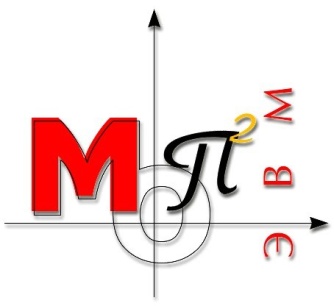
Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждения высшего образования   
«Южный федеральный университет»

Инженерно-технологическая академия

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**

Лабораторная работа №4

по дисциплине

"ООП"

на тему

"Контейнеры STL"

**Выполнил:**

студент группы КТбо2-6

Пустовой А.В

**Проверил:**

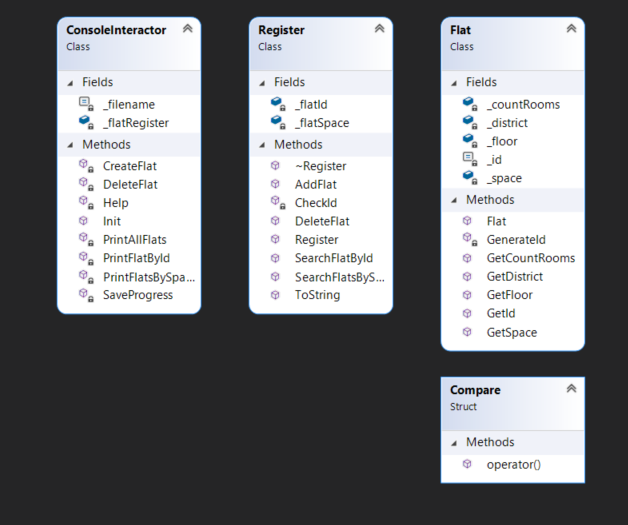
Таганрог, 2020

**Вариант №13**

**Задание:**

Класс «Квартира(Flat)» с полями, описывающими квартиру, предназначенную к продаже: площадь, количество комнат, этаж, район. Вывести сведения о квартирах по заявке на покупку, в которой указана только площадь. В выводимых сведениях площадь квартиры может отличаться от заявленной, но не более, чем на 10%.

**Диаграмма классов**



**Листинг программы**

**FLAT.H**

#pragma once

#include <string>

#include <iostream>

using std::string;

class Flat

{

public:

Flat(int space, int countRooms, int floor, string district) : \

\_space(space), \_countRooms(countRooms), \_floor(floor), \_district(district) {}

int GetSpace() const;

int GetCountRooms() const;

int GetFloor() const;

int GetId() const;

const string& GetDistrict() const;

friend std::ostream& operator<<(std::ostream& out, const Flat& flat);

friend bool operator<(const Flat& a, const Flat& b);

private:

const int \_id = GenerateId();

int \_space;

int \_countRooms;

int \_floor;

string \_district;

static int GenerateId();

};

**FLAT.CPP**

#include "Flat.h"

int Flat::GenerateId()

{

static int id = 0;

return ++id;

}

int Flat::GetSpace() const

{

return \_space;

}

int Flat::GetCountRooms() const

{

return \_countRooms;

}

int Flat::GetFloor() const

{

return \_floor;

}

const string& Flat::GetDistrict() const

{

return \_district;

}

std::ostream& operator<<(std::ostream& out, const Flat& flat)

{

out << "Id: " << flat.\_id << " District: " << flat.\_district << " Floor: " << \

flat.\_floor << " Count rooms: " << flat.\_countRooms << " Space: " << flat.\_space;

return out;

}

bool operator<(const Flat& a, const Flat& b)

{

return a.GetSpace() < b.GetSpace();

}

int Flat::GetId() const

{

return \_id;

}

**REGISTER.H**

#pragma once

#include <set>

#include <map>

#include <vector>

#include <algorithm>

#include <stdexcept>

#include "Flat.h"

struct Compare

{

bool operator()(const Flat\* a,const Flat\* b) const

{

return a->GetSpace() < b->GetSpace();

}

};

class Register

{

public:

Register() = default;

~Register();

void AddFlat(int space, int countRooms, int floor, const string& district);

void DeleteFlat(int id);

const Flat& SearchFlatById(int id) const;

const std::vector<Flat\*> SearchFlatsBySpace(int space) const;

string ToString(int code) const;

private:

std::map<int, Flat\*> \_flatId;

std::set<Flat\*, Compare> \_flatSpace;

void CheckId(int id) const;

};

**REGISTER.CPP**

#include "Register.h"

Register::~Register()

{

if (\_flatId.size())

{

for (auto it : \_flatId)

{

delete it.second;

}

}

\_flatId.clear();

\_flatSpace.clear();

}

void Register::AddFlat(int space, int countRooms, int floor, const string& district)

{

Flat\* newFlat = new Flat(space, countRooms, floor, district);

\_flatId[newFlat->GetId()] = newFlat;

\_flatSpace.insert(newFlat);

}

void Register::DeleteFlat(int id)

{

auto itId = \_flatId.find(id);

Flat\* flat = itId->second;

CheckId(id);

\_flatSpace.erase(flat);

\_flatId.erase(itId);

}

const Flat& Register::SearchFlatById(int id) const

{

CheckId(id);

return \*(\_flatId.find(id)->second);

}

const std::vector<Flat\*> Register::SearchFlatsBySpace(int space) const

{

std::vector<Flat\*> res;

Flat min(space - space / 10, 0, 0, ""), max(space + space / 10, 0, 0, "");

for (auto it = \_flatSpace.lower\_bound(&min); it != \_flatSpace.upper\_bound(&max); ++it)

{

res.push\_back(\*it);

}

return res;

}

string Register::ToString(int code) const

{

string out = "";

if (code)

{

for (auto it : \_flatSpace)

{

out +=it->GetDistrict() + " " + \

std::to\_string(it->GetFloor()) + " " + std::to\_string(it->GetCountRooms()) + \

" " + std::to\_string(it->GetSpace()) + "\n";

}

}

else

{

for (auto it : \_flatSpace)

{

out += "Id: " + std::to\_string(it->GetId()) + " | District: " + it->GetDistrict() + " | Floor: " + \

std::to\_string(it->GetFloor()) + " | Count rooms: " + std::to\_string(it->GetCountRooms()) + \

" | Space: " + std::to\_string(it->GetSpace()) + "\n";

}

}

return out;

}

void Register::CheckId(int id) const

{

auto itId = \_flatId.find(id);

if (itId == \_flatId.end())

{

throw std::logic\_error("WRONG\_ID");

}

}

**CONSOLEINTERACTOR.H**

#pragma once

#include <iostream>

#include <fstream>

#include "Register.h"

using std::cin;

using std::cout;

using std::ifstream;

using std::ofstream;

class ConsoleInteractor

{

public:

void Init();

private:

void Help() const;

void CreateFlat();

void SaveProgress() const;

void DeleteFlat();

void PrintAllFlats() const;

void PrintFlatById() const;

void PrintFlatsBySpace() const;

Register \_flatRegister;

const string \_filename = "register.txt";

};

**CONSOLEINTERACTOR.CPP**

#include "ConsoleInteractor.h"

void ConsoleInteractor::Help() const

{

cout << "\"1\" - Create new Flat\n\"2\" - Delete Flat\n\"3\" - Help\n\"4\" - Print all flats\n\"5\" - Find flat by id\n\

\"6\" - Find flats by space\n\"7\" - Save register\n\"8\" - Exit\n";

}

void ConsoleInteractor::CreateFlat()

{

int space, countRooms, floor;

string district;

cout << "Enter space of new flat:\n";

cout << ">> ";

cin >> space;

cout << "Enter count rooms of new flat:\n";

cout << ">> ";

cin >> countRooms;

cout << "Enter floor of new flat:\n";

cout << ">> ";

cin >> floor;

cout << "Enter district of new flat:\n";

cout << ">> ";

cin >> district;

\_flatRegister.AddFlat(space, countRooms, floor, district);

}

void ConsoleInteractor::DeleteFlat()

{

int id;

cout << "Enter id:\n";

cout << ">> ";

cin >> id;

try

{

\_flatRegister.DeleteFlat(id);

}

catch (const std::logic\_error& err)

{

cout << err.what() << "\n";

}

}

void ConsoleInteractor::PrintAllFlats() const

{

cout << \_flatRegister.ToString(0);

}

void ConsoleInteractor::SaveProgress() const

{

ofstream out(\_filename);

out << \_flatRegister.ToString(1);

out.close();

}

void ConsoleInteractor::PrintFlatById() const

{

int id;

cout << "Enter id:\n";

cout << ">> ";

cin >> id;

try

{

cout << \_flatRegister.SearchFlatById(id) << "\n";

}

catch (const std::logic\_error& err)

{

cout << err.what() << "\n";

}

}

void ConsoleInteractor::PrintFlatsBySpace() const

{

int space;

cout << "Enter space to print you suitable houses:\n";

cout << ">> ";

cin >> space;

std::vector<Flat\*> findResult = \_flatRegister.SearchFlatsBySpace(space);

for (auto it : findResult)

{

cout << \*it << "\n";

}

}

void ConsoleInteractor::Init()

{

int space, countRooms, floor;

string district;

ifstream in(\_filename);

if (!in)

{

cout << "register.txt is not exist or empty\n";

}

else

{

while (in)

{

in >> district >> floor >> countRooms >> space;

\_flatRegister.AddFlat(space, countRooms, floor, district);

}

}

in.close();

bool flag = true;

int code;

while (flag)

{

cout << "Enter \"3\" - Help\n";

cout << ">> ";

cin >> code;

switch (code)

{

case 1:

CreateFlat();

break;

case 2:

DeleteFlat();

break;

case 3:

Help();

break;

case 4:

PrintAllFlats();

break;

case 5:

PrintFlatById();

break;

case 6:

PrintFlatsBySpace();

break;

case 7:

SaveProgress();

break;

case 8:

flag = false;

break;

}

}

}

**Вывод**

Я ознакомился с классами-контейнерами библиотеки STL.