МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 4**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Контейнеры STL»**

*Вариант № 14*

Выполнил:

Студент группы

Ктбо2-8

Жалнин Д. И.

Проверил:

Тарасов С. А.

Оценка

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

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# Оглавление

[Оглавление 2](#_Toc59699348)

[1 Формулировка задания 3](#_Toc59699349)

[2 Спецификация классов 4](#_Toc59699350)

[2.1 Класс Elections 4](#_Toc59699351)

[2.2 Класс ElectionsRepo 5](#_Toc59699352)

[2.3 Класс Interactor 5](#_Toc59699353)

[3 Диаграмма классов 6](#_Toc59699354)

[4 Листинг программы 7](#_Toc59699355)

[4.1 lab4.cpp 7](#_Toc59699356)

[4.2 Interactor.h 7](#_Toc59699357)

[4.3 Interactor.cpp 7](#_Toc59699358)

[4.4 Elections.h 9](#_Toc59699359)

[4.5 Elections.cpp 10](#_Toc59699360)

[4.6 ElectionsRepo.h 11](#_Toc59699361)

[4.7 ElectionsRepo.cpp 12](#_Toc59699362)

# 1 Формулировка задания

В лабораторной работе должна быть создана программа, содержащая класс «Выборы(Elections)» с полями: ФИО кандидата, дата рождения, место работы, рейтинг предварительных опросов. Необходимо вывести сведения о кандидатах, рейтинг которых превышает вводимое число. Объекты класса должны храниться в одном из контейнеров STL.

При выполнении работы необходимо:

− разработать соответствующие классы, конструкторы, поля и методы

− поля класса сделать закрытыми; для чтения и изменения их значений определить открытые методы;

– предусмотреть во всех вариантах консольный ввод данных для создания объектов и консольный вывод результатов.

# 2 Спецификация классов

## 2.1 Класс Elections

Приватные поля:

**int \_id;**

**std::string fullName;**

**std::string birthDate;**

**std::string work;**

**double rating;**

/\* Поля, содержащие id, ФИО, дату рождения, место работы и рейтинг человека \*/

Публичные поля:

**Elections(int id, std::string fullname, std::string birthDate, std::string work, const std::string &rating);**

/\* Конструктор, принимающий необходимую для создания объекта информацию, которая была считана с текстового файла \*/

**std::string toString();**

/\* Метод, возвращающий информацию, хранимую в объекте в виде строки (если быть точным в JSON формате) \*/

**bool operator<(const Elections &e);**

/\* Переопределенный оператор «меньше», необходимый для правильной сортировки объектов данного класса\*/

**std::string GetFullName();**

**std::string GetWork();**

**std::string GetBirthDate();**

**double GetRating();**

**int GetID()**

// Методы, возвращающие значения полей объекта

## 2.2 Класс ElectionsRepo

Класс, реализующий чтение информации из файла, а также из консоли, ее обработку и хранение. Имеет методы:

**Read** – для чтения информации из файла.

**Write** – для записи информации в файл.

**Save** – для сохранения объекта в локальном хранилище.

**Remove** – для удаления объекта из локального хранилища.

## 2.3 Класс Interactor

Класс-обертка, реализующий взаимодействие пользователя с основной логикой программы.

**#pragma once**

**#include "ElectionsRepo.h"**

**#include <iostream>**

**#include "vector"**

**#include "list"**

**#include <algorithm>**

**#include <map>**

**using namespace std;**

**class Interactor {**

**private:**

**ElectionsRepo\* \_fileSystem;**

**bool run = true;**

**void PrintInfo(bool lgToSmall, double limiter);**

**public:**

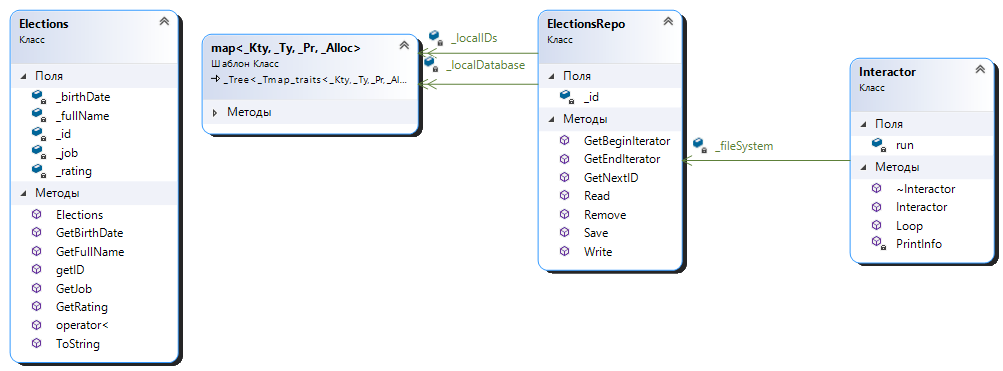
**void Loop();**

**explicit Interactor(ElectionsRepo \*fileSystem);**

**~Interactor();**

**};**

# 3 Диаграмма классов



*Рисунок 1 – UML диаграмма классов.*

# 4 Листинг программы

## 4.1 lab4.cpp

#include "Interactor.h"

#include <vld.h>

#include <locale.h>

int main() {

setlocale(LC\_ALL, "Ru");

auto\* interactor = new Interactor();

interactor->loop();

delete interactor;

return 0;

}

## 4.2 Interactor.h

#pragma once

#include "FileReader.h"

#include <iostream>

#include "vector"

#include <algorithm>

using namespace std;

class Interactor {

private:

bool run = true;

public:

void loop();

};

## 4.3 Interactor.cpp

#include "Interactor.h"

#include <map>

void Interactor::Loop() {

while (run) {

int t;

cout << "\nLoad from file - \"1\""

"\nPrint - \"2\""

"\nUpload to file - \"3\""

"\nAdd element - \"4\""

"\nRemove element - \"5\""

"\nExit - \"0\"\n";

cin >> t;

switch (t) {

case (0): {

run = false;

break;

}

case (1): {

std::string path;

cout << "Path to file:\n";

cin >> path;

\_fileSystem->Read(path);

break;

}

case (2): {

double limiter;

cout << "1 - From large rating\n2 - From small rating\n";

int mode;

cin >> mode;

if (mode != 1 && mode != 2) {

continue;

}

cout << "Enter the limiter (-1 - without lim)\n";

cin >> limiter;

if (limiter == -1 && mode == 1) {

PrintInfo(true, -1);

} else if (limiter == -1 && mode == 2) {

PrintInfo(false, 100000000);

} else if (mode == 1) {

PrintInfo(true, limiter);

} else {

PrintInfo(false, limiter);

}

break;

}

case (3): {

std::string path;

cout << "Path to file:\n";

cin >> path;

\_fileSystem->Write(path);

break;

}

case (4): {

std::string fullname;

std::string birthDate;

std::string job;

std::string rating;

cout << "Fullname:\n";

std::getline(std::cin, fullname);

if (fullname.empty())

std::getline(std::cin, fullname);

cout << "Date of birth:\n";

std::getline(std::cin, birthDate);

cout << "Job:\n";

std::getline(std::cin, job);

cout << "Rating:\n";

std::getline(std::cin, rating);

Elections e(\_fileSystem->GetNextID(),fullname, birthDate, job, rating);

\_fileSystem->Save(e);

break;

}

case(5): {

int id;

cout<<"ID:\n";

std::cin>>id;

\_fileSystem->Remove(id);

break;

}

default:

break;

}

}

}

Interactor::Interactor(ElectionsRepo \*fileSystem) {

this->\_fileSystem = fileSystem;

}

void Interactor::PrintInfo(bool lgToSmall, double limiter) {

bool singleElementFlag = false;

map<double, set<Elections>>::iterator i;

map<double, set<Elections>>::iterator end;

if (lgToSmall) {

i = \_fileSystem->GetEndIterator();

end = \_fileSystem->GetBeginIterator();

if (i != end) {

i--;

end--;

if (!i->second.empty() && i == end) {

singleElementFlag = true;

}

}

} else {

i = \_fileSystem->GetBeginIterator();

end = \_fileSystem->GetEndIterator();

}

cout << "\*\*\*BEGIN\*\*\*\n";

while (i != end && ((i->first < limiter && !lgToSmall) || (i->first > limiter && lgToSmall))) {

for (auto s : i->second) {

cout << s.ToString() << "\n";

}

if (lgToSmall) {

i--;

} else {

i++;

}

}

if (singleElementFlag) {

for (auto s : i->second) {

cout << s.ToString() << "\n";

}

}

cout << "\*\*\*END\*\*\*\n";

}

Interactor::~Interactor() {

delete \_fileSystem;

}

## 4.4 Elections.h

#pragma once

#include <string>

class Elections {

private:

int \_id;

std::string \_fullName;

std::string \_birthDate;

std::string \_job;

double \_rating;

public:

Elections(int id, std::string fullname, std::string birthDate, std::string work, const std::string &rating);

std::string ToString() const;

bool operator<(const Elections &e) const;

std::string GetFullName() const;

std::string GetJob() const;

std::string GetBirthDate() const;

double GetRating() const;

int getID() const;

};

## 4.5 Elections.cpp

#include "Elections.h"

#include "ElectionsRepo.h"

Elections::Elections(int id,std::string fullname, std::string birthDate, std::string work, const std::string &rating) {

this->\_fullName = std::move(fullname);

this->\_birthDate = std::move(birthDate);

this->\_job = std::move(work);

this->\_rating = std::stod(rating);

\_id = id;

}

double Elections::GetRating() const {

return \_rating;

}

std::string Elections::GetBirthDate() const{

return \_birthDate;

}

std::string Elections::GetJob() const{

return \_job;

}

std::string Elections::GetFullName() const{

return \_fullName;

}

bool Elections::operator<(const Elections &e) const{

if (\_rating < e.\_rating) {

return true;

} else {

if (\_rating == e.\_rating && \_fullName < e.\_fullName) {

return true;

} else {

if (\_rating == e.\_rating && \_fullName == e.\_fullName && \_job < e.\_job) {

return true;

} else {

if (\_rating == e.\_rating && \_fullName == e.\_fullName && \_job == e.\_job && \_birthDate < e.\_birthDate) {

return true;

}

}

}

}

return false;

}

std::string Elections::ToString() const{

return "Elections { "

"id: '"+ std::to\_string(\_id)+

"', fullName: '" + \_fullName +

"', birthDate: '" + \_birthDate +

"', job: '" + \_job +

"', rating: '" + std::to\_string(\_rating) +

"' }";

}

int Elections::getID() const {

return \_id;

}

## 4.6 ElectionsRepo.h

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include "Elections.h"

#include <set>

#include <map>

class ElectionsRepo {

private:

int \_id;

std::map<double, std::set<Elections>> \_localDatabase;

std::map<const int, Elections> \_localIDs;

public:

void Read(const std::string &path);

void Write(const std::string &path);

void Save(const Elections &e);

void Remove(int id);

std::map<double, std::set<Elections>>::iterator GetBeginIterator();

std::map<double, std::set<Elections>>::iterator GetEndIterator();

int GetNextID();

};

## 4.7 ElectionsRepo.cpp

#include <list>

#include <map>

#include "ElectionsRepo.h"

#include <set>

void ElectionsRepo::Read(const std::string &path) {

std::ifstream in(path);

std::string line;

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

std::cout<<"Can not open the file!\n";

} else {

std::string name1;

std::string name2;

std::string name3;

std::string date;

std::string work;

std::string rating;

while (getline(in, line)) {

int nameEnd1 = line.find(' ', 0);

name1 = line.substr(0, nameEnd1);

int nameEnd2 = line.find(' ', nameEnd1 + 1);

name2 = line.substr(nameEnd1 + 1, nameEnd2 - nameEnd1 - 1);

int nameEnd = line.find(' ', nameEnd2 + 1);

name3 = line.substr(nameEnd2 + 1, nameEnd - nameEnd2 - 1);

int dateEnd = line.find(" \"", nameEnd + 1);

date = line.substr(nameEnd + 1, dateEnd - nameEnd - 1);

int workEnd = line.find("\" ", dateEnd + 1);

work = line.substr(dateEnd + 2, workEnd - dateEnd - 2);

int ratingEnd = line.length() - 1;

rating = line.substr(workEnd + 2, ratingEnd - workEnd - 1);

Elections e(GetNextID(),name1.append(" ").append(name2).append(" ").append(name3), date, work, rating);

Save(e);

}

}

in.close();

}

void ElectionsRepo::Save(const Elections &e) {

if (!\_localDatabase.contains(e.GetRating())) {

std::set<Elections> s;

s.insert(e);

\_localDatabase.insert(std::pair<double, std::set<Elections>>(e.GetRating(), s));

} else {

\_localDatabase[e.GetRating()].insert(e);

}

\_localIDs.insert(std::pair(e.getID(), e));

}

void ElectionsRepo::Write(const std::string &path) {

std::ofstream out(path);

if (out.is\_open())

for (const auto &set : \_localDatabase) {

for (auto e : set.second) {

out << e.GetFullName() << " "

<< e.GetBirthDate() << " \""

<< e.GetJob() << "\" "

<< e.GetRating() << "\n";

}

}

else {

std::cout<<"Can not open the file!\n";

}

out.close();

}

int ElectionsRepo::GetNextID() {

return \_id++;

}

void ElectionsRepo::Remove(int id) {

int size = \_localDatabase.size();

for (auto i = \_localDatabase.begin(); i != \_localDatabase.end(); i++){

auto j = i->second.begin();

auto end = i->second.end();

for (; j != end; j++){

if (j->getID() == id){

i->second.erase(j);

if (i->second.empty()){

\_localDatabase.erase(i);

}

std::cout<<"Erased!\n";

return;

}

}

}

std::cout<<"Not found!\n";

}

std::map<double, std::set<Elections>>::iterator ElectionsRepo::GetBeginIterator() {

return \_localDatabase.begin();

}

std::map<double, std::set<Elections>>::iterator ElectionsRepo::GetEndIterator() {

return \_localDatabase.end();

}