадь \* тариф

# Диаграмма объектов UML

Построим диаграмму объектов, используя атрибуты и методы классов, описанных выше (Рисунок 8).

Классы Пользователь, Портал услуг и База данных находятся в отношениях агрегации с классом Личный кабинет, так как они входят в его состав, однако они могут использоваться и отдельно от него, поэтому это именно агрегация, а не композиция. Кардинальность: 1 (одна база данных) - \* (много личных кабинетов), однако в данной ситуации разрешен лишь один объект Личного кабинета, он объявлен в глобальной области формы.

Пользователь связан с Порталом услуг ассоциативными отношениями, так как каждая из сторон использует другую в качестве «получения/предоставления услуги» (отправка/получение показаний/счета для оплаты).

Таким же отношением связаны База данных и Пользователь, так как База предоставляет ссылку на фрагмент Пользователя в полной базе.

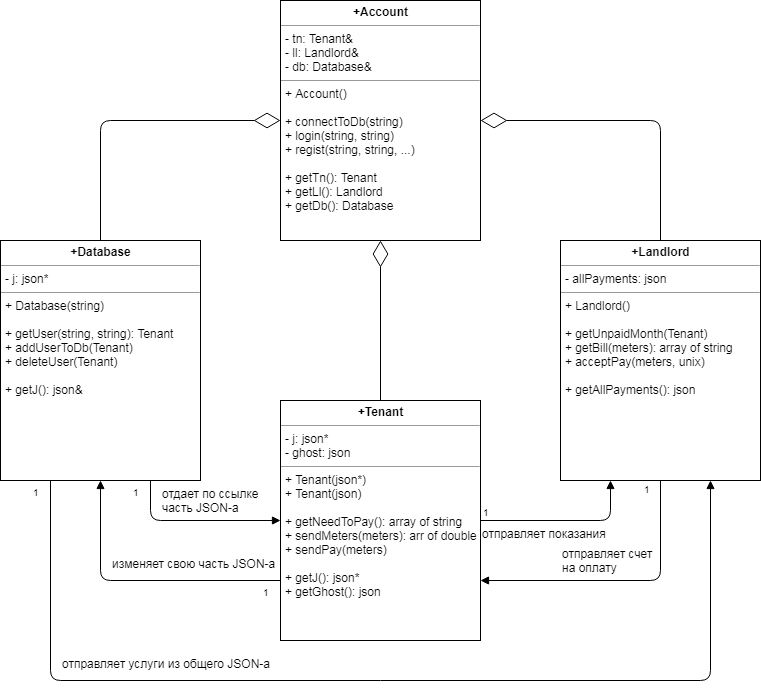


Рисунок – UML-диаграмма объектов

ЭКСПЕРЕМЕНТАЛЬНАЯ ЧАСТЬ

# Решение функциональных задач

В работе необходимо было использовать функции, которые выходили за рамки класса. Они работали в большинстве с формой: дата, отображение структур.

Во-первых, дата. По заданию: пользователь должен отправить показания до 20 числа.

При запуске программы, форма запоминает дату в переменную current. Так как нам нужно будет заплатить за этот месяц, то, если сейчас день месяца > 20, то автоматически изменить на 1-ый день месяца, если день <= 20, то мы можем оставить данный день. Это нужно для того, чтобы мы могли произвольную дату отправить для оплаты, любого месяца, любого года, однако этот режим работает только когда пользователь не авторизован.

Если пользователь авторизован, то если мы находим его последний платеж, то определяем его месяц и изменяем дату оплаты на +1 месяц, если же нет ни одного платежа в файле с платежами, то платить пользователь должен за тот месяц, который был выбран до авторизации.

Если произведена оплата, то можно переключить на следующий месяц дату, но если оплата не произведена, то попытка изменить месяц будет неудачной.

До отправки показаний можно выбрать любой день месяца до 20-го, после отправки мы можем оплатить уже в любой день этого месяца после дня или в день отправки показаний.

При отправке показаний происходит запоминание даты отправки показаний, поэтому мы не можем выбрать день оплаты до дня отправки показаний.

При оплате происходит запись в файл даты оплаты. Цикл повторяется, если мы выйдем из аккаунта, то все равно на следующий раз мы вернемся к следующему периоду оплаты.

Во-вторых, режим администратора. Он включается у определенных (одного или нескольких) пользователей, он (они) способны изменить данные об услугах, как тариф и норматив, так и тип услуги, так ее и название. После этого нужно сохранить файл базы данных.

После чего новая услуга появится к оплате у всех пользователей, а удаленная – удалится одновременно у всех.

Это реализовано на вычитании множеств названий услуг, когда нажата кнопка сохранить, у нас есть старый облик услуг в файле базы данных и новый облик в отображаемой (редактируемой) структуре. Мы загружаем названия услуг в множества (std::set) и с помощью алгоритмов вычитания (#include <algorithm>, std::set\_difference()) получаем новое множество. Вычитая из старого множества новое, мы получаем удаленные услуги из структуры (если в результате операция с множеством содержит хотя бы одну услугу). Вычитая из нового множества старое, мы получаем добавленные услуги при условии, что кол-во элементов в результате множестве больше 0.

Дальше, если что-то добавлено, то добавить в базу данных: основную информацию об услуге и добавить ее всем пользователям, кроме админа, дальше они могут, если имеют льготы, отключить оплату этой услуги, если удалено, удалить из JSON-а базы данных, удалить из всех пользователей (услуги, которые нужно платить).

Также при изменении выбранных льгот, удалять или добавлять их в оплату для данного авторизованного пользователя. В окошке льготы отображаются все возможные услуги, и видно у кого есть льготы и какие, у кого их нет.

# Графический интерфейс

Сконструированная графическая форма имеет 3 различные части (Рисунок 9): регистрация/авторизация, отправка показаний оплата/изменение услуг и отображение всех платежей.

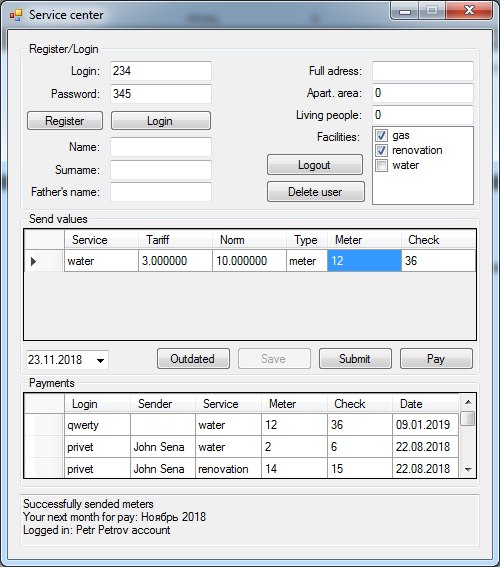


Рисунок – Графический интерфейс формы

Руководство пользователя

Пользователь должен ввести свои логин и пароль и нажать на соответствующую кнопку для авторизации, ввести все данные и нажать «Register» для регистрации, выбрать льготы, если пользователь их имеет.

Далее пользователь перейдет на текущий месяц оплаты, получит все услуги, которые он должен заплатить, в части программы «Send values». Пользователь может выбрать любой день в пределах 1..20.

Затем он должен указать показания счетчиков, если это услуга «по счетчику», указать 0, если счетчика нет, и нужно считать по нормативу. Указать любое число в поле «Meter» >=0, для оплаты услуги, которая рассчитывается по метражу.

Нажать на кнопку «Submit» для отправки данных, после этого выбрать день оплаты и нажать на кнопку «Pay». Теперь платеж успешно отправлен и данные о платеже сохранены.

Для выхода из текущего аккаунта (получение доступа к выбору любой даты) необходимо нажать на кнопку «Logout».

Также можно удалить аккаунт по кнопке «Delete user», находиться в аккаунте необязательно, достаточно указать логин и пароль.

Для сохранения просроченных платежей можно воспользоваться кнопкой «Outdated», данная клавиша сохранит платежи, даже если вы не авторизованы.

Если пользователь входит в аккаунт админа, то он может менять все платежи. Отправить показания и платеж в данном режиме невозможно, однако, в том же окне можно поменять услуги. Для сохранения нажмите «Save».

# Тестирование

Все ошибки и сообщения об успешных завершениях функций отображаются в окошке снизу формы. Тестирование программы показано в таблице 1 ниже.

Таблица 1 – Тестирование программы

|  |  |  |
| --- | --- | --- |
| **Действие** | **Результат** | **Исключения** |
| Авторизация |  | Пользователь не найден в базе данных – сообщение об этом в нижнем логгере |
| Регистрация |  | Введен логин или пароль меньше одного символа, не введены обязательные поля ПЛОЩАДЬ и КОЛ-ВО проживающих или они меньше нуля, пользователь уже существует |
|  |  |  |
| Продолжение таблицы 1 | | |
| Оплата услуг |  | Одно поле или несколько полей «показания счетчика» не заполнено, введены числа, меньшие нуля, не являются double |
| Выбор другой даты |  | **Не отправлены данные**: выбран день > 20, другой месяц или год.  **Отправлены**: выбран день предшествующий дню оплаты, другой день и месяц.  **Оплачено**: выбрана другая дата неравная следующему месяцу. |
|  |  |  |
| Продолжение таблицы 1 | | |
| Добавление услуги админом |  | Неправильно указан тип услуги, не введены оба параметра тариф и норматив, нет названия, указан тип метраж, но не указан тариф |

ЗАКЛЮЧЕНИЕ

Объектно-ориентированное программирование включает в себя классы и объект (экземпляры класса), а также все возможные взаимодействия и отношения между ними. Три важных понятиях ООП: наследование – перенятие свойств и методов класса-родителя классами-детьми; инкапсуляция – объединение и сокрытие важных данных в одной структуре класса; полиморфизм – разная реализация у одних сигнатур методов (например, у классов-потомков).

На диаграммах UML возможно отобразить архитектуру проекта, суть взаимодействий множества классов. Существуют такие типы взаимодействий: наследование (расширение одного класса другим), агрегация (один класс – часть другого), ассоциация (один класс использует другой, «пользуется услугами»), композиция (строгая агрегация, ограниченная по времени жизни классов), зависимость, использование (привязка данных).

Пользовательский интерфейс – важная часть программы, отвечающая за опыт взаимодействия пользователя (UX), важно сделать его удобным, понятным и красивым.

В результате курсовой работы была достигнута основная цель: обучение технологии создания и описания программного обеспечения, анализ прикладной области, изучение объектно-ориентированного программирования и подхода к решению задачи взаимодействия нескольких объектов, выделение из жизненной задачи программных объектов.

В ходе курсовой работы были рассмотрены такие методы и решены такие задачи: методы алгоритмической и объектной декомпозиции, отлажено взаимодействие классов, проведена работа со сложными структурами: JSON и файлами, обработано большое количество исключений из стандартного хода работы обеспечения.

По итогам была создана архитектура программной системы и осуществлена ее практическая реализация.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

* Ларман, Крэг Применение UML 2.0 и шаблонов проектирования. Практическое руководство. 3-е издание. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2013. – 736 с. : ил. – Парал. тит. англ.
* Вайсфельд М. Объектно-ориентированное мышление. – СПб.: Питер, 2014. – 304 с.: ил. – (Серия «Библиотека программиста»).
* Бьёрн Страуструп «Программирование: принципы и практика с использованием C++» отдельное издание, пер в англ. – Игорь Красиков Издательство «Вильямс» 2016 г. Тираж 300, 1328 стр.

ПРИЛОЖЕНИЯ

Программный код модулей приложения приведен ниже.

# Global.h

#pragma once

#include <string>

namespace Global {

static std::string toString(System::String^ str = "") {

using namespace System;

using namespace Runtime::InteropServices;

const char\* ptr = (const char\*)(Marshal::StringToCoTaskMemAnsi(str)).ToPointer();

return std::string(ptr);

}

static System::String^ toFormString(std::string str = "") {

using namespace System;

return gcnew String(str.c\_str());

}

static unsigned int toInt(System::String^ str = "") {

return System::Convert::ToInt16(str);

}

static int toUnixTime(System::DateTime date) {

return (int)(date - System::DateTime(1970, 1, 1)).TotalSeconds;

}

static System::DateTime toSystemDate(int unixtime) {

return System::DateTime(1970, 1, 1, 0, 0, 0, 0).AddSeconds(unixtime);

}

};

# Tenant.h

#pragma once

#include "Global.h"

using namespace Global;

#include <string>

#include <vector>

#include "json.hpp"

using json = nlohmann::json;

class Tenant {

public:

Tenant();

Tenant(json\* j);

Tenant(json j);

Tenant(std::string name, std::string surname);

void addIntoJson(std::string path, std::string key);

std::vector<std::string> getNeedToPay();

System::String^ intro();

System::String^ test();

std::string name = "", surname = "";

bool getIsAdminMode();

void setIsAdminMode(bool value);

json\* getJ();

json& getGhost();

private:

json\* j;

json ghost;

bool isAdminMode = false;

};

# Tenant.cpp

#include "Tenant.h"

#include "json.hpp"

using json = nlohmann::json;

#include <vector>

Tenant::Tenant() {

}

Tenant::Tenant(json j) {

this->ghost = j;

}

Tenant::Tenant(json\* j) {

this->j = j;

}

Tenant::Tenant(std::string name, std::string surname) {

this->name = name;

this->surname = surname;

}

json\* Tenant::getJ() {

return this->j;

}

json& Tenant::getGhost() {

return this->ghost;

}

bool Tenant::getIsAdminMode() {

return this->isAdminMode;

}

void Tenant::setIsAdminMode(bool value) {

this->isAdminMode = value;

}

void Tenant::addIntoJson(std::string path, std::string key) {

(\*j).at("privacy")[path] = key;

}

std::vector<std::string> Tenant::getNeedToPay() {

return (\*j).at("pays").at("needToPay");

}

System::String^ Tenant::intro() {

return toFormString((\*j).at("privacy").at("name")) + " " + toFormString((\*j).at("privacy").at("surname"));

}

System::String^ Tenant::test() {

return toFormString(ghost["privacy"]["money"]);

}

# Landlord.h

#pragma once

#include <string>

#include <vector>

#include <fstream>

#include "Tenant.h"

#include "json.hpp"

using json = nlohmann::json;

class Landlord {

public:

Landlord();

std::vector<double> getBill(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, json payment);

void acceptPay(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, int unixtime);

void savePayments(std::string path);

int getUnpaidMonth(Tenant tn);

json getAllPayments();

std::vector<std::string> saveOutdated(int unixtime);

bool getFinished();

bool getStarted();

void setFinished(bool value);

void setStarted(bool value);

private:

json allPayments;

bool finished = false;

bool started = false;

};

# Landlord.cpp

#include "Landlord.h"

Landlord::Landlord() {

std::ifstream file("payments.json");

if (file.is\_open())

file >> this->allPayments;

file.close();

}

bool Landlord::getStarted() {

return this->started;

}

void Landlord::setStarted(bool value) {

this->started = value;

}

bool Landlord::getFinished() {

return this->finished;

}

void Landlord::setFinished(bool value) {

this->finished = value;

}

std::vector<double> Landlord::getBill(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, json payment) {

// bill = tariff \* pokazaniya

// pok == 0 -> bill = norm \* living people \* tariff

// type == f -> bill = area \* tariff

this->finished = false;

this->started = true;

std::vector<double> result;

for (int i = 0; i < map.size(); i++)

if (payment.at(map[i][0]).at("type") == "meter")

if (numbers[i][4] == 0 || payment.at(map[i][0]).count("tariff") == 0)

result.push\_back(payment.at(map[i][0]).at("norm") \* (int)tn.getJ()->at("privacy").at("living people") \* payment.at(map[i][0]).at("tariff"));

else

result.push\_back(payment.at(map[i][0]).at("tariff") \* numbers[i][4]);

else

result.push\_back((int)tn.getJ()->at("privacy").at("area") \* payment.at(map[i][0]).at("tariff"));

return result;

}

void Landlord::acceptPay(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, int unixtime) {

for (auto i : map) {

this->allPayments.push\_back({

{ "sender", tn.getJ()->at("privacy")["name"].get<std::string>() + " " + tn.getJ()->at("privacy")["surname"].get<std::string>() },

{ "login", tn.getJ()->at("auth")["login"] },

{ "payments", i[0]},

{ "tariff", i[1]},

{ "norm", i[2]},

{ "type", i[3]},

{ "meters", i[4]},

{ "check", i[5]},

{ "unixtime", unixtime}

});

}

this->started = false;

this->finished = true;

}

void Landlord::savePayments(std::string path) {

std::ofstream file(path);

file << std::setw(4) << this->allPayments << std::endl;

file.close();

}

int Landlord::getUnpaidMonth(Tenant tn) {

int time = 0;

for (auto i : this->allPayments)

if (tn.getJ()->at("auth")["login"] == i["login"])

if (i["unixtime"] > time)

time = (int)i["unixtime"];

return time;

}

json Landlord::getAllPayments() {

return this->allPayments;

}

std::vector<std::string> Landlord::saveOutdated(int currentUnixtime) {

json years = {};

std::vector<std::string> fileNames;

for (auto i : this->allPayments)

if (toSystemDate(currentUnixtime).Year - toSystemDate(i["unixtime"]).Year > 0)

years[toString(toSystemDate(i["unixtime"]).Year.ToString())].push\_back(i);

for (auto i : years.items()) {

std::ofstream file(i.key() + ".json");

if (file.is\_open())

file << std::setw(4) << i.value() << std::endl;

file.close();

fileNames.push\_back(i.key());

}

return fileNames;

}

# Database.h

#pragma once

#include "Landlord.h"

#include <fstream>

#include "json.hpp"

using json = nlohmann::json;

class Database {

public:

Database();

Database(std::string filename);

void save(std::string filename = "");

void addUserToDb(Tenant tn);

void deleteUser(Tenant tn);

Tenant getUser(std::string login, std::string password);

bool isDbLoaded;

std::string filename = "";

json& getJ();

private:

json j;

};

# Database.cpp

#include "Database.h"

Database::Database() {

}

Database::Database(std::string filename) {

this->filename = filename;

std::ifstream file(filename);

if (!file.is\_open()) throw std::exception("File corrupted or didn't find");

file >> this->j;

file.close();

this->isDbLoaded = true;

}

void Database::save(std::string filename) {

if (filename == "" && this->filename != "")

filename = this->filename;

std::ofstream file(filename);

file << std::setw(4) << this->j << std::endl;

file.close();

}

json& Database::getJ() {

return this->j;

}

Tenant Database::getUser(std::string login, std::string password) {

for (auto& i : j["users"]) {

if (i["auth"]["login"] == login && i["auth"]["password"] == password) {

if (i["auth"].find("isAdmin") != i["auth"].end()) {

Tenant tmp = Tenant(&i);

tmp.setIsAdminMode(true);

return tmp;

}

return Tenant(&i);

}

}

throw std::exception("User isn't found...");

}

void Database::addUserToDb(Tenant tn) {

if (\*(tn.getJ()) != 0)

this->j["users"].push\_back(\*(tn.getJ()));

else

this->j["users"].push\_back(tn.getGhost());

}

void Database::deleteUser(Tenant tn) {

int index = -1;

for (int i = 0; i < this->j["users"].size(); i++)

if (this->j["users"][i] == \*(tn.getJ())) {

index = i;

break;

}

if (index != -1)

this->j["users"].erase(index);

}

# Account.h

#pragma once

#include <string>

#include <set>

#include <algorithm>

#include "Database.h"

#include "json.hpp"

using json = nlohmann::json;

class Account {

public:

Account();

void connectToDb(std::string path);

void login(std::string login, std::string password);

void regist(std::string login, std::string password,

std::string name, std::string surname, std::string father,

std::string address, unsigned int area, unsigned int people,

std::vector<std::string> facilities);

std::vector<double> sendMeters(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, json payment);

void sendPay(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, int unixtime);

Tenant& getTn();

Landlord& getLl();

Database& getDb();

private:

Tenant tn;

Landlord ll;

Database db;

};

# Account.cpp

#include "Account.h"

Account::Account() {

}

Tenant& Account::getTn() {

return this->tn;

}

Landlord& Account::getLl() {

return this->ll;

}

Database& Account::getDb() {

return this->db;

}

void Account::connectToDb(std::string path) {

if (!this->db.isDbLoaded || path != this->db.filename) {

Database db(path);

this->db = db;

}

}

void Account::login(std::string login, std::string password) {

this->tn = this->db.getUser(login, password);

}

void Account::regist(std::string login, std::string password, std::string name, std::string surname,

std::string father, std::string address, unsigned int area, unsigned int people,

std::vector<std::string> facilities) {

for (auto i : db.getJ()["users"]) {

if (i["auth"]["login"] == login)

throw std::exception("Account with such login already exists, select another");

}

json person;

person["auth"]["login"] = login;

person["auth"]["password"] = password;

person["privacy"]["name"] = name;

person["privacy"]["surname"] = surname;

person["privacy"]["father's name"] = father;

person["privacy"]["address"] = address;

person["privacy"]["area"] = area;

person["privacy"]["living people"] = people;

person["pays"]["needToPay"] = facilities;

Tenant temp(&person);

this->db.addUserToDb(temp);

this->tn = this->db.getUser(login, password);

}

std::vector<double> Account::sendMeters(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, json payment) {

return this->ll.getBill(map, numbers, tn, payment);

// bill = tariff \* pokazaniya

// pok == 0 -> bill = norm \* living people \* tariff

// type == f -> bill = area \* tariff

}

void Account::sendPay(std::vector<std::vector<std::string>> map,

std::vector<std::vector<double>> numbers, Tenant tn, int unixtime) {

this->ll.acceptPay(map, numbers, tn, unixtime);

}

# MyForm.h

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

acc.connectToDb("database.json");

if (System::Convert::ToInt16(textBox7->Text) <= 0)

throw std::exception("Please fill in AREA with integer countable number");

if (System::Convert::ToInt16(textBox8->Text) <= 0)

throw std::exception("Please fill in LIVING PEOPLE with integer countable number");

if (textBox1->Text->Length < 1)

throw std::exception("Enter at least 1 character in the LOGIN field");

if (textBox2->Text->Length < 1)

throw std::exception("Enter at least 1 character in the PASSWORD field");

isCodeClick = true;

for (int j = 0; j < checkedListBox1->Items->Count; j++)

checkedListBox1->SetItemChecked(j, false);

isCodeClick = false;

dataGridView1->Rows->Clear();

std::vector<std::string> facilities;

for (int i = 0; i < checkedListBox1->Items->Count; i++) {

if (checkedListBox1->GetItemCheckState(i) == CheckState::Unchecked)

facilities.push\_back(toString(checkedListBox1->Items[i]->ToString()));

}

acc.regist(toString(textBox1->Text), toString(textBox2->Text), toString(textBox3->Text),

toString(textBox4->Text), toString(textBox5->Text), toString(textBox6->Text),

toInt(textBox7->Text), toInt(textBox8->Text), facilities);

textBox10->Text = "Successfully registered: " + acc.getTn().intro() + "\r\n" + textBox10->Text;

textBox3->Text = ""; textBox4->Text = ""; textBox5->Text = "";

textBox6->Text = ""; textBox7->Text = "0"; textBox8->Text = "0";

acc.getDb().save("database.json");

}

catch (System::FormatException^ e) {

textBox10->Text = "One or few fields don't filled by ints\r\n" + e->Message + "\r\n" + textBox10->Text;

}

catch (json::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

textBox10->Text = dateTimePicker1->Value.ToString() + "\r\n" + textBox10->Text;

//currentTime.current = currentTime.current.Now;

currentTime.current = DateTime(currentTime.current.Year, currentTime.current.Month,

currentTime.current.Day <= 20 ? currentTime.current.Day : 1,

currentTime.current.Hour, currentTime.current.Minute, currentTime.current.Second);

dateTimePicker1->Value = currentTime.current;

try {

try {

acc.connectToDb("database.json");

}

catch (json::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

throw std::exception("File database.json can't be opened");

}

acc.login(toString(textBox1->Text), toString(textBox2->Text));

textBox10->Text = "Logged in: " + acc.getTn().intro() + " account\r\n" + textBox10->Text;

isLogin = true;

isCodeClick = true;

for (int j = 0; j < checkedListBox1->Items->Count; j++)

checkedListBox1->SetItemChecked(j, true);

isCodeClick = false;

int time = acc.getLl().getUnpaidMonth(acc.getTn());

if (time > 0) {

textBox10->Text = "Your next month for pay: "

+ toSystemDate(time).AddMonths(1).ToString("MMMM yyyy") + "\r\n" + textBox10->Text;

currentTime.current = toSystemDate(time).AddMonths(1);

currentTime.current = DateTime(currentTime.current.Year, currentTime.current.Month, 1,

currentTime.current.Hour, currentTime.current.Minute, currentTime.current.Second);

dateTimePicker1->Value = currentTime.current;

}

dataGridView1->Rows->Clear();

if (acc.getTn().getIsAdminMode()) {

button5->Enabled = false;

button4->Enabled = false;

button6->Enabled = true;

dataGridView1->AllowUserToAddRows = true;

dataGridView1->AllowUserToDeleteRows = true;

for (int i = 0; i < dataGridView1->Columns->Count - 2; i++)

dataGridView1->Columns[i]->ReadOnly = false;

for (int i = dataGridView1->Columns->Count - 2; i < dataGridView1->Columns->Count - 1; i++)

dataGridView1->Columns[i]->ReadOnly = true;

checkedListBox1->Enabled = false;

for (auto i : acc.getDb().getJ()["payment"].items()) {

json temp = i.value();

dataGridView1->Rows->Add(toFormString(i.key()),

temp.find("tariff") != temp.end()

? toFormString(std::to\_string((double)temp["tariff"]))

: "-",

temp.find("norm") != temp.end()

? toFormString(std::to\_string((double)temp["norm"]))

: "-",

toFormString(temp["type"]));

}

}

else {

button5->Enabled = true;

button4->Enabled = true;

button6->Enabled = false;

dataGridView1->AllowUserToAddRows = false;

dataGridView1->AllowUserToDeleteRows = false;

checkedListBox1->Enabled = true;

isCodeClick = true;

for (auto i : acc.getTn().getNeedToPay())

for (int j = 0; j < checkedListBox1->Items->Count; j++)

if (checkedListBox1->Items[j]->ToString() == toFormString(i))

checkedListBox1->SetItemChecked(j, false);

isCodeClick = false;

for (int i = 0; i < dataGridView1->Columns->Count - 2; i++)

dataGridView1->Columns[i]->ReadOnly = true;

for (int i = dataGridView1->Columns->Count - 2; i < dataGridView1->Columns->Count - 1; i++)

dataGridView1->Columns[i]->ReadOnly = false;

for (auto i : acc.getTn().getNeedToPay()) {

auto temp = acc.getDb().getJ()["payment"].at(i);

dataGridView1->Rows->Add(toFormString(i),

temp.find("tariff") != temp.end()

? toFormString(std::to\_string((double)temp["tariff"]))

: "-",

temp.find("norm") != temp.end()

? toFormString(std::to\_string((double)temp["norm"]))

: "-",

toFormString(temp["type"]));

}

}

}

catch (json::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

if (acc.getTn().getIsAdminMode())

throw std::exception("You can't delete Admin account");

acc.connectToDb("database.json");

acc.getDb().deleteUser(acc.getDb().getUser(toString(textBox1->Text), toString(textBox2->Text)));

acc.getDb().save();

isCodeClick = true;

for (int j = 0; j < checkedListBox1->Items->Count; j++)

checkedListBox1->SetItemChecked(j, false);

isCodeClick = false;

checkedListBox1->Enabled = false;

dataGridView1->Rows->Clear();

isLogin = false;

textBox10->Text = "Account is deleted from " + toFormString(acc.getDb().filename) + "\r\n" + textBox10->Text;

}

catch (json::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void button4\_Click(System::Object^ sender, System::EventArgs^ e) {

if (!isLogin || acc.getLl().getFinished())

return;

std::vector<std::vector<std::string>> total;

std::vector<std::vector<double>> numbers;

try {

for (int i = 0; i < dataGridView1->Rows->Count; i++) {

double converted = System::Convert::ToDouble(dataGridView1->Rows[i]->Cells[4]->Value->ToString()->Replace(".", ","));

if (converted < 0) throw std::exception("The meter reading can't be less 0");

std::vector<std::string> temp;

std::vector<double> nums;

for (int j = 0; j < dataGridView1->Rows[i]->Cells->Count - 1; j++) {

temp.push\_back(toString(dataGridView1->Rows[i]->Cells[j]->Value->ToString()));

try {

double num = System::Convert::ToDouble(dataGridView1->Rows[i]->Cells[j]->Value->ToString()->Replace(".", ","));

nums.push\_back(num);

}

catch (System::FormatException^) {

nums.push\_back(NULL);

}

}

total.push\_back(temp);

numbers.push\_back(nums);

}

auto pays = acc.sendMeters(total, numbers, acc.getTn(), acc.getDb().getJ()["payment"]);

for (int i = 0; i < pays.size(); i++)

dataGridView1->Rows[i]->Cells[5]->Value = pays[i];

dataGridView1->Columns[4]->ReadOnly = true;

currentTime.sendedMeters = currentTime.current;

textBox10->Text = "Successfully sended meters\r\n" + textBox10->Text;

}

catch (System::FormatException^ e) {

textBox10->Text = "One or few fields don't filled by doubles\r\n" + e->Message + "\r\n" + textBox10->Text;

}

catch (System::NullReferenceException^ e) {

textBox10->Text = "One or few fields don't filled\r\n" + e->Message + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void button5\_Click(System::Object^ sender, System::EventArgs^ e) {

if (acc.getLl().getStarted()) {

std::vector<std::vector<std::string>> total;

std::vector<std::vector<double>> numbers;

std::vector<std::string> temp;

std::vector<double> nums;

try {

for (int i = 0; i < dataGridView1->Rows->Count; i++) {

double converted = System::Convert::ToDouble(dataGridView1->Rows[i]->Cells[4]->Value->ToString()->Replace(".", ","));

if (converted < 0) throw std::exception("The meter reading can't be less 0");

std::vector<std::string> temp;

std::vector<double> nums;

for (int j = 0; j < dataGridView1->Rows[i]->Cells->Count; j++) {

temp.push\_back(toString(dataGridView1->Rows[i]->Cells[j]->Value->ToString()));

try {

double num = System::Convert::ToDouble(dataGridView1->Rows[i]->Cells[j]->Value->ToString()->Replace(".", ","));

nums.push\_back(num);

}

catch (System::FormatException^) {

nums.push\_back(NULL);

}

}

total.push\_back(temp);

numbers.push\_back(nums);

textBox10->Text = "Payment for " + dataGridView1->Rows[i]->Cells[0]->Value->ToString()

+ " is sent...\r\n" + textBox10->Text;

}

acc.sendPay(total, numbers, acc.getTn(), toUnixTime(dateTimePicker1->Value));

acc.getLl().savePayments("payments.json");

dataGridView1->Columns[4]->ReadOnly = false;

dataGridView2->Rows->Clear();

for (auto i : acc.getLl().getAllPayments())

dataGridView2->Rows->Insert(0, toFormString(i["login"]), toFormString(i["sender"]),

toFormString(i["payments"]), toFormString(i["meters"]),

toFormString(i["check"]), toSystemDate(i["unixtime"]).ToString("dd.MM.yyyy"));

}

catch (json::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

}

private: System::Void dateTimePicker1\_ValueChanged(System::Object^ sender, System::EventArgs^ e) {

auto temp = dateTimePicker1->Value;

try {

if (!isLogin)

currentTime.current = temp;

else

if (!acc.getLl().getFinished()) { // not paid

if (!acc.getLl().getStarted() && currentTime.current.ToString("MMMM yyyy") == temp.ToString("MMMM yyyy")

&& temp.Day <= 20) // not sended meters

currentTime.current = temp;

if (acc.getLl().getStarted() && currentTime.current.ToString("MMMM yyyy") == temp.ToString("MMMM yyyy")

&& temp.Compare(currentTime.sendedMeters, temp) <= 0) // sended meters

currentTime.current = temp;

dateTimePicker1->Value = currentTime.current;

}

else // paid

if (currentTime.current.AddMonths(1).ToString("MMMM yyyy") == temp.ToString("MMMM yyyy")) {

currentTime.current = DateTime(temp.Year, temp.Month, 1, temp.Hour, temp.Minute, temp.Second);

textBox10->Text = "New month is " + currentTime.current.ToString("MMMM") + "\r\n" + textBox10->Text;

dataGridView1->Rows->Clear();

for (auto i : acc.getTn().getNeedToPay()) {

auto temp = acc.getDb().getJ()["payment"].at(i);

dataGridView1->Rows->Add(toFormString(i),

temp.find("tariff") != temp.end()

? toFormString(std::to\_string((double)temp["tariff"]))

: "-",

temp.find("norm") != temp.end()

? toFormString(std::to\_string((double)temp["norm"]))

: "-",

toFormString(temp["type"]));

}

acc.getLl().setFinished(false);

}

else {

if (currentTime.current.ToString("MMMM yyyy") == temp.ToString("MMMM yyyy"))

currentTime.current = temp;

dateTimePicker1->Value = currentTime.current;

}

}

catch (json::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void MyForm\_Shown(System::Object^ sender, System::EventArgs^ e) {

currentTime.current = dateTimePicker1->Value;

currentTime.current = DateTime(currentTime.current.Year, currentTime.current.Month, 1,

currentTime.current.Hour, currentTime.current.Minute, currentTime.current.Second);

checkedListBox1->Enabled = false;

try {

acc.connectToDb("database.json");

for (auto i : acc.getDb().getJ()["payment"].items())

checkedListBox1->Items->Add(toFormString(i.key()), false);

dataGridView2->Rows->Clear();

for (auto i : acc.getLl().getAllPayments())

dataGridView2->Rows->Insert(0, toFormString(i["login"]), toFormString(i["sender"]),

toFormString(i["payments"]), toFormString(i["meters"]),

toFormString(i["check"]), toSystemDate(i["unixtime"]).ToString("dd.MM.yyyy"));

}

catch (json::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = "File database.json can't be opened\r\n"

+ toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void button6\_Click(System::Object^ sender, System::EventArgs^ e) {

if (!acc.getTn().getIsAdminMode())

return;

json payment = {};

std::vector<std::string> temp;

std::vector<double> nums;

try {

for (int i = 0; i < dataGridView1->Rows->Count; i++) {

for (int j = 0; j < dataGridView1->Rows[i]->Cells->Count - 2; j++) {

try {

temp.push\_back(toString(dataGridView1->Rows[i]->Cells[j]->Value->ToString()));

}

catch (System::NullReferenceException^) {

temp.push\_back("");

}

try {

double num = System::Convert::ToDouble(dataGridView1->Rows[i]->Cells[j]->Value->ToString()->Replace(".", ","));

nums.push\_back(num);

}

catch (System::FormatException^) {

nums.push\_back(NULL);

}

catch (System::NullReferenceException^) {

nums.push\_back(NULL);

}

}

if (temp[0] == "" && temp[1] == "" && temp[2] == "" && temp[3] == "") {

temp.clear();

nums.clear();

continue;

}

else {

if (temp[0] == "") {

std::string errStr = "Please input name (kind) of service in " + std::to\_string(i + 1) + " row";

throw std::exception(errStr.c\_str());

}

if (temp[3] == "" || (temp[3] != "footage" && temp[3] != "meter")) {

std::string errStr = "Please input type of service (\"" + temp[0] + "\" in "

+ std::to\_string(i + 1) + " row): \"footage\" or \"meter\" only";

throw std::exception(errStr.c\_str());

}

if (nums[1] == NULL && nums[2] == NULL) {

std::string errStr = "Please input tariff or/and norm in \"" + temp[0] + "\" in "

+ std::to\_string(i + 1) + " row";

throw std::exception(errStr.c\_str());

}

}

if (nums[2] == NULL ^ nums[1] == NULL) {

if (nums[2] == NULL)

payment[temp[0]] = {

{ "tariff", nums[1] },

{ "type", temp[3] }

};

if (nums[1] == NULL) {

if (temp[3] == "footage")

throw std::exception("You can't input footage service without tariff");

payment[temp[0]] = {

{ "norm", nums[2] },

{ "type", temp[3] }

};

}

}

else

payment[temp[0]] = {

{ "norm", nums[2] },

{ "tariff", nums[1] },

{ "type", temp[3] }

};

temp.clear();

nums.clear();

}

std::set<std::string> oldJson;

std::set<std::string> newJson;

std::set<std::string> result;

for (auto i : acc.getDb().getJ()["payment"].items())

oldJson.insert(i.key());

for (auto i : payment.items())

newJson.insert(i.key());

std::set\_difference(oldJson.begin(), oldJson.end(),

newJson.begin(), newJson.end(), std::inserter(result, result.begin())); // deleted

if (result.size() > 0) // deleted

for (std::set<std::string>::iterator i = result.begin(); i != result.end(); ++i)

for (auto& j : acc.getDb().getJ()["users"]) {

int cx = 0;

if (j["auth"].count("isAdmin") > 0) continue;

for (auto& k : j["pays"]["needToPay"].items()) {

if (k.value() == \*i) {

j["pays"]["needToPay"].erase(j["pays"]["needToPay"].begin() + cx);

break;

}

cx++;

}

}

result.clear();

std::set\_difference(newJson.begin(), newJson.end(),

oldJson.begin(), oldJson.end(), std::inserter(result, result.begin())); // inserted

if (result.size() > 0) // inserted

for (std::set<std::string>::iterator i = result.begin(); i != result.end(); ++i)

for (auto& j : acc.getDb().getJ()["users"])

if (j["auth"].count("isAdmin") == 0)

j["pays"]["needToPay"].push\_back(\*i);

acc.getDb().getJ()["payment"] = payment;

acc.getDb().save();

checkedListBox1->Items->Clear();

isCodeClick = true;

for (auto i : acc.getDb().getJ()["payment"].items())

checkedListBox1->Items->Add(toFormString(i.key()), true);

isCodeClick = false;

textBox10->Text = "Changes successfully saved\r\n" + textBox10->Text;

}

catch (json::exception e) {

textBox10->Text = "There is some error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void button7\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

if (isLogin) {

acc.getTn().~Tenant();

isLogin = false;

textBox10->Text = "Logout is executed\r\n" + textBox10->Text;

}

else

textBox10->Text = "Logout has executed already\r\n" + textBox10->Text;

dataGridView1->AllowUserToAddRows = false;

dataGridView1->AllowUserToDeleteRows = false;

dataGridView1->Rows->Clear();

checkedListBox1->Enabled = false;

isCodeClick = true;

for (int j = 0; j < checkedListBox1->Items->Count; j++)

checkedListBox1->SetItemChecked(j, false);

isCodeClick = false;

}

catch (std::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void checkedListBox1\_ItemCheck(System::Object^ sender, System::Windows::Forms::ItemCheckEventArgs^ e) {

try {

if (!isLogin || isCodeClick)

return;

if (e->NewValue == CheckState::Checked)

for (auto& j : acc.getDb().getJ()["users"]) {

int cx = 0;

if (j["auth"].count("isAdmin") > 0) continue;

for (auto& k : j["pays"]["needToPay"].items()) {

if (k.value() == toString(checkedListBox1->Items[e->Index]->ToString())) {

j["pays"]["needToPay"].erase(j["pays"]["needToPay"].begin() + cx);

break;

}

cx++;

}

}

else if (e->NewValue == CheckState::Unchecked) {

for (auto& j : acc.getDb().getJ()["users"])

if (j["auth"].count("isAdmin") == 0)

j["pays"]["needToPay"].push\_back(toString(checkedListBox1->Items[e->Index]->ToString()));

}

dataGridView1->Rows->Clear();

for (auto i : acc.getTn().getNeedToPay()) {

auto temp = acc.getDb().getJ()["payment"].at(i);

dataGridView1->Rows->Add(toFormString(i),

temp.find("tariff") != temp.end()

? toFormString(std::to\_string((double)temp["tariff"]))

: "-",

temp.find("norm") != temp.end()

? toFormString(std::to\_string((double)temp["norm"]))

: "-",

toFormString(temp["type"]));

}

acc.getDb().save();

}

catch (json::exception e) {

textBox10->Text = "There is some error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

private: System::Void button8\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

for (auto i : acc.getLl().saveOutdated(toUnixTime(currentTime.current)))

textBox10->Text = toFormString(i) + ".json file is saved\r\n" + textBox10->Text;

}

catch (json::exception e) {

textBox10->Text = "Error: " + toFormString(e.what()) + "\r\n" + textBox10->Text;

}

catch (std::exception e) {

textBox10->Text = toFormString(e.what()) + "\r\n" + textBox10->Text;

}

}

# MyForm.cpp

#include "MyForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

void Main(array<String^>^ args) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

CourseWork::MyForm form;

Application::Run(%form);

}