**СОДЕРЖАНИЕ**

1. [Лист задания](#_Toc8083)

[2. Введение](#_Toc8083)

[3. Постановка задачи](#_Toc23024)

[4. Обзор литературы](#_Toc9120)

[4.1 Обзор методов и алгоритмов решения поставленной задачи](#_Toc21740)

[5. Функциональное проектирование](#_Toc27617)

[5.1 Структура входных и выходных данных](#_Toc25766)

[5.2 Разработка диаграммы классов](#_Toc18142)

[5.3 Описание классов](#_Toc27213)

[6. Разработка программных модулей](#_Toc21162)

[6.1. Разработка схем алгоритмов (два наиболее важных метода)](#_Toc20165)

[6.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов)](#_Toc29515)

[7. Результаты работы программы](#_Toc20966)

[8. Заключение](#_Toc19834)

[9. Литература](#_Toc24767)

[10. Приложения](#_Toc28769)

**ВВЕДЕНИЕ**

C++ - это компилируемый язык программирования, который обладает структурой и ориентирован на объекты. Несмотря на свой долгий возраст, C++ по-прежнему остается одним из самых востребованных и популярных языков программирования в мире. Этот язык поддерживает различные стили программирования, такие как процедурное, абстрактное, объектно-ориентированное и обобщенное программирование, предоставляя разработчику свободу выбора стиля для написания программы.

C++ также есть библиотеки, они представляют собой наборы предопределенных кодов, функций и классов, которые разработчики могут использовать для упрощения различных задач в программировании. Эти библиотеки предоставляют готовые решения для широкого спектра задач, таких как обработка данных, работа с графическим интерфейсом, работа с сетью, многопоточность и многое другое. Вместо того чтобы писать код с нуля, разработчики могут воспользоваться библиотеками для экономии времени и улучшения производительности своих программ. Библиотеки в C++ могут быть частью стандартной библиотеки (STL) или сторонними библиотеками, разработанными сообществами или сторонними организациями.

Одной из ключевых характеристик C++ является его высокая производительность. Важной особенностью этого языка является его объектно-ориентированный аспект, включающий в себя множество функций, таких как конструкторы, деструкторы, перегрузка операторов и создание классов.

Для поддержки принципов объектно-ориентированного программирования, включая инкапсуляцию, полиморфизм, наследование и абстракцию, C++ и другие объектно-ориентированные языки предоставляют соответствующие средства. Инкапсуляция защищает данные и код от внешних воздействий и ошибок. Полиморфизм позволяет обращаться к различным действиям через единый интерфейс. Наследование позволяет объектам наследовать свойства других объектов. Абстракция позволяет использовать только необходимые характеристики объекта, представляя его минимальным набором полей и методов, подходящим для конкретной задачи.

Информационная система (ИС) - это система, предназначенная для хранения, поиска и обработки информации. Результатом работы информационных систем является информационная продукция, такая как документы, базы данных и информационные услуги.

Сегодня C++ активно используется при разработке операционных систем, драйверов устройств и разнообразных прикладных систем в современном мире: разработке игр, научных и инженерных вычислениях, финансовых приложениях, системах управления базами данных, встраиваемых системах (микроконтроллеры, интернет вещей).

# 3 Постановка задачи

Программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Информация должна храниться в нескольких файлах, связанных определенным образом. Необходимо хранить следующие сведения: данные о квартирах, частном секторе, нежилых помещениях, коммерческой недвижимости, а также о информации по заданным признакам. Реализовать добавление, удаление, редактирование данных, поиск информации по заданным признакам. При реализации функций добавление, удаление, редактирование данных необходимо предусмотреть операцию отмены последних действий (более одного).

Разработать иерархию классов с использованием наследования (не меньше 3-х уровней наследования). Разработать и использовать классы контейнеров, итераторов и алгоритмов(свои и STL). Производить обработку исключительных ситуаций. При реализации операции редактирования, добавления, удаления информации необходимо предусмотреть операцию отмены последних действий.

# 4 Обзор литературы

**4.1 Обзор методов и алгоритмов решения поставленной задачи**

Для реализации данной информационной системы было выбран язык программирования C++.

Каждая команда языка высокого уровня эквивалентна нескольким командам в машинных кодах, поэтому программы, написанные на языках высокого уровня, более компактны, чем аналогичные программы в машинных кодах.

Язык C++ основан на объектно-ориентированной парадигме (ООП). ООП — это парадигма разработки программных систем, в которой приложения состоят из объектов. *Объекты* — это сущности, у которых есть свойства и поведение. Обычно объекты являются экземплярами какого-нибудь класса. *Свойства* — это данные, которые связаны с конкретным объектом. *Методы* — это функция или процедура, принадлежащая какому-то классу или объекту и описывающая их поведение.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

* *Абстракция* для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса;
* *Инкапсуляция* для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;
* *Наследование* для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах;
* *Полиморфизм* для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

Выбор языка C++ обусловлен высокой производительностью, описанными выше свойствами и методами, что позволит максимально раскрыть потенциал языка, а так же значительно ускорить разработку, сделать код более читаемым.

# 5 Функциональное проектирование

В данном разделе пояснительной записки описываются входные и выходные данные программы, используемые классы и методы.

Основные части программы:

* файловая система:
* недвижимость:
* интерфейс;
* ввод данных;
* исключительные ситуации.

# 5.1 Структура входных и выходных данных

Таблица 5.1 - файл дом *House.txt*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Название | ID | Актуальность | Стоимость | Площадь | Адрес | Водоемы | Деревья | Этажи | Комнаты | Количество парковочных мест |
| Домик | 38 | Да | 85000 | 180 | Минск, Приозерная ул., д. 19 | Нет | Да | 2 | 7 | 2 |

Таблица 5.2 - файл квартира *Flat.txt*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Название | ID | Актуальность | Стоимость | Площадь | Адрес | Этаж | Комнаты | Балкон |
| Квартирка | 15 | Да | 28000 | 40 | Минск, Приозерная ул., д. 45, кв. 55 | 5 | 2 | Да |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Количество парковочных мест |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |

Таблица 5.3 - файл коммерческое помещение *Commercial.txt*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Название препарата | ID | Актуальность | Стоимость | Площадь | Адрес | Этаж | Комнаты | Балкон |
| Вазилин | 37 | Да | 28000 | 40 | Минск, Приозерная ул., д. 45, кв. 55 | 5 | 2 | Да |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип | Парковочные места |  |  |  |  |  |  |  |
| Офис | 5 |  |  |  |  |  |  |  |

Таблица 5.4 - файл пустой участок *Empty.txt*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Пригодность почвы | Постройки |  |  |  |  |  |  |
| 38 | Да | Нет |  |  |  |  |  |  |

Таблица 5.5 - файл участок *Piece.txt*

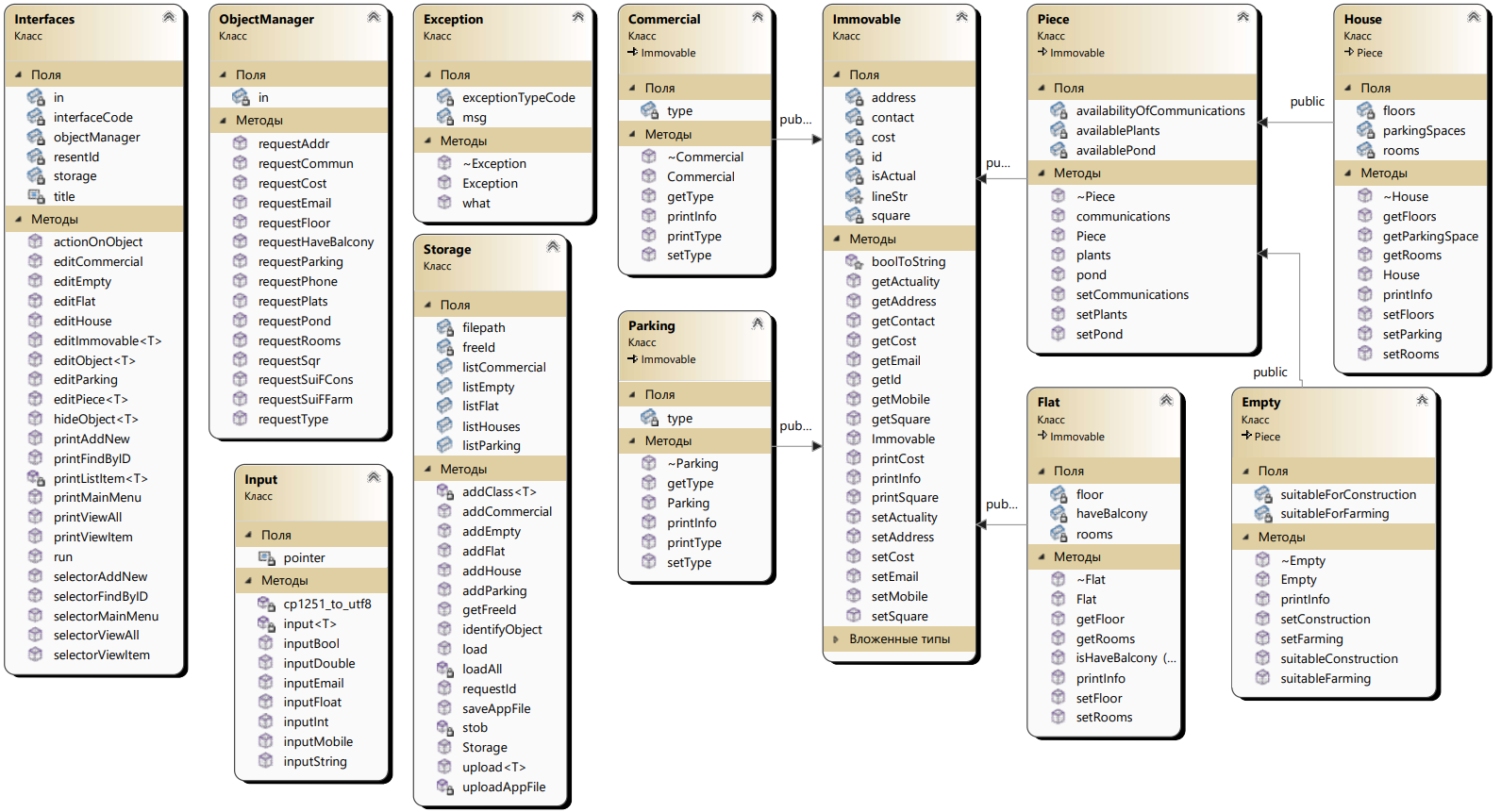
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Водоем | Газон | Коммуникации |  |  |  |  |  |
| 38 | Нет | Да | Электричество, газ, вода, канализация |  |  |  |  |  |

Таблица 5.6 - файл контактные данные *ContactDetails.txt*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Телефон | Email | Telegram |  |  |  |  |  |
| 38 | +375688888 | bam@gmail.com | @mishisha |  |  |  |  |  |

# 5.2 Разработка диаграммы классов

Диаграмма классов представлена в приложении А



# 5.3 Описание классов

Класс Storage.

Поля класса:

― filepath = "app.txt". Данная константа содержит имя файла, который хранит в себе свободное значение ID;

― поле freeId. Содержит свободное значение уникального номера (ID);

― vector<House>. Динамический массив, который хранит в себе объекты класса House;

― vector<Empty>. Динамический массив, который хранит в себе объекты класса Empty;

― vector<Flat>. Динамический массив, который хранит в себе объекты класса Flat;

― vector<Parking>. Динамический массив, который хранит в себе объекты класса Parking;

― vector<Commercial>. Динамический массив, который хранит в себе объекты класса Commercial;

Методы класса:

― Storage(): конструктор по умолчанию, используя конструкцию try{…}catch… вызывает метод uploadAppFile(), а после вызывает метод loadAll();

― uploadAppFile(): этот метод открывает файл filepath и заносит значение из файла в freeId. В случае отсутствия файла, он его создаёт и заносит туда 0;

― saveAppFile(): этот метод открывает файл filepath (в случае отсутствия он его создаёт) и записывает в него значение поля freeId;

― upload(T className): этот шаблонный метод создаёт файл по типу \*id\*.txt, где id - уникальный номер класса который был передан(className). После он записывает значение полей класса в только что созданный файл;

― load(unsigned int id): этот метод используя полученное значение id ищет среди файлов необходимый, после считывает файл, анализирует и идентифицирует. Потом на основе полученной информации создаёт копию этого объекта в динамической памяти;

― loadAll(): этот метод циклически вызывает load(unsigned int id) пока не будут загружены все существующие объекты;

― addClass(T className): этот метод полученный объект загружает и в локальную, и в динамическую память;

― addHouse(): этот метод используя запросы из класса ObjectManager создаёт объект типа House, затем вызывает метод addClass(T className), где в качестве параметра передаётся только что созданный объект;

― addEmpty(): этот метод используя запросы из класса ObjectManager создаёт объект типа Empty, затем вызывает метод addClass(T className), где в качестве параметра передаётся только что созданный объект;

― addFlat(): этот метод используя запросы из класса ObjectManager создаёт объект типа Flat, затем вызывает метод addClass(T className), где в качестве параметра передаётся только что созданный объект;

― addParking(): этот метод используя запросы из класса ObjectManager создаёт объект типа Parking, затем вызывает метод addClass(T className), где в качестве параметра передаётся только что созданный объект;

― addCommercial(): этот метод используя запросы из класса ObjectManager создаёт объект типа Commercial, затем вызывает метод addClass(T className), где в качестве параметра передаётся только что созданный объект;

― getFreeId(): этот метод возвращает значение поля freeId;

― requestId(): этот метод возвращает значение поля freeId и увеличивает его на один;

― identifyObject(unsigned int id): этот метод используя переданный параметр id, открывает необходимый файл и определяет его тип.

Класс Interfaces.

Методы класса:

― printMainMenu(): данный метод выводит на экран меню.

― printFindByID(): данный метод выводит на экран “Поиск по ID”.

― identifyObject(int id): этот метод используя переданный параметр id, открывает необходимый файл.

― printViewAll(): данный метод выводит на экран интерфейс “Просмотреть всё”.

― printAddNew(): данный метод выводит на экран интерфейс “Добавить новый”.

― editObject(T &object): данный шаблонный метод позволяет отредактировать основные параметры объекта.

― editImmovable(T &object): данный шаблонный метод позволяет внести изменения в общие параметры, присущие всем классам.

― editHouse(House &object): данный метод позволяет внести изменения параметров доступных, только для класса House.

― editFlat(Flat &object): данный метод позволяет внести изменения параметров доступных, только для класса Flat.

― editCommercial(Commercial &object(): данный метод позволяет внести изменения параметров доступных, только для класса Commercial.

Класс Exception.

Поля класса:

― поле msg: содержит описание исключительной ситуации;

― поле exceptionTypeCode: содержит код ошибки

Методы класса:

― Exception(const string &msg, int type = 0): конструктор инициализирует поля класса, используя переданные параметры;

― метод what(): Вывод в консоль msg.

Класс Input.

― input(T minValue = NULL, T maxValue = NULL): данный шаблонный метод позволяет вводить значения только в приделах от minValue до maxValue;

― cp1251\_to\_utf8(const char \*str): данный метод преобразовывает массив символов с кодировкой cp1251 в строку c кодировкой UTF-8;

― inputInt(int minValue, int maxValue, const string &msg): данный метод выводит на экран запрос на ввод числа и вызывает шаблонный метод input, после возвращает получение значение;

― inputFloat(float minValue, float maxValue, const string &msg): данный метод выводит на экран запрос на ввод числа и вызывает шаблонный метод input, после возвращает получение значение;

― inputDouble(double minValue, double maxValue, const string &msg): данный метод выводит на экран запрос на ввод числа и вызывает шаблонный метод input, после возвращает получение значение;

― inputMobile(): данный метод выводит запрос на ввод номера телефона и проверят результат ввода на действительность;

― inputEmail(): данный метод выводит запрос на ввод email и проверят результат ввода на действительность;

― inputString(const string &question): данный метод выводит на экран сообщение запроса (параметр question) и принимает строку в качестве ответа. Т.к строку приняли в кодировке cp1251 она преобразовывается в UTF-8, чтобы в дальнейшем строка корректно отображалась и записывалась в файл ;

― inputBool(const string &question): данный метод выводит на экран сообщение запроса (параметр question) и занимается преобразованием строкового, понятного для человека ответа в булевой тип данных (например: Да -> 1).

Класс Immovable.

Поля класса:

― поле id: содержит уникальный номер.

― поле isActual: содержит булевое значение актуальность (по умолчанию оно выставлено как true).

― поле cost: содержит значение о стоимости недвижимости.

― поле square: содержит значение о общей площади недвижимости.

― поле address: содержит строковое значение с адресом недвижимости.

Методы класса:

― метод printInfo(): данный метод выводит в консоль ID и акутальность.

― метод getId(): данный метод возвращает значение поля id.

― метод printCost(): данный метод выводит в консоль стоимость недвижимости.

― метод printSquare(): данный метод выводит в консоль общую площадь недвижимости.

― гетеры и сетеры для полей.

Класс ContactDetails.

Поля класса:

― поле phoneNumber: содержит строку с номером телефона.

― поле email: содержит пустую строку или строку со значением email.

Методы класса:

― getContactDetails(): данный метод возвращает строковое значение, содержащее информацию для связи.

― гетеры и сетеры для полей.

Класс Piece(Plot-участок), участок является наследником класса Immovable и дополнительно содержит следующие поля:

― поле availablePond: содержит значение о наличии водоёмов на участке;

― поле availablePlants: содержит значение о наличии деревьев/кустарниках на участке;

― поле availabilityOfCommunications: содержит значение о проведённых коммуникациях на участке;

Методы класса:

― гетеры и сетеры для полей.

Класс Empty(пустой участок), почва является наследником класса Piece(Plot).

Поля класса:

― поле suitableForConstruction: содержит значение о пригодности почвы для строительства;

― поле suitableForFarming: содержит значение о пригодности почвы для фермерства.

― гетеры и сетеры для полей.

Класс House, является наследником класса Piece.

Поля класса:

― поле floors: содержит значение о количестве этажей в здании;

― поле rooms: содержит значение о количестве жилых комнат в здании;

― поле parkingSpaces: содержит значение о количестве парковочных мест.

Методы класса:

― printInfo() override: данный метод является переписанным и вводит на экран информацию о классе.

― гетеры и сетеры для полей.

Класс Flat, является наследником класса Immovable.

Методы класса:

― поле floors: содержит номер этажа на котором расположена квартира;

― поле rooms: содержит значение о количестве жилых комнат в квартире;

― поле haveBalcony: содержит значение о наличии балкона.

Методы класса:

― printInfo() override: данный метод является переписанным и вводит на экран информацию о недвижимости.

― гетеры и сетеры для полей.



**6 Разработка программных модулей**

Все классы можно разделить на три основных группы:

― Services (Сервисные), к этой группе мы можем отнести все классы, которые помогают программе с обработкой различных процессов. Например для реализации отлова исключительных ситуаций был создан класс Exception. К этой группе так же относится класс Input этот класс предназначен для обработки всех вводимых пользователем параметров. Класс ObjectManager содержит перечень вопросов, которые используются при редактировании/создании нового объекта недвижимости. А также класс Storage, который является ядром программы.

― Immovable что переводится как недвижимость, в этой группе собраны все классы которые описывают различные типы недвижимости и на основе этих классов создаются объекты.

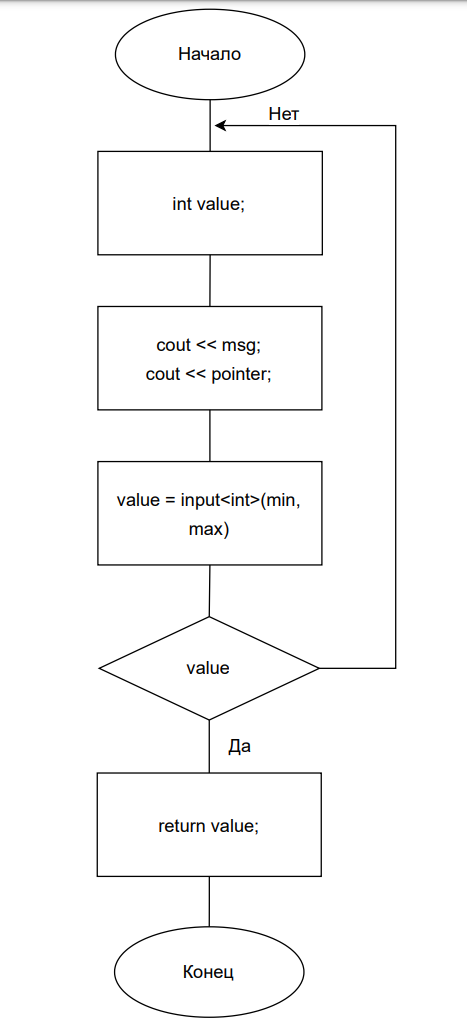
― Interfaces (Интерфейсы) к этой группе относиться класс Interfaces, который ответственен за интерфейс(меню) программы.

**6.1 Разработка схем алгоритмов**

Для ввода целочисленного используется функция inputInt, принадлежащая классу Input. Для работы это функция должна принимать два целочисленных значения, которые описывают минимальную и максимальную границу допустимых значений при вводе. Так же эта функция может принимать строку, которая будет выводиться перед вводом значения, например эта строка может содержать вопрос или пояснения.

Для ввода так же используется шаблонная функция input принадлежащая классу Input. Данная шаблонная функция может использоваться для ввода любых числовых значений:

Схемы алгоритмов приведены в приложении Б



**6.2 Разработка алгоритмов**

Код с пояснениями функции ввода целочисленного inputInt, принадлежащей классу Input:

int inputInt(int minValue, int maxValue, const string &msg = “”) {

int value;

while (true) {

cout << msg; *// Выводим msg, если имеется значение*

cout << pointer; *// Выводим “> ”, для улучшения понятности интерфейса*

try {

// Вызываем шаблонную функцию и возвращаем полученный результат

return input<int>(minValue, maxValue);

} catch (Exception ex) {

ex.what(); // В случае ошибки выведет причину

}

}

return -1;

}

Код с пояснениями шаблонной функции ввода input принадлежащей классу Input. Данная шаблонная функция может использоваться для ввода любых числовых значений:

template<typename T>

T input(T minValue, T maxValue) {

T value; // создаём переменную которая будет хранить результат ввода

cin.clear(); // сброс битов ошибок входного стандартного потока

fflush(stdin); // очистка входного потока

cin >> value; // ввод значения

if (cin.fail()) // проверяем на корректность ввода

throw Exception("Введено неверное значение.");

// проверяем что бы значение было в пределах допустимого

if (value > maxValue || value < minValue)

throw Exception("Значение должно быть в пределах от " + toString(minValue) + " до " + toString(maxValue) + ".", 0);

return value; // возвращаем значение, если все проверки были пройдены

}

**7 Результаты работы программы**

# ЗАКЛЮЧЕНИЕ

В ходе курсового проектирования были решены следующие задачи:

* собрана необходимая информация для реализации проекта;
* продумана структура программы;
* реализация проекта;
* тестирование и отладка программы;
* разработка документации.

Была разработана программа «Информационная система по продаже недвижимости».

Разработанная программа запускается на операционных системах семейства Windows начиная с Windows 7 x64 при наличии процессора с т.ч. 1GHz, оперативной памяти объёмом 500Mb и 5Mb свободного места на жёстком диске.

# ЛИТЕРАТУРА

1. Страуструп, Б. Язык программирования С++ / Б. Страуструп; специальное издание. Пер с англ. - СПб.: BHV, 2008
2. Шилдт, Г. С++: базовый курс / Г. Шилдт - М.: Вильямс; специальное издание. Пер. с англ. - М.: Вильямс, 2007
3. Луцик Ю. А. Объектно-ориентированное программирование на языке С++ / Ю. А. Луцик, А. М. Ковальчук, И. В. Лукьянова - Мн.: БГУИР, 2003
4. Прата, С. - Язык программирования C++. Лекции и упражнения / С. Прата; специальное издание. Пер. с англ. - М.: Вильямс, 2018

**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Структурная схема

**ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Схема алгоритма

**ПРИЛОЖЕНИЕ В**

*(обязательное)*

Код программы

main.cpp

#include <iostream>

#include <windows.h>

#include "interfaces/Interfaces.h"

using namespace std;

int main() {

SetConsoleOutputCP(CP\_UTF8);

SetConsoleCP(1251);

Interfaces myInterface;

try {

myInterface.run();

} catch (Exception exception) {

exception.what();

} catch (...) {

unexpected();

}

return 0;

}

Immovable.h

#ifndef COURSEWORK\_IMMOVABLE\_H

#define COURSEWORK\_IMMOVABLE\_H

#include <iostream>

#include <iomanip>

#include <vector>

using namespace std;

class Immovable {

class ContactDetails {

string phoneNumber;

string email;

public:

ContactDetails(const string &phonePrimary = nullptr, const string &email = nullptr) {

setMobile(phonePrimary);

setEmail(email);

}

void setMobile(const string &phone) {

phoneNumber = phone;

}

string getMobile() {

return phoneNumber;

}

void setEmail(const string &email) {

this->email = email;

}

string getEmail() {

return email;

}

string getContactDetails() {

if (phoneNumber.empty() && email.empty())

return "Нет контактной информации";

string str = " Для связи:";

if (!phoneNumber.empty())

str = str + "\nМобильный: " + phoneNumber;

if (!email.empty())

str = str + "\nE-mail: " + email;

return str;

}

};

unsigned int id;

bool isActual;

float cost;

double square;

string address;

ContactDetails \*contact;

protected:

string lineStr = "----------------------------\n\n";

string boolToString(bool positive) {

if (positive)

return "Да";

return "Нет";

}

public:

Immovable(unsigned int id, const string &phone = nullptr, const string &email = nullptr, float cost = -1,

double square = -1, const string &address = nullptr, bool actuality = true) {

this->id = id;

contact = new Immovable::ContactDetails(phone, email);

setCost(cost);

setSquare(square);

setAddress(address);

setActuality(actuality);

}

virtual void printInfo() = 0;

// contact

string getContact() {

return contact->getContactDetails();

}

void setMobile(const string &newMobile) {

contact->setMobile(newMobile);

}

string getMobile() {

return contact->getMobile();

}

void setEmail(const string &newEmail) {

contact->setEmail(newEmail);

}

string getEmail() {

return contact->getEmail();

}

// address

void setAddress(const string &value) {

address = value;

}

string getAddress() {

return address;

}

// square

void setSquare(double value) {

square = value;

}

string printSquare() {

stringstream stream;

stream << fixed << setprecision(1) << square;

return stream.str() + " м^2";

}

double getSquare() {

return square;

}

// cost

void setCost(float value) {

cost = value;

}

string printCost() {

stringstream stream;

stream << fixed << setprecision(2) << cost;

return stream.str() + "$";

}

float getCost() {

return cost;

}

// id

int getId() {

return id;

}

// actuality

void setActuality(bool isActual) {

this->isActual = isActual;

}

bool getActuality() {

return isActual;

}

};

#endif //COURSEWORK\_IMMOVABLE\_H

Commercial.h

#ifndef COURSEWORK\_COMMERCIAL\_H

#define COURSEWORK\_COMMERCIAL\_H

#include "../Immovable.h"

#include "../../services/Input.h"

class Commercial : public Immovable {

int type;

public:

Commercial(unsigned int id, const string &phone, const string &email, float cost, double square,

const string &address, bool actuality, int type = -1) : Immovable(id, phone, email, cost, square,

address, actuality) {

setType(type);

}

~Commercial() override = default;

void printInfo() override {

cout << " Информация о помещение:" << endl

<< "ID: " << getId();

if (!getActuality())

if (!getActuality()) {

cout << endl << "Недвижимость была скрыта/удалена." << endl << lineStr;

return;

}

cout << endl << "Стоимость: " << printCost() << endl

<< "Адрес: " << getAddress() << endl

<< "Общая площадь: " << printSquare() << endl

<< "Тип: " << printType() << endl

<< getContact() << endl << lineStr;

}

friend ostream &operator<<(ostream &out, Commercial &myClass) {

out << "class commercial\n"

<< "id " << myClass.getId() << '\n'

<< "phone " << myClass.getMobile() << '\n'

<< "email " << myClass.getEmail() << '\n'

<< "cost " << myClass.getCost() << '\n'

<< "sqr " << myClass.getSquare() << '\n'

<< "addr " << myClass.getAddress() << '\n'

<< "actual " << myClass.getActuality() << '\n'

<< "type " << myClass.getType() << '\n';

return out;

};

void setType(int type) {

this->type = type;

}

string printType() {

switch (type) {

case 1:

return "Офис";

case 2:

return "Магазин, торговое помещение";

case 3:

return "Склад";

default:

return "Другое";

}

}

int getType() {

return type;

}

};

#endif //COURSEWORK\_COMMERCIAL\_H

Flat.h

#ifndef COURSEWORK\_FLAT\_H

#define COURSEWORK\_FLAT\_H

#include "../Immovable.h"

class Flat : public Immovable {

int rooms;

int floor;

bool haveBalcony;

public:

Flat(unsigned int id, const string &phonePrimary, const string &email, float cost, double square,

const string &address, bool actuality, int rooms = 2, int floor = 5, bool haveBalcony = true)

: Immovable(id, phonePrimary, email, cost, square, address, actuality) {

setRooms(rooms);

setFloor(floor);

isHaveBalcony(haveBalcony);

}

~Flat() override = default;

void printInfo() override {

cout << " Информация о квартире:" << endl

<< "ID: " << getId();

if (!getActuality()) {

cout << endl << "Недвижимость была скрыта/удалена." << endl << lineStr;

return;

}

cout << endl << "Стоимость: " << printCost() << endl

<< "Адрес: " << getAddress() << endl

<< "Общая площадь: " << printSquare() << endl

<< "Комнат: " << getRooms() << endl

<< "Этаж: " << getFloor() << endl

<< "Есть балкон: " << boolToString(isHaveBalcony()) << endl

<< getContact() << endl << lineStr;

}

friend ostream &operator<<(ostream &out, Flat &myClass) {

out << "class flat\n"

<< "id " << myClass.getId() << '\n'

<< "phone " << myClass.getMobile() << '\n'

<< "email " << myClass.getEmail() << '\n'

<< "cost " << myClass.getCost() << '\n'

<< "sqr " << myClass.getSquare() << '\n'

<< "addr " << myClass.getAddress() << '\n'

<< "actual " << myClass.getActuality() << '\n'

<< "rooms " << myClass.getRooms() << '\n'

<< "floor " << myClass.getFloor() << '\n'

<< "balcony " << myClass.isHaveBalcony() << '\n';

return out;

};

void setFloor(int value) {

floor = value;

}

int getFloor() {

return floor;

}

void setRooms(int value) {

rooms = value;

}

int getRooms() {

return rooms;

}

void isHaveBalcony(bool have) {

haveBalcony = have;

}

bool isHaveBalcony() {

return haveBalcony;

}

};

#endif //COURSEWORK\_FLAT\_H

Parking.h

#ifndef COURSEWORK\_PARKING\_H

#define COURSEWORK\_PARKING\_H

#include "../Immovable.h"

class Parking : public Immovable {

int type;

public:

Parking(unsigned int id, const string &phonePrimary, const string &email, float cost, double square,

const string &address, bool actuality, int type = 3) : Immovable(id, phonePrimary, email, cost, square,

address, actuality) {

setType(type);

}

~Parking() override = default;

void printInfo() override {

cout << " Информация о парковке:" << endl

<< "ID: " << getId();

if (!getActuality()) {

cout << endl << "Недвижимость была скрыта/удалена." << endl << lineStr;

return;

}

cout << endl << "Стоимость: " << printCost() << endl

<< "Адрес: " << getAddress() << endl

<< "Общая площадь: " << printSquare() << endl

<< "Тип: " << printType() << endl

<< getContact() << endl << lineStr;

}

friend ostream &operator<<(ostream &out, Parking &myClass) {

out << "class parking\n"

<< "id " << myClass.getId() << '\n'

<< "phone " << myClass.getMobile() << '\n'

<< "email " << myClass.getEmail() << '\n'

<< "cost " << myClass.getCost() << '\n'

<< "sqr " << myClass.getSquare() << '\n'

<< "addr " << myClass.getAddress() << '\n'

<< "actual " << myClass.getActuality() << '\n'

<< "type " << myClass.getType() << '\n';

return out;

};

void setType(int type) {

this->type = type;

}

string printType() {

switch (type) {

case 1:

return "Машино место";

case 2:

return "Бокс";

case 3:

return "Гараж";

default:

return "Другое";

}

}

int getType() {

return type;

}

};

#endif //COURSEWORK\_PARKING\_H

Empty.h

#ifndef COURSEWORK\_EMPTY\_H

#define COURSEWORK\_EMPTY\_H

#include "Piece.h"

class Empty : public Piece {

bool suitableForConstruction;

bool suitableForFarming;

public:

Empty(unsigned int id, const string &phonePrimary, const string &email, float cost, double square,

const string &address, bool actuality, bool pond, bool plants, bool communications,

bool suitableForConstruction = true, bool suitableForFarming = true) :

Piece(id, phonePrimary, email, cost, square, address, actuality, pond, plants, communications) {

setConstruction(suitableForConstruction);

setFarming(suitableForFarming);

}

~Empty() override = default;

void printInfo() override {

cout << " Информация о участке:" << endl

<< "ID: " << getId();

if (!getActuality()) {

cout << endl << "Недвижимость была скрыта/удалена." << endl << lineStr;

return;

}

cout << endl << "Стоимость: " << printCost() << endl

<< "Адрес: " << getAddress() << endl

<< "Общая площадь: " << printSquare() << endl

<< "Пригодно для строительства: " << boolToString(suitableConstruction()) << endl

<< "Пригодно для фермерства: " << boolToString(suitableFarming()) << endl

<< "Водоемы: " << boolToString(pond()) << endl

<< "Растения: " << boolToString(plants()) << endl

<< "Коммуникации: " << boolToString(communications()) << endl

<< getContact() << endl << lineStr;

}

friend ostream &operator<<(ostream &out, Empty &myClass) {

out << "class empty\n"

<< "id " << myClass.getId() << '\n'

<< "phone " << myClass.getMobile() << '\n'

<< "email " << myClass.getEmail() << '\n'

<< "cost " << myClass.getCost() << '\n'

<< "sqr " << myClass.getSquare() << '\n'

<< "addr " << myClass.getAddress() << '\n'

<< "actual " << myClass.getActuality() << '\n'

<< "pond " << myClass.pond() << '\n'

<< "plant " << myClass.plants() << '\n'

<< "commun " << myClass.communications() << '\n'

<< "suifcons " << myClass.suitableConstruction() << '\n'

<< "suiffarm " << myClass.suitableFarming() << '\n';

return out;

};

void setConstruction(bool suitable) {

suitableForConstruction = suitable;

}

bool suitableConstruction() {

return suitableForConstruction;

}

void setFarming(bool suitable) {

suitableForFarming = suitable;

}

bool suitableFarming() {

return suitableForFarming;

}

};

#endif //COURSEWORK\_EMPTY\_H

House.h

#ifndef COURSEWORK\_HOUSE\_H

#define COURSEWORK\_HOUSE\_H

#include "Piece.h"

class House : public Piece {

int floors;

int rooms;

int parkingSpaces;

public:

House(unsigned int id, const string &phonePrimary, const string &email, float cost, double square,

const string &address, bool actuality,

bool pond, bool plants, bool communications, int floors = 1, int rooms = 4, int parkingSpaces = 2)

: Piece(id, phonePrimary, email, cost, square, address, actuality, pond, plants, communications) {

setFloors(floors);

setRooms(rooms);

setParking(parkingSpaces);

}

~House() override = default;

void printInfo() override {

cout << " Информация о доме/коттедже:" << endl

<< "ID: " << getId();

if (!getActuality()) {

cout << endl << "Недвижимость была скрыта/удалена." << endl << lineStr;

return;

}

cout << endl << "Стоимость: " << printCost() << endl

<< "Адрес: " << getAddress() << endl

<< "Общая площадь: " << printSquare() << endl

<< "Комнат: " << getRooms() << endl

<< "Этажей: " << getFloors() << endl

<< "Парковочный мест: " << getParkingSpace() << endl

<< "Водоемы - " << boolToString(pond()) << endl

<< "Растения - " << boolToString(plants()) << endl

<< getContact() << endl << lineStr;

}

friend ostream &operator<<(ostream &out, House &myClass) {

out << "class house\n"

<< "id " << myClass.getId() << '\n'

<< "phone " << myClass.getMobile() << '\n'

<< "email " << myClass.getEmail() << '\n'

<< "cost " << myClass.getCost() << '\n'

<< "sqr " << myClass.getSquare() << '\n'

<< "addr " << myClass.getAddress() << '\n'

<< "actual " << myClass.getActuality() << '\n'

<< "pond " << myClass.pond() << '\n'

<< "plant " << myClass.plants() << '\n'

<< "commun " << myClass.communications() << '\n'

<< "parking " << myClass.getParkingSpace() << '\n'

<< "rooms " << myClass.getRooms() << '\n'

<< "floor " << myClass.getFloors() << '\n';

return out;

};

void setFloors(int value) {

floors = value;

}

int getFloors() {

return floors;

}

void setRooms(int value) {

rooms = value;

}

int getRooms() {

return rooms;

}

void setParking(int value) {

parkingSpaces = value;

}

int getParkingSpace() {

return parkingSpaces;

}

};

#endif //COURSEWORK\_HOUSE\_H

Piece.h

#ifndef COURSEWORK\_PIECE\_H

#define COURSEWORK\_PIECE\_H

#include "../Immovable.h"

class Piece : public Immovable {

bool availablePond;

bool availablePlants;

bool availabilityOfCommunications;

public:

Piece(unsigned int id, const string &phonePrimary, const string &email, float cost, double square,

const string &address, bool actuality, bool pond = false, bool plants = true, bool communications = false)

: Immovable(id, phonePrimary, email, cost, square, address, actuality) {

setPond(pond);

setPlants(plants);

setCommunications(communications);

}

~Piece() override = default;

void setPond(bool available) {

availablePond = available;

}

bool pond() {

return availablePond;

}

void setPlants(bool available) {

availablePlants = available;

}

bool plants() {

return availablePlants;

}

void setCommunications(bool available) {

availabilityOfCommunications = available;

}

bool communications() {

return availabilityOfCommunications;

}

};

#endif //COURSEWORK\_PIECE\_H

Interfaces.h

#ifndef COURSEWORK\_INTERFACES\_H

#define COURSEWORK\_INTERFACES\_H

#include <iostream>

#include <synchapi.h>

#include "../services/Exception.h"

#include "../services/Storage.h"

#include "../services/ObjectManager.h"

#include "../services/Input.h"

#include "../immovables/piece/House.h"

#include "../immovables/piece/Empty.h"

#include "../immovables/flat/Flat.h"

#include "../immovables/parking/Parking.h"

#include "../immovables/commercial/Commercial.h"

#include "../services/ObjectManager.h"

using namespace std;

class Interfaces {

Storage storage;

Input in;

ObjectManager objectManager;

unsigned int interfaceCode = 0;

unsigned int resentId = 0;

const string title = "\tИнформационная система по продаже недвижимости\n";

template<typename T>

void printListItem(vector<T> array, unsigned int id);

public:

void run();

void printMainMenu(); // 0

int selectorMainMenu();

void printFindByID(); // 100

int selectorFindByID();

void printViewAll(); // 200

int selectorViewAll();

// 210

void printViewItem(string className, unsigned int id, bool advancedMode = false);

int selectorViewItem();

void printAddNew(); // 300

void selectorAddNew();

void actionOnObject(unsigned int id, bool justHide = false);

template<typename T>

bool editObject(T &object);

template<typename T>

void editImmovable(T &object);

template<typename T>

void editPiece(T &object);

void editHouse(House &object);

void editEmpty(Empty &object);

void editFlat(Flat &object);

void editParking(Parking &object);

void editCommercial(Commercial &object);

template<typename T>

void hideObject(T &object);

};

#endif //COURSEWORK\_INTERFACES\_H

Interfaces.cpp

#include <cstdlib>

#include "Interfaces.h"

void Interfaces::run() {

while (true)

switch (interfaceCode) {

case 0:

printMainMenu();

interfaceCode = selectorMainMenu();

break;

case 100:

printFindByID();

interfaceCode = selectorFindByID();

break;

case 110:

interfaceCode = selectorViewItem();

break;

case 111:

actionOnObject(resentId);

interfaceCode = 0;

cout << "\nРедакция успешно завершена." << endl

<< "Возвращение в главное меню через 3 секунд...";

Sleep(3000);

break;

case 112:

actionOnObject(resentId, true);

interfaceCode = 0;

cout << "\nНедвижимость была удалена." << endl

<< "Возвращение в главное меню через 3 секунд...";

Sleep(3000);

break;

case 200:

printViewAll();

interfaceCode = selectorViewAll();

break;

case 300:

printAddNew();

interfaceCode = 0;

selectorAddNew();

break;

case 999:

try {

storage.saveAppFile();

} catch (Exception exception) {

exception.what();

}

exit(EXIT\_SUCCESS);

default:

throw Exception("Неизвестный код интерфейса.", 1);

}

}

void Interfaces::printMainMenu() { // Code 0

system("CLS");

cout << title;

cout << "1. Поиск по ID\n"

"2. Просмотр всей доступной недвижимость\n"

"3. Добавить недвижимость\n"

"0. Выход\n";

}

int Interfaces::selectorMainMenu() {

switch (in.inputInt(0, 3, "Выберите пункт:")) {

case 1:

return 100;

case 2:

return 200;

case 3:

return 300;

case 0:

return 999;

default:

throw Exception("Не существующее значение.", -1);

}

}

void Interfaces::printFindByID() { // Code 100

system("CLS");

cout << title;

cout << "Введите ID('-1' - что-бы вернуться назад):";

}

int Interfaces::selectorFindByID() {

int id = in.inputInt(-1, storage.getFreeId() - 1);

if (id == -1)

return 0;

printViewItem(storage.identifyObject(id), id, true);

return 110;

}

void Interfaces::printViewAll() { // Code 200

system("CLS");

cout << title;

for (unsigned int i = 0; i < storage.getFreeId(); ++i)

printViewItem(storage.identifyObject(i), i);

cout << "Доступные действия:" << endl

<< "1. Выбрать(по ID)" << endl

<< "0. Назад" << endl;

}

int Interfaces::selectorViewAll() {

switch (in.inputInt(0, 1, "Выберите действие:")) {

case 1:

return 100;

case 0:

return 0;

default:

throw Exception("Не существующее значение.", -1);

}

}

void Interfaces::printAddNew() { // Code 300

system("CLS");

cout << title;

cout << "Типы недвижимости:" << endl

<< "1. Дом, Коттедж" << endl

<< "2. Пустой участок" << endl

<< "3. Квартира" << endl

<< "4. Гаражи и стоянки" << endl

<< "5. Коммерческая" << endl

<< "0. Назад" << endl;

}

void Interfaces::selectorAddNew() {

switch (in.inputInt(0, 5, "Выберите тип недвижимости:")) {

case 1:

system("CLS");

cout << "\tДобавить дом/коттедж:" << endl;

storage.addHouse();

break;

case 2:

system("CLS");

cout << "\tДобавить участок:" << endl;

storage.addEmpty();

break;

case 3:

system("CLS");

cout << "\tДобавить квартиру:" << endl;

storage.addFlat();

break;

case 4:

system("CLS");

cout << "\tДобавить гараж/парковку:" << endl;

storage.addParking();

break;

case 5:

system("CLS");

cout << "\tДобавить коммерческое помещение:" << endl;

storage.addCommercial();

break;

case 0:

return;

default:

cout << "Введён неверный тип недвижимости." << endl

<< "Возвращение в главное меню через 5 секунд...";

Sleep(5000);

return;

}

cout << "Объект был добавлен. Возвращение в главное меню через 5 секунд...";

Sleep(5000);

}

template<typename T>

void Interfaces::printListItem(vector<T> array, unsigned int id) {

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

if (array[i].getId() == id) {

array[i].printInfo();

return;

}

}

// Code 210

void Interfaces::printViewItem(string className, unsigned int id, bool advancedMode) {

if (advancedMode) {

resentId = id;

system("CLS");

}

if (className == "house") {

printListItem(storage.listHouses, id);

} else if (className == "empty") {

printListItem(storage.listEmpty, id);

} else if (className == "flat") {

printListItem(storage.listFlat, id);

} else if (className == "parking") {

printListItem(storage.listParking, id);

} else if (className == "commercial") {

printListItem(storage.listCommercial, id);

}

if (advancedMode) {

cout << "Доступные действия:" << endl

<< "1. Редактировать\n"

"2. Удалить\n"

"0. Назад\n";

}

}

int Interfaces::selectorViewItem() {

switch (in.inputInt(0, 2, "Выберите действие:")) {

case 1:

return 111; // edit

case 2:

return 112; // hide

case 0:

return 100; // (come back) to find by ID

default:

throw Exception("Не существующее значение.", -1);

}

}

void Interfaces::actionOnObject(unsigned int id, bool justHide) {

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

if (storage.listHouses[i].getId() == id) {

if (justHide)

hideObject(storage.listHouses[i]);

else {

if (editObject(storage.listHouses[i]))

editHouse(storage.listHouses[i]);

}

storage.upload(storage.listHouses[i]);

return;

}

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

if (storage.listEmpty[i].getId() == id) {

if (justHide)

hideObject(storage.listEmpty[i]);

else {

if (editObject(storage.listEmpty[i]))

editEmpty(storage.listEmpty[i]);

}

storage.upload(storage.listEmpty[i]);

return;

}

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

if (storage.listFlat[i].getId() == id) {

if (justHide)

hideObject(storage.listFlat[i]);

else {

if (editObject(storage.listFlat[i]))

editFlat(storage.listFlat[i]);

}

storage.upload(storage.listFlat[i]);

return;

}

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

if (storage.listParking[i].getId() == id) {

if (justHide)

hideObject(storage.listParking[i]);

else {

if (editObject(storage.listParking[i]))

editParking(storage.listParking[i]);

}

storage.upload(storage.listParking[i]);

return;

}

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

if (storage.listCommercial[i].getId() == id) {

if (justHide)

hideObject(storage.listCommercial[i]);

else {

if (editObject(storage.listCommercial[i]))

editCommercial(storage.listCommercial[i]);

}

storage.upload(storage.listCommercial[i]);

return;

}

}

template<typename T>

bool Interfaces::editObject(T &object) {

if (!object.getActuality()) {

cout << "Доступные действия:\n"

"1. Только сделать актуальным\n"

"2. Сделать актуальным и отредактировать\n"

"3. Только редактировать\n";

int answer = in.inputInt(1, 3, "Выберите действие:");

switch (answer) {

case 1:

object.setActuality(true);

return false;

case 2:

object.setActuality(true);

case 3:

break;

default:

throw Exception("Выбрано недопустимое действие.\n При выборе способа редакции.", 1);

}

}

system("CLS");

bool flag = object.getActuality();

object.setActuality(true);

object.printInfo();

object.setActuality(flag);

editImmovable(object);

return true;

}

template<typename T>

void Interfaces::editImmovable(T &object) {

ObjectManager manager;

try {

// Contacts

object.setMobile(manager.requestPhone());

object.setEmail(manager.requestEmail());

// for Immovable

object.setCost(manager.requestCost());

object.setSquare(manager.requestSqr());

object.setAddress(manager.requestAddr());

} catch (...) {}

}

template<typename T>

void Interfaces::editPiece(T &object) {

ObjectManager manager;

try {

object.setPond(manager.requestPond());

object.setPlants(manager.requestPlats());

object.setCommunications(manager.requestCommun());

} catch (...) {}

}

void Interfaces::editHouse(House &object) {

editPiece(object);

ObjectManager manager;

try {

object.setRooms(manager.requestRooms());

object.setFloors(manager.requestFloor());

object.setParking(manager.requestParking());

} catch (...) {}

}

void Interfaces::editEmpty(Empty &object) {

editPiece(object);

ObjectManager manager;

try {

object.setFarming(manager.requestSuiFFarm());

object.setConstruction(manager.requestSuiFCons());

} catch (...) {}

}

void Interfaces::editFlat(Flat &object) {

ObjectManager manager;

try {

object.setRooms(manager.requestRooms());

object.setFloor(manager.requestFloor(true));

object.isHaveBalcony(manager.requestHaveBalcony());

} catch (...) {}

}

void Interfaces::editParking(Parking &object) {

ObjectManager manager;

try {

object.setType(manager.requestType(true));

} catch (...) {}

}

void Interfaces::editCommercial(Commercial &object) {

ObjectManager manager;

try {

object.setType(manager.requestType());

} catch (...) {}

}

template<typename T>

void Interfaces::hideObject(T &object) {

object.setActuality(false);

}

Exception.h

#ifndef COURSEWORK\_EXCEPTION\_H

#define COURSEWORK\_EXCEPTION\_H

#include <string>

#include <iostream>

using namespace std;

class Exception {

string msg;

int exceptionTypeCode; // -1 - Silent exception | 0 - Error | 1 - Fatal error | 2 - Warning

public:

Exception(const string &msg, int type = 0) {

exceptionTypeCode = type;

this->msg = msg;

}

~Exception() = default;

void what() {

switch (exceptionTypeCode) {

case -1:

cout << msg << endl;

break;

case 0:

cout << "[Error] " << msg << endl;

break;

case 1:

cout << "[ERROR] " << msg << endl;

exit(EXIT\_FAILURE);

case 2:

cout << "[Warning] " << msg << endl;

break;

default:

cout << "[System] Unknown error code. \nError: " << msg << endl;

}

}

};

#endif //COURSEWORK\_EXCEPTION\_H

Input.h

#ifndef COURSEWORK\_INPUT\_H

#define COURSEWORK\_INPUT\_H

#include <string>

#include <iostream>

#include "Exception.h"

using namespace std;

class Input {

const string pointer = "\n> ";

template<typename T>

T input(T minValue = NULL, T maxValue = NULL);

string cp1251\_to\_utf8(const char \*str);

public:

int inputInt(int minValue, int maxValue, const string &msg = "");

float inputFloat(float minValue, float maxValue, const string &msg = "");

double inputDouble(double minValue, double maxValue, const string &msg = "");

string inputMobile();

string inputEmail();

string inputString(const string &question);

bool inputBool(const string &question);

};

#endif //COURSEWORK\_INPUT\_H

Input.cpp

#include <windows.h>

#include <iomanip>

#include "Input.h"

template<typename T>

string toString(T value) {

std::stringstream stream;

stream << fixed << setprecision(2) << value;

return stream.str();

}

template<typename T>

T Input::input(T minValue, T maxValue) {

T value;

cin.clear();

fflush(stdin);

cin >> value;

if (cin.fail())

throw Exception("Введено неверное значение.");

if (value > maxValue || value < minValue)

throw Exception("Значение должно быть в пределах от " + toString(minValue) +

" до " + toString(maxValue) + ".", 0);

return value;

}

int Input::inputInt(int minValue, int maxValue, const string &msg) {

int value;

while (true) {

cout << msg;

cout << pointer;

try {

value = input<int>(minValue, maxValue);

break;

} catch (Exception ex) {

ex.what();

}

}

return value;

}

float Input::inputFloat(float minValue, float maxValue, const string &msg) {

float value;

while (true) {

cout << msg;

cout << pointer;

try {

value = input<float>(minValue, maxValue);

break;

} catch (Exception ex) {

ex.what();

}

}

return value;

}

double Input::inputDouble(double minValue, double maxValue, const string &msg) {

double value;

while (true) {

cout << msg;

cout << pointer;

try {

value = input<double>(minValue, maxValue);

break;

} catch (Exception ex) {

ex.what();

}

}

return value;

}

string Input::inputMobile() {

const string msg = "Введите ваш номер:\n+375";

unsigned long maxValue = 999999999;

unsigned long value;

while (true) {

cout << msg;

try {

value = input<unsigned long>(100000000, maxValue);

break;

} catch (Exception ex) {

ex.what();

}

}

return ("+375" + to\_string(value));

}

string Input::inputEmail() {

string email = "";

cout << "Введите ваш email (Не обязательно):";

cout << pointer;

fflush(stdin);

try {

char c;

while (true) {

c = (char) getchar();

if (c == '\n')

break;

if (c == EOF)

break;

email.push\_back(c);

}

} catch (Exception ex) {

ex.what();

}

bool isEmail = false;

for (char c: email) {

if (c == '@') {

isEmail = true;

break;

}

}

if (!isEmail)

email = "";

return email;

}

string Input::inputString(const string &question) {

string value;

cout << question;

cout << pointer;

cin.clear();

fflush(stdin);

getline(cin, value);

value = cp1251\_to\_utf8(value.data());

return value;

}

bool Input::inputBool(const string &question) {

while (true) {

string answer = inputString(question);

if (answer == "Да" || answer == "да" || answer == "Lf" || answer == "lf" || answer == "y" ||

answer == "Y" || answer == "Yes" || answer == "yes")

return true;

if (answer == "Нет" || answer == "нет" || answer == "Ytn" || answer == "ytn" || answer == "n" ||

answer == "N" || answer == "No" || answer == "no")

return false;

cout << "Непонятный ответ. \nДля ответа используйте такие слова как: да, нет." << endl;

}

}

string Input::cp1251\_to\_utf8(const char \*str) {

std::string res;

WCHAR \*ures = NULL;

char \*cres = NULL;

int result\_u = MultiByteToWideChar(1251, 0, str, -1, 0, 0);

if (result\_u != 0) {

ures = new WCHAR[result\_u];

if (MultiByteToWideChar(1251, 0, str, -1, ures, result\_u)) {

int result\_c = WideCharToMultiByte(CP\_UTF8, 0, ures, -1, 0, 0, 0, 0);

if (result\_c != 0) {

cres = new char[result\_c];

if (WideCharToMultiByte(CP\_UTF8, 0, ures, -1, cres, result\_c, 0, 0)) {

res = cres;

}

}

}

}

delete[] ures;

delete[] cres;

return res;

}

ObjectManager.h

#ifndef COURSEWORK\_OBJECTMANAGER\_H

#define COURSEWORK\_OBJECTMANAGER\_H

#include <string>

#include <fstream>

#include <windows.h>

#include "Exception.h"

#include "Input.h"

using namespace std;

class ObjectManager {

Input in;

public:

string requestPhone() {

return in.inputMobile();

}

string requestEmail() {

return in.inputEmail();

}

float requestCost() {

return in.inputFloat(50, 2000000000, "Введите цену: ");

}

double requestSqr() {

return in.inputDouble(2, 100000000, "Введите общую площадь: ");

}

string requestAddr() {

return in.inputString("Введите адрес:");

}

bool requestPond() {

return in.inputBool("Есть водоёмы?");

}

bool requestPlats() {

return in.inputBool("Есть деревья/кустарники?");

}

bool requestCommun() {

return in.inputBool("Проведены ли коммуникации?");

}

int requestFloor(bool isFlat = false) {

if (isFlat)

return in.inputInt(1, 100, "Введите на каком этаже?");

return in.inputInt(1, 5, "Сколько этажей?");

}

int requestRooms() {

return in.inputInt(1, 100, "Сколько комнат?");

}

int requestParking() {

return in.inputInt(1, 100, "Сколько парковочных мест?");

}

bool requestSuiFCons() {

return in.inputBool("Пригодно ли для строительства?");

}

bool requestSuiFFarm() {

return in.inputBool("Пригодно ли для фермерства?");

}

bool requestHaveBalcony() {

return in.inputBool("Есть ли балкон?");

}

int requestType(bool isParking = false) {

if (isParking) {

string msg = "1. Машино место\n"

"2. Бокс\n"

"3. Гараж\n"

"4. Другое\n"

"Выберите тип:";

return in.inputInt(1, 4, msg);

}

string msg = "1. Офис\n"

"2. Магазин, торговое помещение\n"

"3. Склад\n"

"4. Другое\n"

"Выберите тип:";

return in.inputInt(1, 4, msg);

}

};

#endif //COURSEWORK\_OBJECTMANAGER\_H

Storage.h

#ifndef COURSEWORK\_STORAGE\_H

#define COURSEWORK\_STORAGE\_H

#include <string>

#include <list>

#include <fstream>

#include "../services/Exception.h"

#include "../services/Input.h"

#include "../services/ObjectManager.h"

#include "../immovables/piece/House.h"

#include "../immovables/piece/Empty.h"

#include "../immovables/flat/Flat.h"

#include "../immovables/parking/Parking.h"

#include "../immovables/commercial/Commercial.h"

class Storage {

string filepath = "app.txt";

unsigned int freeId = 0;

// string to bool

bool stob(string str);

void uploadAppFile();

// load all exists objects

void loadAll();

template<typename T>

void addClass(T className);

public:

Storage(); // get actual id

vector<House> listHouses;

vector<Empty> listEmpty;

vector<Flat> listFlat;

vector<Parking> listParking;

vector<Commercial> listCommercial;

void saveAppFile();

// from class to file

template<class T>

void upload(T className);

// from file to list

void load(unsigned int id);

// create & add to file & to list

void addHouse();

void addEmpty();

void addFlat();

void addParking();

void addCommercial();

unsigned int getFreeId();

unsigned int requestId();

// from ID to classType

string identifyObject(unsigned int id);

};

#endif //COURSEWORK\_STORAGE\_H

Storage.cpp

#include "Storage.h"

Storage::Storage() {

try {

uploadAppFile(); // get actual ID

loadAll(); // load all objects

} catch (Exception exception) {

exception.what();

} catch (...) {

unexpected();

}

}

void Storage::uploadAppFile() {

ifstream in;

in.open(filepath);

if (!in.is\_open()) {

saveAppFile();

throw Exception("AppFile не найден.\nAppFile был пересоздан.", 2);

}

in >> freeId;

in.close();

}

void Storage::saveAppFile() {

ofstream out;

out.open(filepath);

if (!out.is\_open())

throw Exception("Невозможно открыть AppFile для записи.", 0);

out << freeId;

out.close();

}

template<class T>

void Storage::upload(T className) {

const string path = to\_string(className.getId()) + ".txt";

ofstream out;

out.open(path);

if (!out.is\_open())

throw Exception("Невозможно открыть файл для записи.", 0);

out << className;

out.close();

}

void Storage::load(unsigned int id) {

const string path = to\_string(id) + ".txt";

ifstream in;

in.open(path);

if (!in.is\_open())

throw Exception("Невозможно открыть файл для чтения.", 0);

// FLAGS

string classType;

// all

string phone;

string email;

float cost = -1;

double sqr = -1;

string addr;

bool actual = false;

// piece

bool pond = false;

bool plant = false;

bool commun = false;

// empty

bool suifcons = false;

bool suiffarm = false;

// house

int parking = -1;

// flat & house

int rooms = -1;

int floor = -1;

// flat

bool balcony = false;

// commercial & parking

int type = -1;

// Set flags

string str;

while (getline(in, str)) {

string name = str.substr(0, str.find(' '));

string value = str.substr(str.find(' ') + 1, str.find('\n'));

if (name == "class")

classType = value;

else if (name == "phone")

phone = value;

else if (name == "email")

email = value;

else if (name == "cost")

cost = stof(value);

else if (name == "sqr")

sqr = stod(value);

else if (name == "addr")

addr = value;

else if (name == "actual")

actual = stob(value);

else if (name == "pond")

pond = stob(value);

else if (name == "plant")

plant = stob(value);

else if (name == "commun")

commun = stob(value);

else if (name == "suifcons")

suifcons = stob(value);

else if (name == "suiffarm")

suiffarm = stob(value);

else if (name == "parking")

parking = stoi(value);

else if (name == "rooms")

rooms = stoi(value);

else if (name == "floor")

floor = stoi(value);

else if (name == "balcony")

balcony = stob(value);

else if (name == "type")

type = stoi(value);

}

// Create object & push it

if (classType == "house") {

House house(id, phone, email, cost, sqr, addr, actual, pond, plant, commun,

floor, rooms, parking);

listHouses.push\_back(house);

} else if (classType == "empty") {

Empty empty(id, phone, email, cost, sqr, addr, actual, pond, plant, commun, suifcons, suiffarm);

listEmpty.push\_back(empty);

} else if (classType == "flat") {

Flat flat(id, phone, email, cost, sqr, addr, actual, rooms, floor, balcony);

listFlat.push\_back(flat);

} else if (classType == "parking") {

Parking parking(id, phone, email, cost, sqr, addr, actual, type);

listParking.push\_back(parking);

} else if (classType == "commercial") {

Commercial commercial(id, phone, email, cost, sqr, addr, actual, type);

listCommercial.push\_back(commercial);

}

in.close();

}

void Storage::loadAll() {

for (int i = 0; i < freeId; ++i) {

load(i); // load objects

Sleep(75);

}

}

template<typename T>

void Storage::addClass(T className) {

try {

upload(className);

load(getFreeId() - 1);

} catch (Exception exception) {

exception.what();

} catch (...) {

unexpected();

}

}

void Storage::addHouse() {

ObjectManager manager;

House house(requestId(), manager.requestPhone(), manager.requestEmail(), manager.requestCost(),

manager.requestSqr(), manager.requestAddr(), true, manager.requestPond(), manager.requestPlats(),

manager.requestCommun(), manager.requestFloor(), manager.requestRooms(), manager.requestParking());

addClass(house);

}

void Storage::addEmpty() {

ObjectManager manager;

Empty empty(requestId(), manager.requestPhone(), manager.requestEmail(), manager.requestCost(),

manager.requestSqr(), manager.requestAddr(), true, manager.requestPond(), manager.requestPlats(),

manager.requestCommun(), manager.requestSuiFCons(), manager.requestSuiFFarm());

addClass(empty);

}

void Storage::addFlat() {

ObjectManager manager;

Flat flat(requestId(), manager.requestPhone(), manager.requestEmail(), manager.requestCost(),

manager.requestSqr(), manager.requestAddr(), true, manager.requestRooms(), manager.requestFloor(true),

manager.requestHaveBalcony());

addClass(flat);

}

void Storage::addParking() {

ObjectManager manager;

Parking parking(requestId(), manager.requestPhone(), manager.requestEmail(), manager.requestCost(),

manager.requestSqr(), manager.requestAddr(), true, manager.requestType(true));

addClass(parking);

}

void Storage::addCommercial() {

ObjectManager manager;

Commercial commercial(requestId(), manager.requestPhone(), manager.requestEmail(), manager.requestCost(),

manager.requestSqr(), manager.requestAddr(), true, manager.requestType());

addClass(commercial);

}

unsigned int Storage::getFreeId() {

return freeId;

}

unsigned int Storage::requestId() {

return freeId++;

}

string Storage::identifyObject(unsigned int id) {

const string path = to\_string(id) + ".txt";

ifstream in;

in.open(path);

if (!in.is\_open())

throw Exception("Невозможно открыть файл для чтения.", 0);

string str;

string value;

while (getline(in, str)) {

string name = str.substr(0, str.find(' '));

value = str.substr(str.find(' ') + 1, str.find('\n'));

if (name == "class")

break;

}

in.close();

return value;

}

bool Storage::stob(string str) {

int i = stoi(str);

if (i)

return true;

return false;

}

**ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Диаграмма классов

**ПРИЛОЖЕНИЕ Д**

*(обязательное)*

Ведомость документов