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

#include "Immovable.h"

Immovable::ContactDetails::ContactDetails(const string& phonePrimary, const string& email) {

setMobile(phonePrimary);

setEmail(email);

}

void Immovable::ContactDetails::setMobile(const string& phone) {

phoneNumber = phone;

}

string Immovable::ContactDetails::getMobile() {

return phoneNumber;

}

void Immovable::ContactDetails::setEmail(const string& email) {

this->email = email;

}

string Immovable::ContactDetails::getEmail() {

return email;

}

string Immovable::ContactDetails::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;

}

Immovable::Immovable(unsigned int id, const string& phone, const string& email, float cost,

double square, const string& address, bool actuality) {

this->id = id;

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

setCost(cost);

setSquare(square);

setAddress(address);

setActuality(actuality);

}

string Immovable::boolToString(bool positive) {

if (positive)

return "Да";

return "Нет";

}

// contact

string Immovable::getContact() {

return contact->getContactDetails();

}

void Immovable::setMobile(const string& newMobile) {

contact->setMobile(newMobile);

}

string Immovable::getMobile() {

return contact->getMobile();

}

void Immovable::setEmail(const string& newEmail) {

contact->setEmail(newEmail);

}

string Immovable::getEmail() {

return contact->getEmail();

}

// address

void Immovable::setAddress(const string& value) {

address = value;

}

string Immovable::getAddress() {

return address;

}

// square

double Immovable::setSquare(double value) {

return square = value;

}

string Immovable::printSquare() {

stringstream stream;

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

return stream.str() + " м^2";

}

double Immovable::getSquare() {

return square;

}

// cost

void Immovable::setCost(float value) {

cost = value;

}

string Immovable::printCost() {

stringstream stream;

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

return stream.str() + "$";

}

float Immovable::getCost() {

return cost;

}

// id

int Immovable::getId() {

return id;

}

// actuality

void Immovable::setActuality(bool isActual) {

this->isActual = isActual;

}

bool Immovable::getActuality() {

return isActual;

}

Immovable.cpp

#include "Immovable.h"

Immovable::ContactDetails::ContactDetails(const string& phonePrimary, const string& email) {

setMobile(phonePrimary);

setEmail(email);

}

void Immovable::ContactDetails::setMobile(const string& phone) {

phoneNumber = phone;

}

string Immovable::ContactDetails::getMobile() {

return phoneNumber;

}

void Immovable::ContactDetails::setEmail(const string& email) {

this->email = email;

}

string Immovable::ContactDetails::getEmail() {

return email;

}

string Immovable::ContactDetails::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;

}

Immovable::Immovable(unsigned int id, const string& phone, const string& email, float cost,

double square, const string& address, bool actuality) {

this->id = id;

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

setCost(cost);

setSquare(square);

setAddress(address);

setActuality(actuality);

}

string Immovable::boolToString(bool positive) {

if (positive)

return "Да";

return "Нет";

}

string Immovable::getContact() { // contact

return contact->getContactDetails();

}

void Immovable::setMobile(const string& newMobile) {

contact->setMobile(newMobile);

}

string Immovable::getMobile() {

return contact->getMobile();

}

void Immovable::setEmail(const string& newEmail) {

contact->setEmail(newEmail);

}

string Immovable::getEmail() {

return contact->getEmail();

}

void Immovable::setAddress(const string& value) { // address

address = value;

}

string Immovable::getAddress() {

return address;

}

double Immovable::setSquare(double value) { // square

return square = value;

}

string Immovable::printSquare() {

stringstream stream;

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

return stream.str() + " м^2";

}

double Immovable::getSquare() {

return square;

}

void Immovable::setCost(float value) { // cost

cost = value;

}

string Immovable::printCost() {

stringstream stream;

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

return stream.str() + "$";

}

float Immovable::getCost() {

return cost;

}

int Immovable::getId() { // id

return id;

}

void Immovable::setActuality(bool isActual) { // actuality

this->isActual = isActual;

}

bool Immovable::getActuality() {

return isActual;

}

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);

~Flat();

void printInfo() override;

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

void setFloor(int value);

int getFloor();

void setRooms(int value);

int getRooms();

void isHaveBalcony(bool have);

bool isHaveBalcony();

};

#endif //COURSEWORK\_FLAT\_H

Flat.cpp

#include "Flat.h"

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

const string& address, bool actuality, int rooms, int floor, bool haveBalcony)

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

setRooms(rooms);

setFloor(floor);

isHaveBalcony(haveBalcony);

}

Flat::~Flat() {}

void Flat::printInfo() {

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;

}

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 Flat::setFloor(int value) {

floor = value;

}

int Flat::getFloor() {

return floor;

}

void Flat::setRooms(int value) {

rooms = value;

}

int Flat::getRooms() {

return rooms;

}

void Flat::isHaveBalcony(bool have) {

haveBalcony = have;

}

bool Flat::isHaveBalcony() {

return haveBalcony;

}

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);

~Parking();

void printInfo() override;

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

void setType(int type);

string printType();

int getType();

};

#endif //COURSEWORK\_PARKING\_H

Parking.cpp

#include "Parking.h"

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

const string& address, bool actuality, int type)

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

setType(type);

}

Parking::~Parking() {}

void Parking::printInfo() {

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;

}

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 Parking::setType(int type) {

this->type = type;

}

string Parking::printType() {

switch (type) {

case 1:

return "Машино место";

case 2:

return "Бокс";

case 3:

return "Гараж";

default:

return "Другое";

}

}

int Parking::getType() {

return type;

}

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);

~Empty();

void printInfo() override;

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

void setConstruction(bool suitable);

bool suitableConstruction();

void setFarming(bool suitable);

bool suitableFarming();

};

#endif //COURSEWORK\_EMPTY\_H

Empty.cpp

#include "Empty.h"

Empty::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, bool suitableForFarming) :

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

setConstruction(suitableForConstruction);

setFarming(suitableForFarming);

}

Empty::~Empty() {}

void Empty::printInfo() {

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;

}

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 Empty::setConstruction(bool suitable) {

suitableForConstruction = suitable;

}

bool Empty::suitableConstruction() {

return suitableForConstruction;

}

void Empty::setFarming(bool suitable) {

suitableForFarming = suitable;

}

bool Empty::suitableFarming() {

return suitableForFarming;

}

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);

~House();

void printInfo() override;

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

void setFloors(int value);

int getFloors();

void setRooms(int value);

int getRooms();

void setParking(int value);

int getParkingSpace();

};

#endif //COURSEWORK\_HOUSE\_H

House.cpp

#include "House.h"

House::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, int rooms, int parkingSpaces)

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

setFloors(floors);

setRooms(rooms);

setParking(parkingSpaces);

}

House::~House() {}

void House::printInfo() {

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;

}

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 House::setFloors(int value) {

floors = value;

}

int House::getFloors() {

return floors;

}

void House::setRooms(int value) {

rooms = value;

}

int House::getRooms() {

return rooms;

}

void House::setParking(int value) {

parkingSpaces = value;

}

int House::getParkingSpace() {

return parkingSpaces;

}

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);

~Piece();

void setPond(bool available);

bool pond();

void setPlants(bool available);

bool plants();

void setCommunications(bool available);

bool communications();

};

#endif //COURSEWORK\_PIECE\_H

Piece.cpp

#include "Piece.h"

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

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

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

setPond(pond);

setPlants(plants);

setCommunications(communications);

}

~Piece() {}

void Piece::setPond(bool available) {

availablePond = available;

}

bool Piece::pond() {

return availablePond;

}

void Piece::setPlants(bool available) {

availablePlants = available;

}

bool Piece::plants() {

return availablePlants;

}

void Piece::setCommunications(bool available) {

availabilityOfCommunications = available;

}

bool Piece::communications() {

return availabilityOfCommunications;

}

Interfaces.h

#ifndef COURSEWORK\_INTERFACES\_H

#define COURSEWORK\_INTERFACES\_H

#include "Exception.h"

#include <iostream>

#include "Exception.h"

#include "Storage.h"

#include "ObjectManager.h"

#include "Input.h"

#include "House.h"

#include "Empty.h"

#include "Flat.h"

#include "Parking.h"

#include "Commercial.h"

#include "ObjectManager.h"

using namespace std;

class Interfaces {

Storage storage;

Input in;

ObjectManager objectManager;

unsigned int interfaceCode = 0;

unsigned int resentId = 0;

string selectedType;

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

template<typename T>

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

public:

void run();

template<typename T>

void removeObject(T& object);

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

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 <iomanip>

#include <iostream>

#include "Exception.h"

#include <sstream>

#include <codecvt>

using namespace std;

class Input {

const string pointer = "\n\033[38;2;128;0;128m> \033[0m"; // ANSI escape code для цвета

template<typename T>

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

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);

template<typename T>

string toStringWithPrecision(T value, int precision);

template<typename T>

string toString(T value) {

std::stringstream stream;

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

return stream.str();

}

string cp1251\_to\_utf8(const std::string& cp1251Str);

};

#endif //COURSEWORK\_INPUT\_H

Input.cpp

#include "Input.h"

#include "Exception.h"

#include <iomanip>

#include <string>

#include <iostream>

#include <sstream>

#include <codecvt>

#include <algorithm>

template<typename T>

std::string Input::toStringWithPrecision(T value, int precision) {

std::stringstream stream;

stream << std::fixed << std::setprecision(precision) << 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);

//string svalue = toStringWithPrecision(value, 2);

//value = std::stof(svalue);

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);

//string svalue = toStringWithPrecision(value, 2);

//value = stof(svalue);

break;

}

catch (Exception ex) {

ex.what();

}

}

return value;

}

string Input::inputMobile() {

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

unsigned long minValue = 100000000;

unsigned long maxValue = 999999999;

unsigned long value;

while (true) {

cout << msg;

try {

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

break;

}

catch (Exception ex) {

ex.what();

}

}

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

}

string Input::inputEmail() {

string email = "";

cout << "Желаете ввести email?";

if (inputBool("")) {

try {

cout << "Введите ваш email:";

email = inputString("");

}

catch (Exception ex) {

ex.what();

}

}

return email;

}

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

string value;

cout << question;

cout << pointer;

cin.clear();

//fflush(stdin);

getline(cin, value);

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" || answer == "д" || answer == "Д") {

return true;

}

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

answer == "N" || answer == "No" || answer == "no" || answer == "н" || answer == "Н") {

return false;

}

else {

cout << "Для ответа используйте: да, нет." << endl;

}

}

}

string Input::cp1251\_to\_utf8(const std::string& cp1251Str) {

std::wstring\_convert<std::codecvt\_utf8<wchar\_t>> converter;

std::wstring utf16 = converter.from\_bytes(cp1251Str);

std::wstring\_convert<std::codecvt\_utf8\_utf16<wchar\_t>> utf16Converter;

std::string utf8 = utf16Converter.to\_bytes(utf16);

return utf8;

}

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 <fstream>

#include "Exception.h"

#include "Input.h"

#include "ObjectManager.h"

#include "House.h"

#include "Empty.h"

#include "Flat.h"

#include "Parking.h"

#include "Commercial.h"

class Storage {

string filepath = "app.txt";

unsigned int freeId = 0;

bool stob(const string& str); // string to bool

void loadAll();// load all exists objects

template<typename T>

void addClass(T className);

public:

Storage();

vector<House> vectorHouses;

vector<Empty> vectorEmpty;

vector<Flat> vectorFlat;

vector<Parking> vectorParking;

vector<Commercial> vectorCommercial;

void saveAppFile();

void uploadAppFile();

template<class T>

void upload(T className); // from vector to file

void load(unsigned int id); // from file to vector

void addHouse(); // create & add to file & to vector

void addEmpty();

void addFlat();

void addParking();

void addCommercial();

unsigned int getFreeId();

unsigned int requestId();

unsigned int decId();

void delFile(unsigned int resentId);

string identifyObject(unsigned int id); // from ID to classType

};

#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::delFile(unsigned int resentId) {

string filename = std::to\_string(resentId) + ".txt";

remove(filename.c\_str());

ofstream out(filename);

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

throw Exception("Не удалось обновить файл", 2);

}

out.close();

}

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);

vectorHouses.push\_back(house);

}

else if (classType == "empty") {

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

vectorEmpty.push\_back(empty);

}

else if (classType == "flat") {

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

vectorFlat.push\_back(flat);

}

else if (classType == "parking") {

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

vectorParking.push\_back(parking);

}

else if (classType == "commercial") {

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

vectorCommercial.push\_back(commercial);

}

in.close();

}

void Storage::loadAll() {

for (size\_t 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(const string& str) {

return str != "0";

}

Commercial.h

#ifndef COURSEWORK\_COMMERCIAL\_H

#define COURSEWORK\_COMMERCIAL\_H

#include "Immovable.h"

#include "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);

~Commercial();

void printInfo() override;

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

void setType(int type);

string printType();

int getType();

};

#endif //COURSEWORK\_COMMERCIAL\_H

Commercial.cpp

#include "Commercial.h"

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

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

address, actuality) {

setType(type);

}

Commercial::~Commercial() {}

void Commercial::printInfo() {

cout << " Информация о помещении:" << 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;

}

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 Commercial::setType(int type) {

this->type = type;

}

string Commercial::printType() {

switch (type) {

case 1:

return "Офис";

case 2:

return "Магазин, торговое помещение";

case 3:

return "Склад";

default:

return "Другое";

}

}

int Commercial::getType() {

return type;

}