СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Тема курсовой работы: Информационная система по продаже недвижимости на языке программирования C++.

Цель курсовой работы это написании программы, соответствующей теме курсовой, построенной на основах ООП и использовании STL, предоставляющее пользователю доступный и простой интерфейс.

Информационная система (ИС) – система, предназначенная для хранения, поиска и обработки информации. ИС предназначена для своевременного обеспечения надлежащих людей надлежащей информацией, то есть для удовлетворения конкретных информационных потребностей в рамках определённой предметной области, при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы, базы данных и информационные услуги.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор предметной области

Язык высокого уровня – это язык программирования, предназначенный для программиста; он не зависит от внутренних машинных кодов компьютера любого типа. Языки высокого уровня используют для решения проблем, и поэтому их часто называют проблемно-ориентированными языками. 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.

В нашем случае мы используем язык высокого уровня C++. 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.

Используя эти свойства и методы, можно значительно ускорить разработку, сделать код более читаемым.

1.2 Постановка задачи

В рамках курсового проекта поставлена задача разработать информационную систему по продаже недвижимости, предоставляющий пользователям функциональность добавления, изменения и удаления объявлений о покупке недвижимости.

Минимальные требования к реализации программного продукта:
— хранение данных;
— возможность добавления, изменения и удаления объектов;
— простой и понятный пользовательский интерфейс.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Полностью ознакомившись с теоретическими аспектами реализуемого программного обеспечения, было решено структурно разделить приложение на основные блоки. Такое решение позволяет оптимизировать использование ресурсов. На схеме ГУИР.400201.217 С1 изображена структурная схема данного программного обеспечения.

Из основных блоков в программе можно выделить следующие:

1. Блок интерфейса;
2. Блок файловой системы;
3. Блок ввода данных;
4. Блок обработки исключительных ситуаций;
5. Блок недвижимости.

2.1 Блок интерфейса

Блок интерфейса содержит средства, обеспечивающие взаимодействие между пользователем и приложением.

2.2 Блок файловой системы

Блок файловой системы содержит средства, позволяющие осуществлять работу с фалами, а именно позволяет открывть для чтения и записи различные файлы, также этот блок может создавать файлы. Главная задача этого блока это хранить и обрабатывать данные о недвижимости.

2.3 Блок ввода данных

Блок ввода данных содержит средства, отвечают за форму подачи данных и за их корректность. Использование этого блока позволяет исключить большую часть возникновения исключительных ситуаций.

2.4 Блок обработки исключительных ситуаций

Блок обработки исключительных ситуаций содержит средства, обеспечивающие корректную работу приложения при возникновение исключительных ситуаций.

2.5 Блок недвижимости

Блок недвижимости содержит различные классы, которые описывают методы и свойства разных типов недвижимости.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе пояснительной записки детально рассмотрим функционирование программного обеспечения.

Для выполнения задачи, поставленной в ходе реализации проекта, будет произведён подробный обзор архитектуры реализуемого программного обеспечения, будут рассмотрены основные классы, их строение, связи между классами и другие зависимости, а также используемые методы классов, поля и константы.

Так как в разработке приложения используется объектно-ориентированная парадигма программирования, все модули данного программного обеспечения представлены различными классами, логически объединёнными в зависимости от выполняемой функции. На схеме ГУИР.400201.217 РР.1 изображена диаграмма классов данного программного обеспечения.

Основным блоком разрабатываемого приложения является блок файловой системы. Данный блок выполняет следующие функции:

- локальное хранение данных о недвижимости (создания, чтения и записи файлов);
- обработка данных о недвижимости (создание новых и редактирование уже существующих объектов).

Оставшимися, но не менее важными блоками приложения, являются следующие блоки:

- блок исключительных ситуаций;
- блок интерфейса;
- блок ввода данных;
- блок недвижимости.

Функциональное представление каждого из вышеперечисленных блоков приведено в подраздел, описанных ниже.

3.1 Блок файловой системы

Файловая система описана в классе `Storage`. Этот класс содержит такие свойства как:

— `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;

Так же класс Storage содержит следующие методы:

— 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.

3.2 Блок исключительных ситуаций

Блок исключительных ситуаций описан в классе `Exception`. Этот класс содержит такие свойства как:

— поле `msg`: содержит описание исключительной ситуации;

— поле `exceptionTypeCode`: содержит код ошибки (коды исключительных ситуаций описаны в таблице 3.1).

Таблица 3.1 - Описание кодов исключительных ситуаций

Код	Тип исключения	Описание
-1	Уведомление	Выводит сообщение без тегов
0	Ошибка	Выводит сообщение с тегом [Error]
1	Критическая ошибка	Выводит сообщение с тегом [Error] и принудительно завершает работу программы
2	Предупреждение	Выводит сообщение с тегом [Warning]

Класс `Exception` содержит следующие методы:

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

— метод `what()`: Вывод в консоль `msg`.

3.3 Блок интерфейса

Блок интерфейса описан в классе `Interfaces`. Этот класс содержит следующие свойства:

— поле `interfaceCode`: содержит код интерфейса. По умолчанию код = 0 (коды и их описание приведены в таблице 3.2);

— поле `resentId`: содержит `id` недавно просмотренного объекта.

Таблица 3.2 - Описание кодов интерфейса.

Код	**Описание**
0	Главное меню
100	Поиск по ID
110	Просмотреть объект
111	Редактировать объект
112	Удаление объекта
200	Просмотреть все объекты
300	Добавить новый объект
999	Выход

Ещё класс `Interfaces` содержит следующие методы:

— `run()`: данный метод является бесконечным циклом, внутри которого мы можем переключаться среди интерфейсов, и выйти из цикла можно только выбрав интерфейс под кодом 999;

— `printMainMenu()`: данный метод выводит на экран интерфейс “Главное меню”;

— `selectorMainMenu()`: данный метод используя класс `Input` обеспечивает выбор действия;

— `printFindByID()`: данный метод выводит на экран интерфейс “Поиск по ID”;

— `selectorFindByID()`: данный метод используя класс `Input` обеспечивает выбор действия;

— `printViewAll()`: данный метод выводит на экран интерфейс “Просмотреть всё”;

— `selectorViewAll()`: данный метод используя класс `Input` обеспечивает выбор действия;

— `printAddNew()`: данный метод выводит на экран интерфейс “Добавить новый”;

— `selectorAddNew()`: данный метод используя класс `Input` предоставляет выбор типа недвижимости, после переадресовывает на более конкретный метод из класса `Storage` (Например метод `addHouse()`);

— `printListItem()`: данный метод находит объект в массиве и выводит на экран информацию о объекте;

— `printViewItem()`: данный метод выводит на экран интерфейс “Просмотреть объект”;

— `selectorViewItem()`: данный метод используя класс `Input` обеспечивает выбор действия;

— `actionOnObject(unsigned int id, bool justHide)`: данный метод находит по id необходимый объект и предоставляет выбор редактировать или удалить/скрыть;

— `hideObject(T &object)`: данный шаблонный метод позволяет удалить/скрыть полученный объект;

— `editObject(T &object)`: данный шаблонный метод позволяет отредактировать основные параметры объекта, так же именно этот метод позволяет восстановить объект после удаления;

— `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`;

— `editEmpty(Empty &object)`: данный метод позволяет внести изменения параметров доступных, только для класса `Empty`;

— `editFlat(Flat &object)`: данный метод позволяет внести изменения параметров доступных, только для класса `Flat`;

— `editParking(Parking &object)`: данный метод позволяет внести изменения параметров доступных, только для класса `Parking`;

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

3.4 Блок ввода данных

Блок ввода данных описан в классе `Input`. Этот класс содержит только константное поле `pointer` = “\n>”: единственная задача которого, сделать интерфейс понятнее.

Помимо этого класс содержит следующие методы:

— `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 с 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 ср1251 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).

3.5 Блок недвижимости

Блок недвижимости состоит из нескольких классов.

3.5.1 Класс `Immovable` содержит следующие поля:

— 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;

Так же класс `Immovable` содержит контейнерный класс `ContactDetails, а ещё следующие методы:`

— виртуальный метод `printInfo()` = 0;

— 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;

И ещё есть гетеры и сетеры для полей `isActual`, `cost`, `square`, `address`.

3.5.2 Класс `ContactDetails` содержит следующие поля:

— поле `phoneNumber`: содержит строку с номером телефона;

— поле `email`: содержит пустую строку или строку со значением email;

Так же класс `ContactDetails` содержит гетеры и сетеры для полей `phoneNumber`, `email`и метод:

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

3.5.3 Класс `Piece`, 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;

И гетеры и сетеры для полей `availablePond`, `availablePlants`, `availabilityOfCommunications`.

3.5.4 Класс `Empty`, является наследником класса `Piece` и дополнительно содержит следующие поля:

— поле `suitableForConstruction`: содержит значение о пригодности почвы для строительства;

— поле `suitableForFarming`: содержит значение о пригодности почвы для фермерства.

Помимо этого класс содержит следующие методы:

— `printInfo() override`: данный метод является переписанным и вводит на экран информацию о классе;

— Перегруженный оператор <<, который возвращает значение всех полей класса.

Так же гетеры и сетеры для полей `suitableForConstruction` и `suitableForFarming`.

3.5.5 Класс `House`, является наследником класса `Piece` и дополнительно содержит следующие поля:

— 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.

Помимо этого класс содержит следующие методы:

— `printInfo() override`: данный метод является переписанным и вводит на экран информацию о классе;

— Перегруженный оператор <<, который возвращает значение всех полей класса.

Так же гетеры и сетеры для полей `floors`, `rooms` и `parkingSpaces`.

3.5.6 Класс `Flat`, является наследником класса `Immovable` и дополнительно содержит следующие поля:

— 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.

Помимо этого класс содержит следующие методы:

— `printInfo() override`: данный метод является переписанным и вводит на экран информацию о недвижимости;

— Перегруженный оператор <<, который возвращает значение всех полей класса.

Так же гетеры и сетеры для полей `floors`, `rooms` и `haveBalcony`.

3.5.7 Класс `Parking`, является наследником класса `Immovable` и дополнительно содержит поле `type`. Это поле содержит значение о типе парковки.

Помимо этого класс содержит следующие методы:

— `printInfo() override`: данный метод является переписанным и вводит на экран информацию о недвижимости;

— `printType()`: данный метод вводит на экран информацию о типе недвижимости;

— Перегруженный оператор <<, который возвращает значение всех полей класса.

А так же гетеры и сетеры для поля `type`.

3.5.8 Класс `Commercial`, является наследником класса `Immovable` и дополнительно содержит поле `type`. Это поле содержит значение о типе помещения.

Помимо этого класс содержит следующие методы:

— `printInfo() override`: данный метод является переписанным и вводит на экран информацию о недвижимости;

— `printType()`: данный метод вводит на экран информацию о типе недвижимости;

— Перегруженный оператор <<, который возвращает значение всех полей класса.

А так же гетеры и сетеры для поля `type`.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Все классы мы можем 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 собственный класс Exception. Помимо класса 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 этот класс отвечает за хранение объектов как локально так и оперативной памяти. Класс Storage является ядром программы.

— Immovable что переводиться как недвижимость, в этой группе собранны все классы которые описывают различные типы недвижимости и на основе этих классов создаются объекты.

— Interfaces (Интерфейсы) к этой группе относиться класс Interfaces, который как следует из названия ответственен за интерфейс программы.

Для примера рассмотрим алгоритм ввода целочисленного. 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 ввода значения, например эта строка может содержать вопрос или пояснения.

На схеме ГУИР.400201.217 ПД1 изображена схема данного алгоритма.

Ниже приведён код этой функции:

```
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;
}
```

Для ввода так же используется шаблонная функция input принадлежащая классу Input. Данная шаблонная функция может использоваться для ввод 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
}
```

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Результатом выполнения курсового проекта стало приложение для платформы Windows.

5.1 Требования к аппаратному обеспечению

Для нормального функционирования программы требуется:

- Операционные системы Windows 7 x64 и старше;
- Процессор с тактовой частотой 1 GHz или больше;
- Объем оперативной памяти 500Mb и больше;
- 5Mb свободного места на жёстком диске.

5.2 Руководство пользователя

Перед использованием рекомендуется создать отдельную папку и поместить в папку исполнимый файл.

Для использования приложение не требует установки. Для использования необходимо запустить файл с именем «Coursework.exe». После откроется консоль с главным меню (рисунок 5.1) (см. п. 5.2.1).

5.2.1 Главное меню

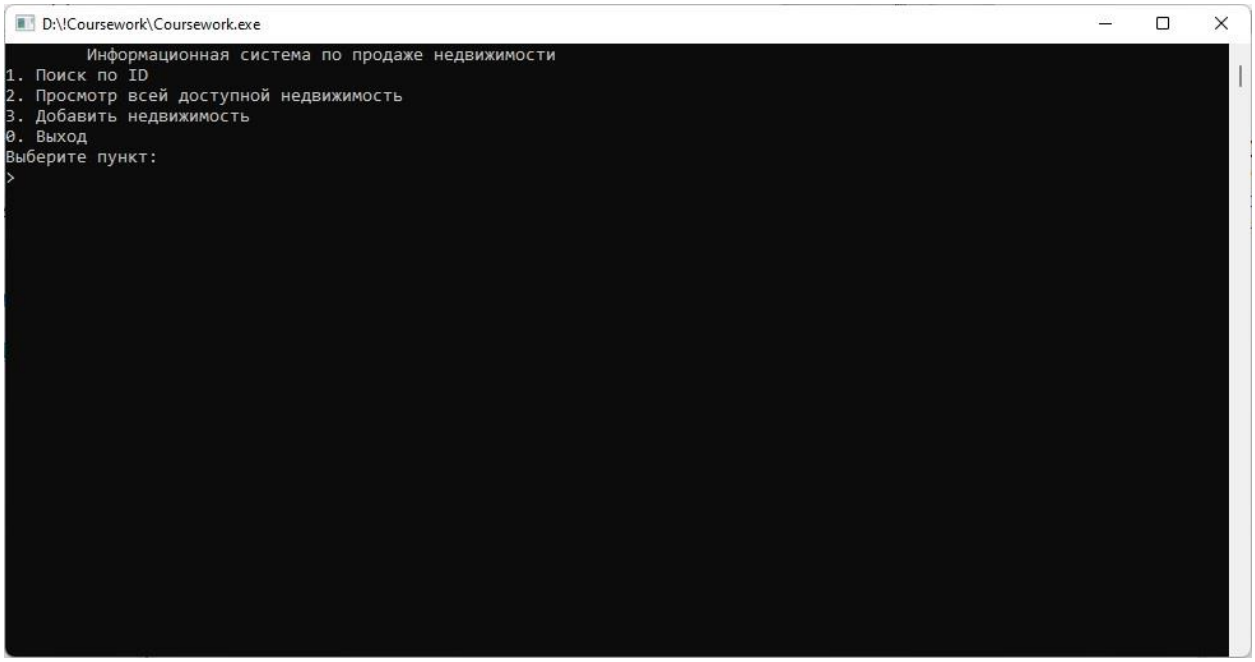

Рисунок 5.1 – Главное меню

В главном меню (рисунок 5.1) нам предоставляется выбор. Для совершения выбора нам необходимо ввести значение, которое будет соответствовать номеру пункта. Из главного меню мы можем:

– найти объект по ID (уникальному номеру) для этого необходимо ввести «1» (см. п. 5.2.2);

– просмотр всей недвижимости для этого необходимо ввести «2» (см. п. 5.2.3);

– добавить новый объект недвижимости для этого необходимо ввести «3» (см. п. 5.2.5);

– завершить работу приложения для этого необходимо ввести «0».

5.2.2 Меню поиска по ID

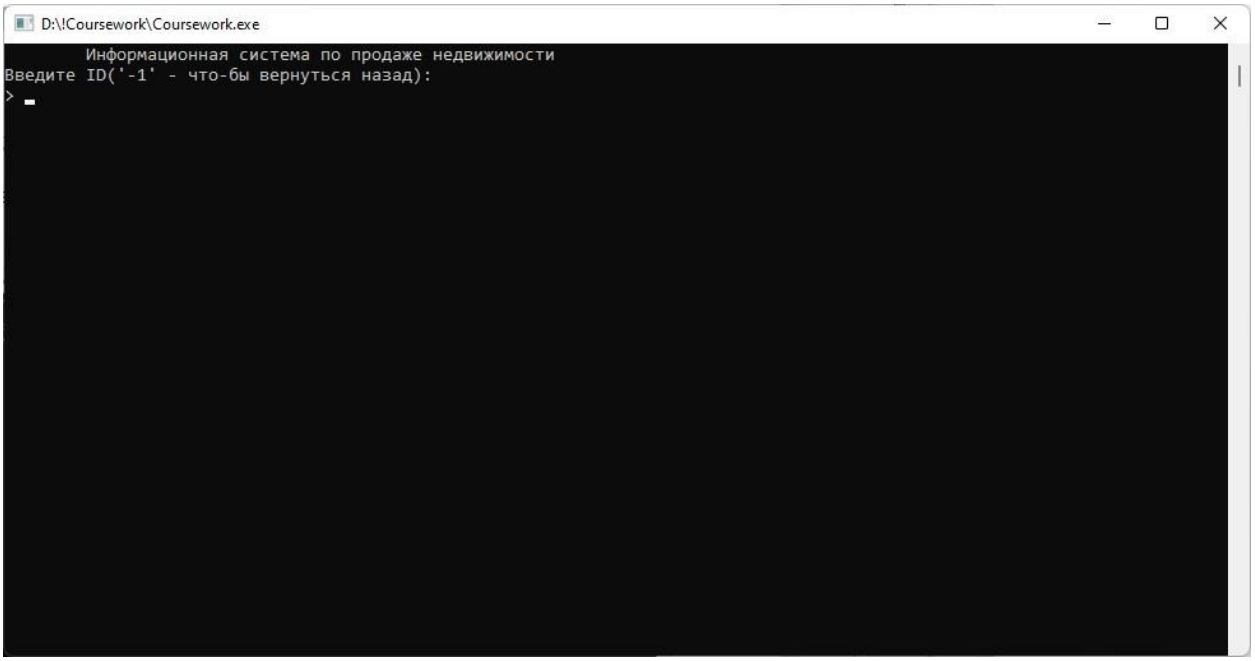

Рисунок 5.2 – Меню поиска по ID

В меню поиска по ID (рисунок 5.2) нам предоставляется выбор. Из меню поиска по ID мы можем:

– выбрать объект для этого необходимо ввести ID (см. п. 5.2.4);

– вернуться в главное меню для этого необходимо ввести «-1» (см. п. 5.2.1).

5.2.3 Меню просмотра всей недвижимости

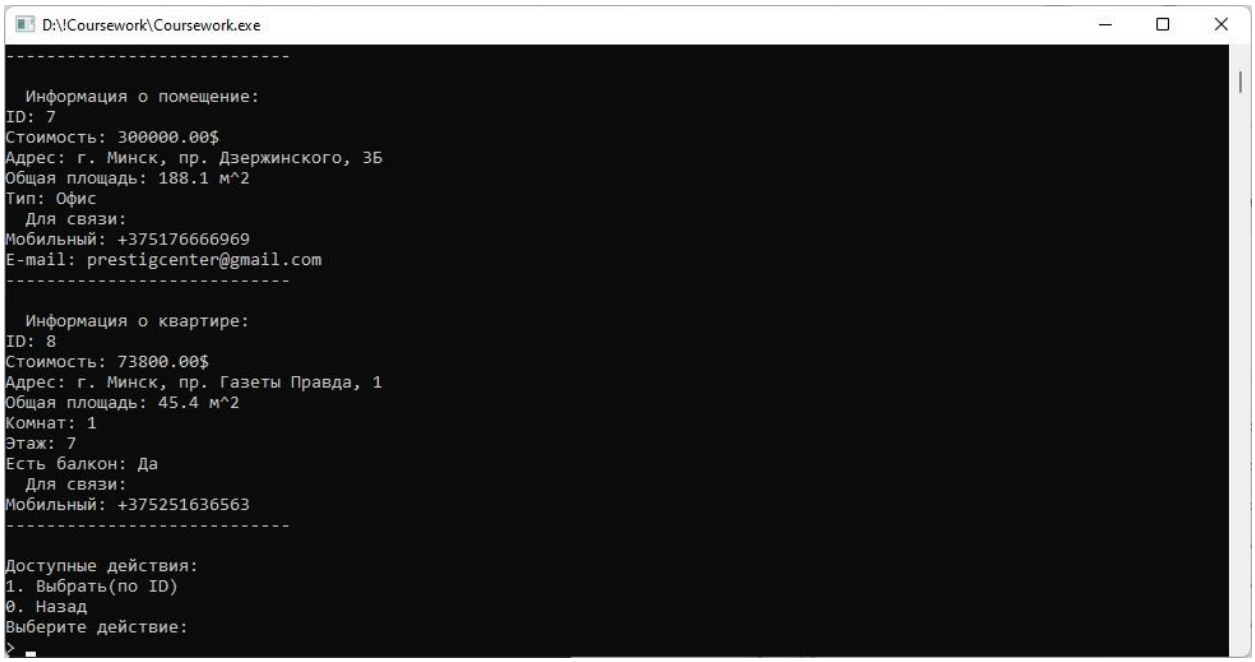

Рисунок 5.3 – Меню просмотра всей недвижимости

В этом меню мы можем просмотреть всю недвижимость. Из этого меню мы можем:

– найти объект по ID для этого необходимо ввести «1» (см. п. 5.2.2);

– вернуться в главное меню для этого необходимо ввести «0» (см. п. 5.2.1).

5.2.4 Меню объекта

Рисунок 5.4 – Меню объекта

В меню объекта (рисунок 5.4) нам предоставляется выбор. Из меню объекта мы можем:

– редактировать объект для этого необходимо ввести ввести «1». После ввода объект будет пройдено анкетирование в котором необходимо указать новые параметры и через пару секунд мы вернёмся в главное меню (см. п. 5.2.1);

– удалить/скрыть объект для этого необходимо ввести «2». После ввода объект будет удалён/скрыт и через пару секунд мы вернёмся в главное меню (см. п. 5.2.1);

– вернуться в меню поиск по ID для этого необходимо ввести «0» (см. п. 5.2.2).

5.2.5 Меню добавления объекта

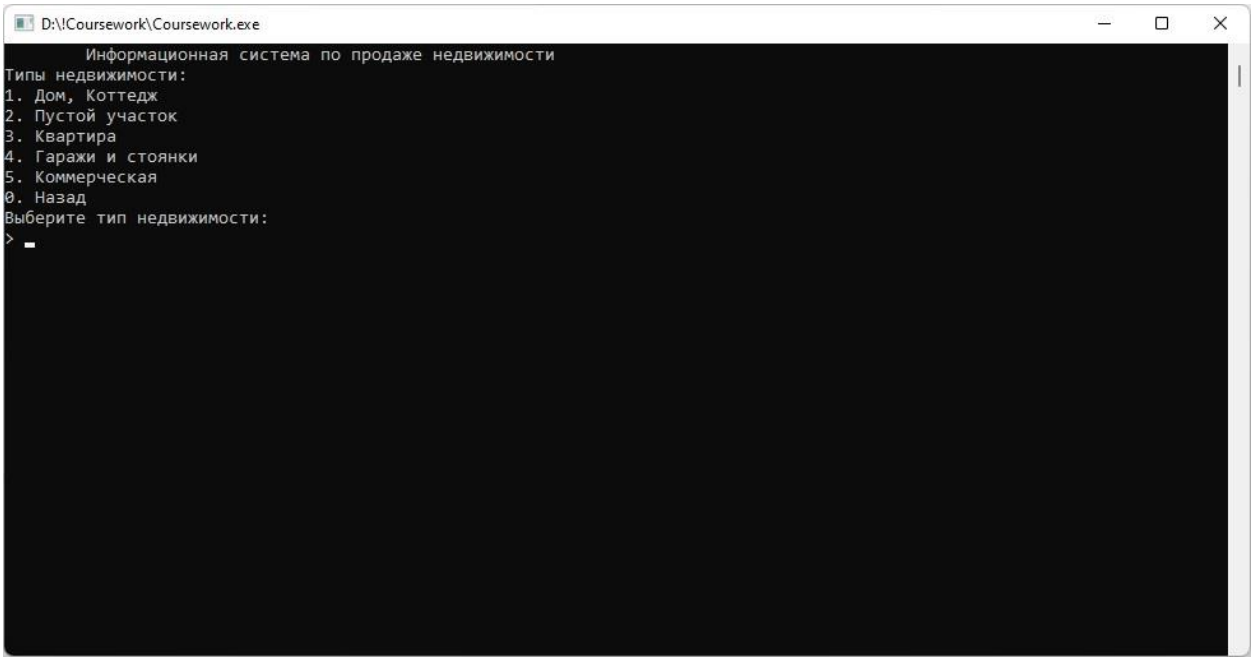

Рисунок 5.5 – Меню добавления объекта

В меню добавления объекта (рисунок 5.5) нам предоставляется выбор. Из меню добавления объекта мы можем:

– при вводе значения от 1 до 5, мы можем добавить недвижимость различных типов. Например введём «5» для добавления недвижимости коммерческого типа (см. п. 5.2.6);

– вернуться в главное меню для этого необходимо ввести «0» (см. п. 5.2.1).

5.2.6 Меню добавления коммерческой недвижимости

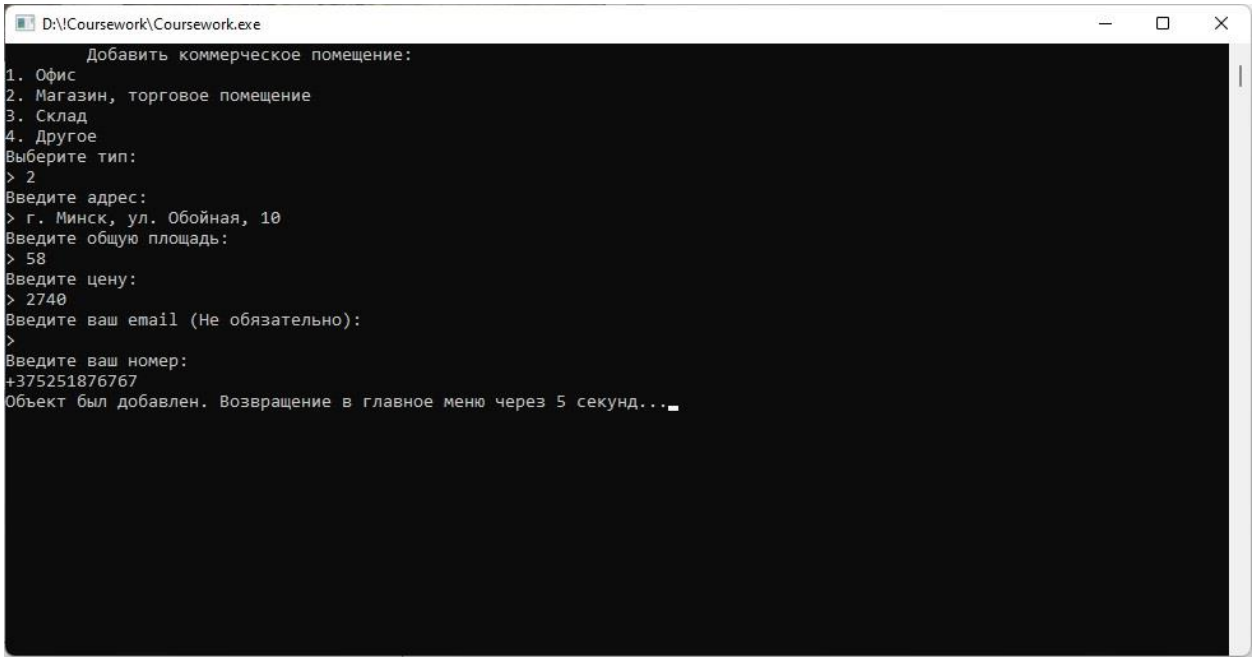

Рисунок 5.6 – Меню добавления коммерческой недвижимости

В меню добавления коммерческой недвижимости (рисунок 5.6) нам необходимо будет пройти анкету. После прохождения анкеты, объект будет создан и сохранён локально. Когда анкета будет заполнена и объект создан, через пару секунд мы возвращаемся в главное меню (см. п. 5.2.1).

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», которая была выбрана темой для курсового проекта.

Хотя программа и имеет интуитивно понятный интерфейс и является максимально простым в использовании, всё же к ней было создано пользовательское руководство.

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.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Страуструп, Б. Язык программирования C++ / Б. Страуструп; специальное издание. Пер с англ. - СПб.: BHV, 2008

[2] Шилдт, Г. C++: базовый курс / Г. Шилдт - М.: Вильямс; специальное издание. Пер. с англ. - М.: Вильямс, 2007

[3] Луцик Ю. А. Объектно-ориентированное программирование на языке C++ / 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