МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**Лабораторная работа № 4**

по курсу «Объектно-ориентированное программирование»

**«Контейнеры STL»**

**14 ВАРИАНТ**

Выполнили:

студенты гр. КТбо2-7

Миронченко П.Д.

Проверил:

Тарасов С. А.

**Таганрог 2020**

1. **Вариант задания №14**

Класс «Выборы (Elections)» с полями: ФИО кандидата, дата рождения, место работы, рейтинг предварительных опросов; вывести сведения о кандидатах, рейтинг которых превышает вводимое число.

1. **Спецификации классов**

**Спецификация класса Interactor**

Класс Interactor представляет собой фасад. Содержит приватное поле указателя на интерфейс ICandidateContainer и методы для взаимодействия с пользователем и манипуляций с данными

**Спецификация класса Candidate**

Содержит в себе приватные поля ФИО, даты рождения, места работы и рейтинга, ID, а также геттеры к ним

**Спецификация класса Date**

Содержит в себе приватные поля дня, месяца и года предназначенные для хранения даты рождения в классе Electtions

**Спецификация класса ICandidateContainer**

Представляет собой интерфейс, требуемый для динамического выбора классов CandidateMap и CandidateVector. Содержит виртуальные методы для добавления элемента и поиска элементов по полям класса Candidate

**Спецификация класса CandidateMap**

Представляет собой реализацию интерфейса ICandidateContainer, содержит приватное поля типа map и реализацию методов интерфейса

**Спецификация класса ElectionsVector**

Представляет собой реализацию интерфейса ICandidateContainer, содержит приватное поле типа vector <Elections> и реализацию методов интерфейса

**Спецификация класса ContainerFactory**

Представляет собой фабрику, которая выдаёт объекты, реализующие интерфейс ICandidateContainer, по запросу пользователя

1. **Используемые математические зависимости и алгоритмы**

В данной программе используются алгоритмы линейного поиска и бинарного поиска, реализованные как методы контейнеров stl

1. **Диаграмма класса**

Diagram

Description automatically generated

1. **Листинг программы**

**Lab4.cpp**

#include <iostream>

#include <fstream>

#include "interactor.h"

using namespace std;

int main()

{

Interactor it;

it.Init("test.txt");

return 0;

}

**Interactor.h**

#ifndef INTERACTOR\_H

#define INTERACTOR\_H

#include <iostream>

#include <fstream>

#include <vector>

#include "ICandidateContainer.h"

#include "ContainerFactory.h"

class Interactor

{

public:

Interactor();

~Interactor();

void Init(const std::string& file\_to\_input);

private:

ICandidateContainer\* \_container;

Candidate \_ReadCandidate(std::istream& in, bool greaters) const;

void \_Help() const;

void \_FileParse(const std::string& file\_to\_input);

void \_Interaction(const std::string& file\_to\_input);

void \_ChooseContainer();

};

#endif

**Date.h**

#ifndef DATE\_H

#define DATE\_H

#include <iostream>

class Date

{

public:

Date(int dd, int mm, int yy);

Date();

~Date() = default;

int GetDay() const;

int GetMonth() const;

int GetYear() const;

int GetDaysCount() const;

static int GetDaysCountInMonth(int month, int year);

static int GetDaysCountInYear(int year);

bool operator==(const Date& other) const;

bool operator!=(const Date& other) const;

bool operator<(const Date& other) const;

bool operator>(const Date& other) const;

friend std::ostream& operator<<(std::ostream& out, const Date& object);

private:

int \_day;

int \_month;

int \_year;

};

#endif *// !DATE\_H*

**Candidate.h**

#ifndef CANDIDATE\_H

#define CANDIDATE\_H

#include <string>

#include "Date.h"

class Candidate

{

friend class CandidateMap;

friend class CandidateVector;

public:

Candidate(const std::string& fullName, const Date& dateOfBirth, const std::string& job, const double& rating);

Candidate() = default;

~Candidate() = default;

std::string GetFullName() const;

Date GetDateOfBirth() const;

std::string GetJob() const;

double GetRating() const;

int GetID() const;

friend std::ostream& operator<<(std::ostream& out, const Candidate& object);

private:

static int s\_idGenerator;

int \_objectID;

std::string \_fullName;

Date \_dateOfBirth;

std::string \_job;

double \_rating;

};

#endif *// !CANDIDATE\_H*

**ICandidateContainer.h**

#ifndef I\_CANDIDATE\_CONTAINER\_H

#define I\_CANDIDATE\_CONTAINER\_H

#include "Candidate.h"

#include <vector>

class ICandidateContainer

{

public:

ICandidateContainer() =default;

virtual ~ICandidateContainer() {};

virtual void AddItem(const Candidate& item) =0;

virtual std::vector<Candidate\*> SearchByRating(double rait, char mode) =0;

virtual std::vector<Candidate\*> SearchByName(const std::string& name) =0;

virtual std::vector<Candidate\*> SearchByJobPlace(const std::string& job) =0;

virtual std::vector<Candidate\*> SearchByBirthDate(Date date, char mode) =0;

virtual std::vector<Candidate\*> GetCandidates() =0;

virtual void UpdateRatingById(int id, double rating) =0;

virtual void UpdateNameById(int id, const std::string& name) =0;

virtual void UpdateJobPlaceById(int id, const std::string& job) =0;

virtual void UpdateBirthDatById(int id, Date date) =0;

virtual void RemoveById(int id) =0;

virtual Candidate\* FindById(int id) =0;

};

#endif *// !I\_CANDIDATE\_CONTAINER\_H*

**CandidateVector.h**

#ifndef CANDIDATE\_VECTOR\_H

#define CANDIDATE\_VECTOR\_H

#include <vector>

#include "Candidate.h"

#include "ICandidateContainer.h"

class CandidateVector : public ICandidateContainer

{

public:

CandidateVector() =default;

~CandidateVector() override =default;

void AddItem(const Candidate& item) override;

std::vector<Candidate\*> SearchByRating(double rait, char mode) override;

std::vector<Candidate\*> SearchByName(const std::string& name) override;

std::vector<Candidate\*> SearchByJobPlace(const std::string& job) override;

std::vector<Candidate\*> SearchByBirthDate(Date date, char mode) override;

std::vector<Candidate\*> GetCandidates() override;

void UpdateRatingById(int id, double rating) override;

void UpdateNameById(int id, const std::string& name) override;

void UpdateJobPlaceById(int id, const std::string& job) override;

void UpdateBirthDatById(int id, Date date) override;

void RemoveById(int id) override;

Candidate\* FindById(int id) override;

private:

std::vector<Candidate> \_candidates;

};

#endif *// !CANDIDATE\_VECTOR\_H*

**ElectionsMap.h**

#ifndef CANDIDATE\_MAP\_H

#define CANDIDATE\_MAP\_H

#include <map>

#include <set>

#include "Candidate.h"

#include "ICandidateContainer.h"

class CandidateMap : public ICandidateContainer

{

public:

CandidateMap() =default;

~CandidateMap() override =default;

void AddItem(const Candidate& item) override;

std::vector<Candidate\*> SearchByRating(double rait, char mode) override;

std::vector<Candidate\*> SearchByName(const std::string& name) override;

std::vector<Candidate\*> SearchByJobPlace(const std::string& job) override;

std::vector<Candidate\*> SearchByBirthDate(Date date, char mode) override;

std::vector<Candidate\*> GetCandidates() override;

void UpdateRatingById(int id, double rating) override;

void UpdateNameById(int id, const std::string& name) override;

void UpdateJobPlaceById(int id, const std::string& job) override;

void UpdateBirthDatById(int id, Date date) override;

void RemoveById(int id) override;

Candidate\* FindById(int id) override;

private:

std::map< int, Candidate > \_candidates;

std::map< double, std::set<int> > \_candidatesByRating;

std::map< std::string, std::set<int> > \_candidatesByName;

std::map< std::string, std::set<int> > \_candidatesByJob;

std::map< Date, std::set<int> > \_candidatesByBirthDate;

};

#endif *// !CANDIDATE\_MAP\_H*

**ContainerFactory.h**

#ifndef CONTAINER\_FACTORY\_H

#define CONTAINER\_FACTORY\_H

#include "ICandidateContainer.h"

#include "CandidateMap.h"

#include "CandidateVector.h"

class ContainerFactory

{

public:

ContainerFactory() = default;

~ContainerFactory() = default;

ICandidateContainer\* GetContainer(char type);

};

#endif *// !CONTAINER\_FACTORY\_H*

**Interactor.cpp**

#include "Interactor.h"

Interactor::Interactor(): \_container(nullptr)

{

}

Interactor::~Interactor()

{

if (\_container != nullptr)

{

delete \_container;

}

}

void Interactor::\_Help() const

{

std::cout <<"===========================================\n";

std::cout << "Commands: \n";

std::cout << "1) Add new item\n";

std::cout << "2) Search by name\n";

std::cout << "3) Search by rating\n";

std::cout << "4) Search by job place\n";

std::cout << "5) Search by birth date\n";

std::cout << "6) Get a list of all candidates\n";

std::cout << "7) Change container type\n";

std::cout << "8) Remove item by id\n";

std::cout << "9) Find item by id\n";

std::cout << "10) Update candidate name by id\n";

std::cout << "11) Update candidate rating by id\n";

std::cout << "12) Update candidate job place by id\n";

std::cout << "13) Update candidate birth date by id\n";

std::cout << "14) Help\n";

std::cout << "0) Exit\n";

std::cout <<"===========================================\n";

}

void Interactor::\_Interaction(const std::string& file\_to\_input)

{

int command = -1;

while (command != 0)

{

std::cout <<"Please, enter your command: ";

std::cin >> command;

std::vector<Candidate\*> result;

if (command == 0)

{

continue;

}

else if (command == 1)

{

try

{

\_container->AddItem(\_ReadCandidate(std::cin, true));

}

catch (std::logic\_error e)

{

std::cout << "There is so mistakes in data: " << e.what() << '\n';

}

}

else if (command == 2)

{

std::cout << "Please, enter full name: ";

std::string firstName;

std::string secondName;

std::string lastName;

std::cin >> firstName >> secondName >> lastName;

std::string fullName = firstName + " " + secondName + " " + lastName;

result = \_container->SearchByName(fullName);

}

else if (command == 3)

{

std::cout << "Please, enter a rating: ";

double rait;

std::cin >> rait;

std::cout << "Please, enter a type of seatching(L- less, M - more, E - equal): ";

char type;

std::cin >> type;

try

{

result = \_container->SearchByRating(rait, type);

}

catch(const std::exception& e)

{

std::cout << e.what() << '\n';

}

}

else if (command == 4)

{

std::cout << "Please, enter a place of job: ";

std::string job;

std::cin >> job;

result = \_container->SearchByJobPlace(job);

}

else if (command == 5)

{

std::cout << "Please, enter a date (dd.mm.yy): ";

int d, m, y;

std::cin >> d;

std::cin.get();

std::cin >> m;

std::cin.get();

std::cin >> y;

Date date(d,m,y);

std::cout << "Please, enter a type of seatching(L- less, M - more, E - equal): ";

char type;

std::cin >> type;

try

{

result = \_container->SearchByBirthDate(date, type);

}

catch(const std::exception& e)

{

std::cout << e.what() << '\n';

}

}

else if (command == 6)

{

result = \_container->GetCandidates();

}

else if (command == 7)

{

delete \_container;

\_ChooseContainer();

\_FileParse(file\_to\_input);

}

else if (command == 8)

{

std::cout << "Please, enter id: ";

int id;

std::cin >> id;

\_container->RemoveById(id);

}

else if (command == 9)

{

std::cout << "Please, enter id: ";

int id;

std::cin >> id;

Candidate\* candidate = \_container->FindById(id);

if (candidate == nullptr) std::cout << "Candiate with given id does not exist\n";

else std::cout << \*candidate;

}

else if (command == 10)

{

std::cout << "Please, enter id: ";

int id;

std::cin >> id;

std::cout << "Please, enter new full name: ";

std::string firstName;

std::string secondName;

std::string lastName;

std::cin >> firstName >> secondName >> lastName;

std::string fullName = firstName + " " + secondName + " " + lastName;

\_container->UpdateNameById(id, fullName);

}

else if (command == 11)

{

std::cout << "Please, enter id: ";

int id;

std::cin >> id;

std::cout << "Please, enter a new rating: ";

double rait;

std::cin >> rait;

\_container->UpdateRatingById(id, rait);

}

else if (command == 12)

{

std::cout << "Please, enter id: ";

int id;

std::cin >> id;

std::cout << "Please, enter a new place of job: ";

std::string job;

std::cin >> job;

\_container->UpdateJobPlaceById(id, job);

}

else if (command == 13)

{

std::cout << "Please, enter id: ";

int id;

std::cin >> id;

std::cout << "Please, enter a date (dd.mm.yy): ";

int d, m, y;

std::cin >> d;

std::cin.get();

std::cin >> m;

std::cin.get();

std::cin >> y;

try

{

Date date(d,m,y);

\_container->UpdateBirthDatById(id,date);

}

catch(std::logic\_error e)

{

std::cout << e.what() << '\n';

}

}

else if (command == 14)

{

\_Help();

}

for (auto c : result)

{

std::cout << \*c << std::endl;

}

}

}

Candidate Interactor::\_ReadCandidate(std::istream& in, bool greaters) const

{

std::ios::sync\_with\_stdio(0);

if (greaters) std::cout << "Please, enter full name: ";

std::string firstName;

std::string secondName;

std::string lastName;

in >> firstName >> secondName >> lastName;

std::string fullName = firstName + " " + secondName + " " + lastName;

if (greaters) std::cout << "Please, enter a date (dd.mm.yy): ";

int d, m, y;

in >> d;

in.get();

in >> m;

in.get();

in >> y;

Date date(d,m,y);

if (greaters) std::cout << "Please, enter a place of job: ";

std::string job;

in >> job;

if (greaters) std::cout << "Please, enter a rating: ";

double rait;

in >> rait;

return Candidate(fullName, date, job,rait);

}

void Interactor::\_FileParse(const std::string& file\_to\_input)

{

int n;

std::ifstream fin(file\_to\_input);

fin >> n;

for (int i = 0; i < n; ++i)

{

try {

\_container->AddItem(\_ReadCandidate(fin,false));

}

catch (std::logic\_error e)

{

std::cout << "There is so mistakes in file: " << e.what() << '\n';

exit(1);

}

}

fin.close();

}

void Interactor::\_ChooseContainer()

{

std::cout << "Choose to type of a container [V: Vector, M: Map]: ";

char type;

std::cin >> type;

ContainerFactory factory;

\_container = factory.GetContainer(toupper(type));

while (\_container == nullptr)

{

std::cout << "The given type doesn't exist. Choose from V and M [V: Vector, M: Map]: ";

std::cin >> type;

\_container = factory.GetContainer(toupper(type));

}

std::cout << "You choose " << (toupper(type) == 'V' ? "vector\n" : "map\n");

}

void Interactor::Init(const std::string& file\_to\_input)

{

\_ChooseContainer();

\_FileParse(file\_to\_input);

std::cout << "Files parsed from " << file\_to\_input << '\n';

\_Help();

\_Interaction(file\_to\_input);

}

**Date.cpp**

#include "Date.h"

#include <iostream>

Date::Date(int dd, int mm, int yy): \_day(dd), \_month(mm), \_year(yy)

{

if (dd < 1 || dd > 31)

throw std::logic\_error("bad day argument\n");

if (mm < 1 || mm > 12)

throw std::logic\_error("bad mounth argument\n");

if (yy < 0)

throw std::logic\_error("bad year argument\n");

}

Date::Date(): \_day(1), \_month(1), \_year(0) {}

int Date::GetDay() const

{

return \_day;

}

int Date::GetMonth() const

{

return \_month;

}

int Date::GetYear() const

{

return \_year;

}

bool Date::operator==(const Date& other) const

{

return (\_day == other.\_day) && (\_month == other.\_month) && (\_year == other.\_year);

}

bool Date::operator!=(const Date& other) const

{

bool equal = (\_day == other.\_day) && (\_month == other.\_month) && (\_year == other.\_year);

return !equal;

}

bool Date::operator<(const Date& other) const

{

int date1 = GetDaysCount();

int date2 = other.GetDaysCount();

if (date1 < date2)

{

return true;

}

return false;

}

bool Date::operator>(const Date& other) const

{

int date1 = GetDaysCount();

int date2 = other.GetDaysCount();

if (date1 > date2)

{

return true;

}

return false;

}

std::ostream& operator<<(std::ostream& out, const Date& object)

{

out << object.\_day << '.' << object.\_month << '.' << object.\_year;

return out;

}

int Date::GetDaysCount() const

{

int res = \_day;

for (int i = 1; i < \_month; i++)

{

res += GetDaysCountInMonth(\_month, \_year);

}

int leapYearsCount = \_year / 4;

int yearsCount = \_year - leapYearsCount;

res += leapYearsCount \* 366 + yearsCount \* 365;

return res;

}

int Date::GetDaysCountInMonth(int month, int year)

{

if (month == 4 || month == 6 || month == 9 || month == 11)

{

return 30;

}

else if (month == 2)

{

if (GetDaysCountInYear(year) == 365)

{

return 28;

}

else

{

return 29;

}

}

else

{

return 31;

}

}

int Date::GetDaysCountInYear(int year)

{

if (year % 4 != 0 || year % 100 == 0 && year % 400 != 0)

{

return 365;

}

return 366;

}

**Candidate.cpp**

#include "Candidate.h"

#include <iostream>

int Candidate::s\_idGenerator = 1;

Candidate::Candidate(const std::string& fullName,

const Date& dateOfBirth,

const std::string& job,

const double& rating): \_fullName(fullName),

\_dateOfBirth(dateOfBirth),

\_job(job),

\_rating(rating),

\_objectID(s\_idGenerator++)

{

if (\_rating < 0.0)

throw std::logic\_error("bad rating\n");

}

int Candidate::GetID() const

{

return \_objectID;

}

std::string Candidate::GetFullName() const

{

return \_fullName;

}

Date Candidate::GetDateOfBirth() const

{

return \_dateOfBirth;

}

std::string Candidate::GetJob() const

{

return \_job;

}

double Candidate::GetRating() const

{

return \_rating;

}

std::ostream& operator<<(std::ostream& out, const Candidate& object)

{

out << "---------------------------------------------------\n"

<< "Id: " << object.\_objectID <<"\n"

<< "Full name: " << object.\_fullName <<"\n"

<< "Date of birth: " << object.\_dateOfBirth <<"\n"

<< "Place of job: " << object.\_job <<"\n"

<< "Raiting: " << object.\_rating <<"\n";

return out;

}

**CandidateVector.cpp**

#include "CandidateVector.h"

#include <math.h>

void CandidateVector::AddItem(const Candidate& item)

{

\_candidates.push\_back(item);

}

void CandidateVector::RemoveById(int id)

{

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

{

if (\_candidates[i].GetID() == id)

{

\_candidates.erase(\_candidates.begin() + i);

break;

}

}

}

Candidate\* CandidateVector::FindById(int id)

{

for (Candidate& c : \_candidates)

{

if (c.GetID() == id) return &c;

}

return nullptr;

}

void CandidateVector::UpdateRatingById(int id, double rating)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr) toUpdate->\_rating = rating;

}

void CandidateVector::UpdateNameById(int id, const std::string& name)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr) toUpdate->\_fullName = name;

}

void CandidateVector::UpdateJobPlaceById(int id, const std::string& job)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr) toUpdate->\_job = job;

}

void CandidateVector::UpdateBirthDatById(int id, Date date)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr) toUpdate->\_dateOfBirth = date;

}

std::vector<Candidate\*> CandidateVector::SearchByRating(double rait, char mode)

{

if (mode != 'E' && mode != 'L' && mode != 'M')

{

throw std::invalid\_argument("Bad comporation type");

}

std::vector<Candidate\*> toReturn;

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

{

if (mode == 'E' && (abs(\_candidates[i].GetRating() - rait) < 1e6))

{

toReturn.push\_back(&\_candidates[i]);

}

else if (mode == 'L' && \_candidates[i].GetRating() < rait)

{

toReturn.push\_back(&\_candidates[i]);

}

else if (mode == 'M' && \_candidates[i].GetRating() > rait)

{

toReturn.push\_back(&\_candidates[i]);

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateVector::SearchByName(const std::string& name)

{

std::vector<Candidate\*> toReturn;

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

{

if (\_candidates[i].GetFullName() == name)

{

toReturn.push\_back(&\_candidates[i]);

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateVector::SearchByJobPlace(const std::string& job)

{

std::vector<Candidate\*> toReturn;

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

{

if (\_candidates[i].GetFullName() == job)

{

toReturn.push\_back(&\_candidates[i]);

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateVector::SearchByBirthDate(Date date, char mode)

{

if (mode != 'E' && mode != 'L' && mode != 'M')

{

throw std::invalid\_argument("Bad comporation type");

}

std::vector<Candidate\*> toReturn;

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

{

if (mode == 'E' && date == \_candidates[i].GetDateOfBirth())

{

toReturn.push\_back(&\_candidates[i]);

}

else if (mode == 'L' && \_candidates[i].GetDateOfBirth() < date)

{

toReturn.push\_back(&\_candidates[i]);

}

else if (mode == 'M' && \_candidates[i].GetDateOfBirth() > date)

{

toReturn.push\_back(&\_candidates[i]);

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateVector::GetCandidates()

{

std::vector<Candidate\*> toReturn(\_candidates.size());

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

{

toReturn[i] = &\_candidates[i];

}

return toReturn;

}

**CandidateMap.cpp**

#include "CandidateMap.h"

void CandidateMap::AddItem(const Candidate& item)

{

int id = item.GetID();

\_candidates.insert(std::pair<int, Candidate>(id,item));

auto itRating = \_candidatesByRating.find(item.GetRating());

auto itName = \_candidatesByName.find(item.GetFullName());

auto itJob = \_candidatesByJob.find(item.GetJob());

auto itDate = \_candidatesByBirthDate.find(item.GetDateOfBirth());

if (itRating != \_candidatesByRating.end())

{

itRating->second.insert(id);

}

else

{

\_candidatesByRating[item.GetRating()] = std::set<int>();

\_candidatesByRating[item.GetRating()].insert(id);

}

if (itName != \_candidatesByName.end())

{

itName->second.insert(id);

}

else

{

\_candidatesByName[item.GetFullName()] = std::set<int>();

\_candidatesByName[item.GetFullName()].insert(id);

}

if (itJob != \_candidatesByJob.end())

{

itJob->second.insert(id);

}

else

{

\_candidatesByJob[item.GetJob()] = std::set<int>();

\_candidatesByJob[item.GetJob()].insert(id);

}

if (itDate != \_candidatesByBirthDate.end())

{

itDate->second.insert(id);

}

else

{

\_candidatesByBirthDate[item.GetDateOfBirth()] = std::set<int>();

\_candidatesByBirthDate[item.GetDateOfBirth()].insert(id);

}

}

void CandidateMap::RemoveById(int id)

{

auto it = \_candidates.find(id);

if (it != \_candidates.end())

{

\_candidatesByRating[it->second.GetRating()].erase(id);

\_candidatesByName[it->second.GetFullName()].erase(id);

\_candidatesByJob[it->second.GetJob()].erase(id);

\_candidatesByBirthDate[it->second.GetDateOfBirth()].erase(id);

\_candidates.erase(id);

}

}

Candidate\* CandidateMap::FindById(int id)

{

auto it = \_candidates.find(id);

if (it != \_candidates.end()) return &(it->second);

return nullptr;

}

void CandidateMap::UpdateRatingById(int id, double rating)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr)

{

\_candidatesByRating[toUpdate->\_rating].erase(id);

\_candidatesByRating[rating].insert(id);

toUpdate->\_rating = rating;

}

}

void CandidateMap::UpdateNameById(int id, const std::string& name)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr)

{

\_candidatesByName[toUpdate->\_fullName].erase(id);

\_candidatesByName[name].insert(id);

toUpdate->\_fullName = name;

}

}

void CandidateMap::UpdateJobPlaceById(int id, const std::string& job)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr)

{

\_candidatesByJob[toUpdate->\_job].erase(id);

\_candidatesByJob[job].insert(id);

toUpdate->\_job = job;

}

}

void CandidateMap::UpdateBirthDatById(int id, Date date)

{

Candidate\* toUpdate = FindById(id);

if (toUpdate != nullptr)

{

\_candidatesByBirthDate[toUpdate->\_dateOfBirth].erase(id);

\_candidatesByBirthDate[date].insert(id);

toUpdate->\_dateOfBirth = date;

}

}

std::vector<Candidate\*> CandidateMap::SearchByRating(double rait, char mode)

{

if (mode != 'E' && mode != 'L' && mode != 'M')

{

throw std::invalid\_argument("Bad comporation type");

}

std::vector<Candidate\*> toReturn;

if (mode == 'E')

{

auto it = \_candidatesByRating.find(rait);

if (it != \_candidatesByRating.end())

{

auto itSet = it->second.begin();

for (;itSet != it->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

}

else if (mode == 'L')

{

auto to = \_candidatesByRating.lower\_bound(rait);

to++;

for (auto it = \_candidatesByRating.begin(); it != to; ++it)

{

if (it == \_candidatesByRating.end() || it->first == rait) continue;

else

{

auto itSet = it->second.begin();

for (;itSet != it->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

}

}

else if (mode == 'M')

{

auto from = \_candidatesByRating.upper\_bound(rait);

for (; from != \_candidatesByRating.end(); ++from)

{

auto itSet = from->second.begin();

for (;itSet != from->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateMap::SearchByName(const std::string& name)

{

std::vector<Candidate\*> toReturn;

auto it = \_candidatesByName.find(name);

if (it != \_candidatesByName.end())

{

auto itSet = it->second.begin();

for (;itSet != it->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateMap::SearchByJobPlace(const std::string& job)

{

std::vector<Candidate\*> toReturn;

auto it = \_candidatesByJob.find(job);

if (it != \_candidatesByJob.end())

{

auto itSet = it->second.begin();

for (;itSet != it->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateMap::SearchByBirthDate(Date date, char mode)

{

if (mode != 'E' && mode != 'L' && mode != 'M')

{

throw std::invalid\_argument("Bad comporation type");

}

std::vector<Candidate\*> toReturn;

if (mode == 'E')

{

auto it = \_candidatesByBirthDate.find(date);

if (it != \_candidatesByBirthDate.end())

{

auto itSet = it->second.begin();

for (;itSet != it->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

}

else if (mode == 'L')

{

auto to = \_candidatesByBirthDate.lower\_bound(date);

to++;

for (auto it = \_candidatesByBirthDate.begin(); it != to; ++it)

{

if (it == \_candidatesByBirthDate.end() || it->first == date) continue;

else

{

auto itSet = it->second.begin();

for (;itSet != it->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

}

}

else if (mode == 'M')

{

auto from = \_candidatesByBirthDate.upper\_bound(date);

for (; from != \_candidatesByBirthDate.end(); ++from)

{

auto itSet = from->second.begin();

for (;itSet != from->second.end();++itSet)

{

toReturn.push\_back(&\_candidates[\*itSet]);

}

}

}

return toReturn;

}

std::vector<Candidate\*> CandidateMap::GetCandidates()

{

std::vector<Candidate\*> toReturn;

for (auto it = \_candidates.begin(); it != \_candidates.end(); ++it)

{

toReturn.push\_back(&it->second);

}

return toReturn;

}

**ContainerFactory.cpp**

#include "ContainerFactory.h"

ICandidateContainer\* ContainerFactory::GetContainer(char type)

{

if (type == 'V')

{

return new CandidateVector;

}

else if (type == 'M')

{

return new CandidateMap;

}

else

{

return nullptr;

}

}