Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Рязанский государственный радиотехнический университет имени В. Ф. Уткина»

Кафедра «Вычислительная и прикладная математика»

Отчет по лабораторной работе №6

по дисциплине «Объектно-ориентированное программирование»

на тему «Исключения, контейнерные классы STL»

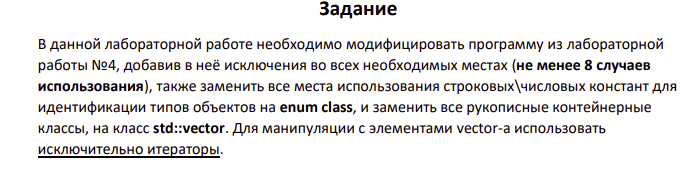
Выполнил: студент гр.1413

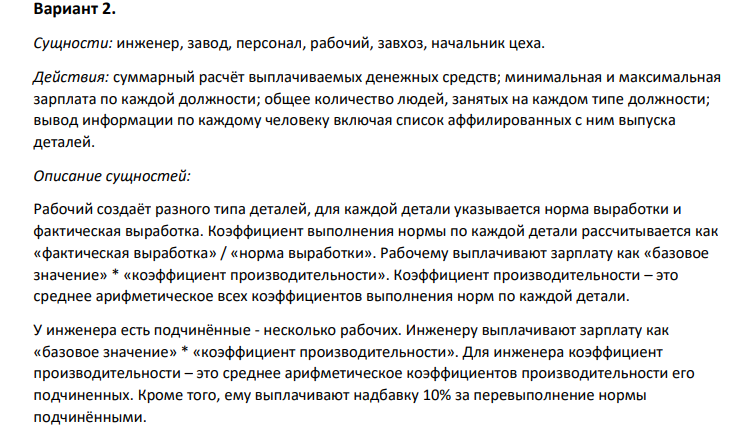
Аверкин М.Е.

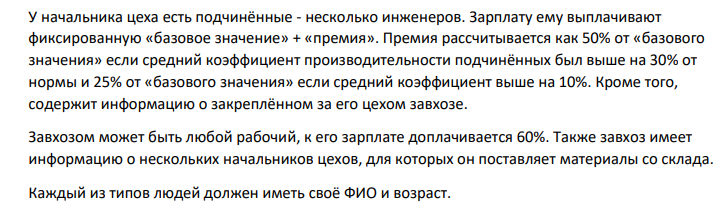
Проверил:

доц. Антипов О.В.

**Цель работы**: освоить исключения, контейнерные классы и итераторы STL.

**Задание**: 





**Анализ программы:**

**Входные данные:**

Случайным образом генерируемые сущности, имена которых выбираются из заранее определенных массивов имен. Количество сущностей классов Factory, Worker, Engineer, Foreman, Manager.

**Промежуточные данные:**

Переменные N, numberOfChoise, numberOfPersomal выступающие в качестве параметров цикла for и оператора switch.

**Выходные данные:**

Композитная сущность класса Factory, которая хранит в себе ссылки на объекты классов, наследуемых от базового класса Personal.

**Описание структуры:**

Personal – абстрактный класс, с чистыми виртуальными функциями printInfoAboutPersonal и calculateSalary, которые будут переопределяться в классах-наследниках.

Detail – класс, добавлен для реализации изготавливаемой продукции. Поля – numberOfDetails (число деталей), normRate (норма выработки деталей), name (название данного вида изделия), factRate (фактическая выработка), koef(коэффициент выполнения нормы по каждой детали).

Randomization – класс, добавлен для реализации рандомизации заполняемой информации о персонале. Содержит в себе 4 массива имен: для имен, фамилии, названия завода, название детали.

Engineer – класс – наследник класса Personal.. Выступает родительским классом для класса Worker. Хранит в себе список с информацией о починенных ему рабочих. Добавлены методы получения списка рабочих, получения количества подчиненных ему рабочих и среднего коэффициента выполнения нормы.

Foreman – класс – наследник от класса Personal.. Выступает родительским классом для класса: Engineer. Хранит в себе список с информацией о починенных ему инженерах. Добавлены методы получения списка инженеров, получения количества подчиненных ему инженеров и дополнен метод расчёта зарплаты.

Manager – класс – наследник от класса Personal. Выступает родительским классом для класса: Foreman. Хранит в себе список с информацией о начальниках цеха. Добавлен метод расчёта числа начальников цеха и получения списка начальников цеха, дополнен метод расчёта зарплаты.

Worker – класс – наследник от класса Personal. Хранит в себе список деталей с информацией о них и о их количестве. Добавлен метод расчета среднего коэффициента выполнения нормы.

Factory – композитная сущность, хранит в себе список указателей на объекты базовой сущности Personal. Имеет методы расчета суммарных выплат по всему персоналу.

**Листинг программы:**

**Factory.h**

#pragma once

#include "Personal.h"

#include "Worker.h"

#include "Engineer.h"

#include "GeneralOfDepartment.h"

#include "Manager.h"

#include "PersonalPost.h"

class Factory{

private:

vector<Personal\*> listOfPersonals;

string \_name;

public:

~Factory();

Factory(const string name);

Personal& GetPersonal(int index) const {

if (index < 0 || index >= listOfPersonals.size()) {

throw("invalid index");

}

return \*listOfPersonals[index];

}

Personal& GetPersonalByPost(PersonalPost::PersonalPost type, int index) const {

if (type == PersonalPost::PersonalPost::DEFAULT) {

throw ("Position not defined");

}

int Counter = 0;

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

if ((\*it)->getPost() == PersonalPost::PersonalPost(type)) {

Counter++;

if (Counter == index)

return (\*\*it);

}

}

throw("wrong index");

}

float getPayouts();

void printInfoAboutMinMaxSalary();

void printInfoAboutPersonal();

void printInfoAboutPersonalByPost();

void deleteById(int id, PersonalPost::PersonalPost type);

void AddPerson(const string fio, int age, int BaseValue, PersonalPost::PersonalPost type);

const string getNameOfFactory() {

return \_name;

}

int getCapacityOfPersonals(){

return listOfPersonals.size();

}

};

**Factory.cpp**

#include "Factory.h"

#include "Personal.h"

#include "Worker.h"

#include "Engineer.h"

#include "Manager.h"

#include "GeneralOfDepartment.h"

Factory::~Factory(){

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

delete \*it;

}

cout << "Factory dtor" << endl;

}

void Factory::AddPerson(const string fio, int age, int BaseValue, PersonalPost::PersonalPost type){

if (type == PersonalPost::PersonalPost::DEFAULT) {

throw ("Position not defined");

}

auto it = listOfPersonals.end();

switch (type)

{

case PersonalPost::PersonalPost::WORKER:

listOfPersonals.insert(it, new Worker(fio, age, BaseValue));

break;

case PersonalPost::PersonalPost::ENGINEER:

listOfPersonals.insert(it, new Engineer(fio, age, BaseValue));

break;

case PersonalPost::PersonalPost::FOREMAN:

listOfPersonals.insert(it, new Foreman(fio, age, BaseValue));

break;

case PersonalPost::PersonalPost::MANAGER:

listOfPersonals.insert(it, new Manager(fio, age, BaseValue));

break;

}

}

Factory::Factory(const string name) {

this->\_name = name;

}

float Factory::getPayouts(){

float summPayout = 0.0f;

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

summPayout += (\*it)->calculateSalary();

}

return summPayout;

}

void Factory::printInfoAboutMinMaxSalary(){

for (int i = 1; i < 5; i++) {

cout << "For post " << PersonalPost::ToString(PersonalPost::PersonalPost(i)) << endl;

float minSalary = 1000000;

float maxSalary = 0;

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

if ((\*it)->getPost() == PersonalPost::PersonalPost(i)) {

if ((\*it)->calculateSalary() < minSalary) {

minSalary = (\*it)->calculateSalary();

}

}

}

if (minSalary == 1000000) {

minSalary = 0;

}

cout << "Minimal salary: " << minSalary << endl;

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

if ((\*it)->getPost() == PersonalPost::PersonalPost(i)) {

if ((\*it)->calculateSalary() > maxSalary) {

maxSalary = (\*it)->calculateSalary();

}

}

}

cout << "Maximum salary: " << maxSalary << endl;

}

}

void Factory::printInfoAboutPersonal(){

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

(\*it)->printInfoAboutPersonal();

}

}

void Factory::printInfoAboutPersonalByPost(){

for (int i = 1; i < 5; i++) {

int totalCounter = 0;

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

if ((\*it)->getPost() == PersonalPost::PersonalPost(i)) {

totalCounter++;

}

}

cout << "Total number of personal on post " << PersonalPost::ToString(PersonalPost::PersonalPost(i)) << " : " << totalCounter << endl;

}

}

void Factory::deleteById(int index, PersonalPost::PersonalPost type){

if (type == PersonalPost::PersonalPost::DEFAULT) {

throw ("Position not defined");

}

int Counter = 0;

for (auto it = listOfPersonals.begin(); it != listOfPersonals.end(); ++it) {

if ((\*it)->getPost() == PersonalPost::PersonalPost(type)) {

Counter++;

if (Counter == index) {

delete (\*it);

listOfPersonals.erase(it);

return;

}

}

}

throw("wrong index");

}

**Personal.h**

#pragma once

#include <stdlib.h>

#include <string>

#include "Detail.h"

#include "PersonalPost.h"

using namespace std;

class Personal

{

private:

int \_BaseValue;

int \_age;

string \_fio;

public:

virtual void printInfoAboutPersonal() const;

virtual ~Personal();

virtual float getBaseValue() const;

virtual float getAverageKoef() const = 0;

virtual float calculateSalary() const;

Personal(const string fio, int age, int BaseValue);

string getFio() {

return \_fio;

}

void setBaseValue(int BaseValue) {

this->\_BaseValue = BaseValue;

}

int getAge() {

return \_age;

}

void setAge(int age) {

this->\_age = age;

}

virtual PersonalPost::PersonalPost getPost() const {

return PersonalPost::PersonalPost::DEFAULT;

}

};

**Personal.cpp**

#include "Personal.h"

void Personal::printInfoAboutPersonal() const{

cout << "Job title: " << PersonalPost::ToString(getPost()) << endl;

cout << "FIO: " << \_fio << endl;

cout << "Age: " << \_age << endl;

cout << "Salary: " << calculateSalary() << endl;

cout << endl;

}

Personal::~Personal(){

cout << "Personal dtor" << endl;

}

float Personal::getBaseValue() const{

return \_BaseValue;

}

float Personal::calculateSalary() const{

return getAverageKoef() \* getBaseValue();

}

Personal::Personal(const string fio, int age, int BaseValue)

:\_age(age),\_BaseValue(BaseValue){

this-> \_fio = fio;

}

**Worker.h**

#pragma once

#include "Detail.h"

#include "Personal.h"

#include "Except.h"

#include <vector>

class Worker:public Personal

{private:

vector<Detail\*> listOfDetails;

public:

~Worker() override;

Worker(string name, int age, int BaseValue);

PersonalPost::PersonalPost getPost() const override {

return PersonalPost::PersonalPost::WORKER;

}

void AddDetail(string name, int normRate, float factRate);

float getAverageKoef() const override;

void printInfoAboutPersonal() const override;

};

**Worker.cpp**

#include "Worker.h"

Worker::~Worker(){

for (auto it = listOfDetails.begin(); it != listOfDetails.end(); ++it) {

delete\* it;

}

cout << "Worker dtor" << endl;

}

Worker::Worker(const string fio, int age, int BaseValue)

:Personal(fio, age, BaseValue){

}

void Worker::AddDetail(const string name, int normRate, float factRate) {

auto it = listOfDetails.end();

listOfDetails.insert(it, new Detail(name, normRate, factRate));

}

float Worker::getAverageKoef() const {

float averageKoef = 0.0f;

float summaryKoef = 0.0f;

if (listOfDetails.size() == 0) {

throw Exception("A worker without parts is unacceptable.");

}

for (auto it = listOfDetails.begin(); it != listOfDetails.end(); ++it) {

summaryKoef += (\*it)->getKoef();

}

averageKoef = summaryKoef / listOfDetails.size();

return averageKoef;

}

void Worker::printInfoAboutPersonal() const{

cout << "---------------------------------------------" << endl;

Personal::printInfoAboutPersonal();

for (auto it = listOfDetails.begin(); it != listOfDetails.end(); ++it) {

(\*it)->toString();

}

cout << "---------------------------------------------" << endl;

cout << endl;

}

**Detail.h**

#pragma once

#include <stdlib.h>

#include <string>

#include <iostream>

using namespace std;

class Detail

{

private:

string \_name;

public:

int normRate;

float factRate;

float koef;

Detail(const string name, int normRate, float factRate);

~Detail();

string getName() {

return \_name;

}

void toString() {

cout << "Name of detail: " << \_name << endl;

cout << "NormRate of detail: " << normRate << endl;

cout << "FactRate of detail: " << factRate << endl;

cout << "Koef of detail: " << koef << endl;

cout << endl;

}

void setName(const string name) {

this->\_name = name;

}

int getNormRate() {

return normRate;

}

void setNormRate(int normRate) {

this->normRate = normRate;

}

float getFactRate() {

return factRate;

}

void setFactRate(int factRate) {

this->factRate = factRate;

}

float getKoef() {

return koef;

}

};

**Detail.cpp**

#include "Detail.h"

Detail::Detail(const string name, int normRate, float factRate) {

this->\_name = name;

this->normRate = normRate;

this->factRate = factRate;

this->koef = factRate / normRate;

}

Detail::~Detail(){

cout << "Detail dtor" << endl;

}

**Engineer.h**

#pragma once

#include "Worker.h"

class Engineer:public Personal

{

private:

vector<Worker\*> listOfWorkers;

public:

~Engineer() override;

Engineer(const string fio, int age, int BaseValue);

PersonalPost::PersonalPost getPost() const override {

return PersonalPost::PersonalPost::ENGINEER;

}

void addWorker(Worker& worker);

float calculateSalary() const override;

void printInfoAboutPersonal() const override;

float getAverageKoef() const override;

}

**Engineer.cpp**

#include "Engineer.h"

Engineer::~Engineer(){

cout << "Engineer dtor" << endl;

}

Engineer::Engineer(const string fio, int age, int BaseValue)

:Personal(fio, age, BaseValue) {

}

void Engineer::addWorker(Worker& worker) {

auto it = listOfWorkers.end();

listOfWorkers.insert(it, &worker);

}

void Engineer::printInfoAboutPersonal() const{

cout << "---------------------------------------------" << endl;

Personal::printInfoAboutPersonal();

for (auto it = listOfWorkers.begin(); it != listOfWorkers.end(); ++it) {

(\*it)->Personal::printInfoAboutPersonal();

}

cout << "---------------------------------------------" << endl;

cout << endl;

}

float Engineer::getAverageKoef() const{

float averageKoef = 0.0f;

float summaryKoef = 0.0f;

if (listOfWorkers.size() == 0) {

throw Exception("An engineer without workers is unacceptable.");

}

for (auto it = listOfWorkers.begin(); it != listOfWorkers.end(); ++it) {

summaryKoef += (\*it) -> getAverageKoef();

}

averageKoef = summaryKoef / listOfWorkers.size();

return averageKoef;

}

float Engineer::calculateSalary() const{

float averageKoef = 0.0f;

averageKoef = this->getAverageKoef();

float surchargeMultiplier = 1.0f;

if (averageKoef > 1){

surchargeMultiplier = 1.1;

}

return getBaseValue() \* surchargeMultiplier \* averageKoef;

}

**GeneralOfDepartment.h**

#pragma once

#include "Engineer.h"

class Manager;

class Foreman: public Personal

{

private:

vector<Engineer\*> listOfEngineers;

Manager\* manager = nullptr;

public:

~Foreman() override;

Foreman(const string fio, int age, int BaseValue);

PersonalPost::PersonalPost getPost() const override {

return PersonalPost::PersonalPost::FOREMAN;

}

void setManager(Manager\* manager) {

this->manager = manager;

}

void addEngineer(Engineer& engineer);

float calculateSalary() const override;

void printInfoAboutPersonal() const override;

float getAverageKoef() const override;

};

**GeneralOfDepartment.cpp**

#include "GeneralOfDepartment.h"

#include "Manager.h"

Foreman::~Foreman(){

cout << "Foreman dtor" << endl;

}

Foreman::Foreman(const string fio, int age, int BaseValue)

:Personal(fio, age, BaseValue) {

}

void Foreman::printInfoAboutPersonal() const{

cout << "---------------------------------------------" << endl;

Personal::printInfoAboutPersonal();

for (auto it = listOfEngineers.begin(); it != listOfEngineers.end(); ++it) {

(\*it)->Personal::printInfoAboutPersonal();

}

if (manager == nullptr) {

throw "No manager, not enough data to process";

}

else {

manager->Personal::printInfoAboutPersonal();

}

cout << "---------------------------------------------" << endl;

cout << endl;

}

void Foreman::addEngineer(Engineer& engineer) {

auto it = listOfEngineers.end();

listOfEngineers.insert(it, &engineer);

}

float Foreman::getAverageKoef() const {

float averageKoef = 0.0f;

float summaryKoef = 0.0f;

if (listOfEngineers.size() == 0) {

throw Exception("A foreman without engineers is unacceptable.");

}

for (auto it = listOfEngineers.begin(); it != listOfEngineers.end(); ++it) {

summaryKoef += (\*it)->getAverageKoef();

}

averageKoef = summaryKoef / listOfEngineers.size();

return averageKoef;

}

float Foreman::calculateSalary() const{

float averageKoef = 0.0f;

averageKoef = this->getAverageKoef();

float surchargeMultiplier = 1.0f;

if (averageKoef > 1.3) {

surchargeMultiplier = 1.5f;

}

else if (averageKoef > 1.1) {

surchargeMultiplier = 1.25f;

}

return getBaseValue() \* surchargeMultiplier;

}

**Manager.h**

#pragma once

#include "Worker.h"

class Foreman;

class Manager :public Worker

{

private:

vector<Foreman\*> listOfForemans;

public:

~Manager() override;

Manager(const string fio, int age, int BaseValue)

:Worker(fio, age, BaseValue) {

}

PersonalPost::PersonalPost getPost() const override {

return PersonalPost::PersonalPost::MANAGER;

}

void addForeman(Foreman& foreman);

float calculateSalary() const override;

void printInfoAboutPersonal() const override;

float getAverageKoef()const override;

};

**Manager.cpp**

#include "Manager.h"

#include "GeneralOfDepartment.h"

Manager::~Manager(){

cout << "Manager dtor" << endl;

}

void Manager::addForeman(Foreman& foreman) {

auto it = listOfForemans.end();

listOfForemans.insert(it, &foreman);

}

void Manager::printInfoAboutPersonal() const {

cout << "---------------------------------------------" << endl;

Personal::printInfoAboutPersonal();

for (auto it = listOfForemans.begin(); it != listOfForemans.end(); ++it) {

(\*it)->Personal::printInfoAboutPersonal();

}

cout << "---------------------------------------------" << endl;

cout << endl;

}

float Manager::getAverageKoef() const {

return getAverageKoef();

}

float Manager::calculateSalary() const {

return 1.6f \* getBaseValue();

}

**Randomization.h**

#pragma once

#include<iostream>

#include<stdlib.h>

#include <string>

using namespace std;

static const string ranFackName[10] = {

"Microelectronics",

"LabaevArms",

"RadioEngineering",

"Avto",

"RedFlag",

"Automobile",

"Millitary",

"Metell",

"Litie",

"RanNaMe"

};

static const string ranDetailName[10] = {

"Microelectronic",

"Plata",

"Software",

"Toyota",

"Barrel",

"Gun",

"RanDetAIl",

"Transistor",

"Antenna",

"Refrezeraitor"

};

static const string ranName[65] = {

"Kaisha",

"Alexandria",

"Hania",

"Heidi",

"Arthur",

"Shanae",

"Haydn",

"Amara",

"Aleah",

"Emilie",

"Arissa",

"Ryker",

"Mathild",

"Ocean",

"Jacqueline",

"Heather",

"Rafferty",

"Umar",

"Nansi",

"Ethel",

"Kiaan",

"Chyna",

"Drake",

"Loretta",

"Aarron",

"Henri",

"Jillian",

"Dani",

"Shereen",

"Shayaan",

"Daisy",

"Piers",

"Firat",

"Summer",

"Maxime",

"Letitia",

"Kiki",

"Henrietta",

"Cheryl",

"Ayyan",

"Ishmael",

"Shreya",

"Danish",

"Uwais",

"Storm",

"Ewan",

"Poppy",

"Lani",

"Roseanne",

"Will",

"Saul",

"Bibi",

"Alishba",

"Viaan",

"Matias",

"Dominick",

"Amber",

"Christopher",

"Osama",

"Arjan",

"Rhonda",

"Ziggy",

"Savannah",

"Judith",

"James",

};

static const string ranSurname[65] = {

"Sierra",

"Ramos",

"Edge",

"Pratt",

"Read",

"Calderon",

"Crane",

"Waters",

"Rawlings",

"Shepherd",

"Russell",

"Felix",

"Ball",

"Golden",

"Fields",

"Bird",

"Hoover",

"Vinson",

"Tillman",

"Nicholson",

"Sparks",

"Petty",

"Truong",

"Kenny",

"Singleton",

"Kramer",

"Little",

"Buckley",

"Wicks",

"York",

"Neale",

"Lin",

"Jefferson",

"Hough",

"Bradford",

"Alexander",

"Bryan",

"Hulme",

"Wolf",

"Coffey",

"Ratcliffe",

"Dawe",

"Vang",

"Arnold",

"Glenn",

"Figueroa",

"Crouch",

"Davies",

"Simon",

"Choi",

"Chung",

"Leigh",

"AliZhba",

"Taylor",

"Mohammed",

"Mcknight",

"Todd",

"Mills",

"Orr",

"Nash",

"Mora",

"Cannon",

"Corona",

"Summers",

"Lake",

};

class Randomization

{

public:

static int getRanAge() {

int age = 18 + (rand() \* 10) % (75 - 18 + 1);

return age;

}

static float getRanSalary() {

float Salary = 22000 + (rand() \* 5512) % (220000 - 22000 + 1);

return Salary;

}

static const string getFackName() {

int i = (rand() \* 5) % (10 - 0);

const string fackName = ranFackName[i];

return fackName;

}

static const string getRanFIO() {

int i = (rand() \* 10) % (64 - 0);

string FIO;

const string name = ranName[i];

const string surname = ranSurname[i];

FIO = name + " " + surname;

return FIO;

}

static float getRanK1() {

float Koeff = 40 + (rand() \* 10) % (150 - 40 + 1);

return Koeff;

}

static float getRanK2() {

float Koeff = 40 + (rand() \* 15) % (150 - 40 + 1);

return Koeff;

}

static const string getDetailName() {

int i = rand() % (10);

const string detailName = ranDetailName[i];

return detailName;

}

};

**PersonalPost.h**

#pragma once

namespace PersonalPost

{

enum class PersonalPost

{

DEFAULT = 0,

WORKER,

ENGINEER,

FOREMAN,

MANAGER

};

inline const string ToString(PersonalPost type)

{

switch (type)

{

case PersonalPost::WORKER:

return "Worker";

case PersonalPost::ENGINEER:

return "Engineer";

case PersonalPost::FOREMAN:

return "Foreman";

case PersonalPost::MANAGER:

return "Manager";

default:

return "Default";

}

}

}

**Except.h**

#pragma once

namespace PersonalPost

{

enum class PersonalPost

{

DEFAULT = 0,

WORKER,

ENGINEER,

FOREMAN,

MANAGER

};

inline const string ToString(PersonalPost type)

{

switch (type)

{

case PersonalPost::WORKER:

return "Worker";

case PersonalPost::ENGINEER:

return "Engineer";

case PersonalPost::FOREMAN:

return "Foreman";

case PersonalPost::MANAGER:

return "Manager";

default:

return "Default";

}

}

}

**LabaevARMS4.cpp**

#include <iostream>

#include "Detail.h"

#include "Worker.h"

#include "Engineer.h"

#include "GeneralOfDepartment.h"

#include "Manager.h"

#include "Factory.h"

#include "Randomization.h"

#include<ctime>

using namespace std;

int main() {

int numOfChoiseS, numOfChoise, numOfChoiseE;

int N, numberOfPersonal;

Factory newFactory(Randomization::getFackName());

int counterWorkers = 0;

int counterEngineers = 0;

int counterForemans = 0;

int counterManagers = 0;

int AddedWorker = 0;

int AddedEngineer = 0;

int AddedForeman = 0;

srand(time(NULL));

cout << "The program implies the sequential filling of the database of workers" << endl;

do {

cout << "add(1) or delete?(2)" << endl;

cin >> numOfChoiseS;

switch (numOfChoiseS)

{

case 1:

{

cout << "What entities to add?" << endl;

cout << "1 - Worker and Detail" << endl;

cout << "2 - Engineer" << endl;

cout << "3 - Foreman" << endl;

cout << "4 - Manager" << endl;

cin >> numOfChoise;

if (numOfChoise == 1) {

cout << "How many Workers to add?" << endl;

cin >> N;

for (int i = 1; i <= N; i++) {

newFactory.AddPerson(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary(), PersonalPost::PersonalPost::WORKER);

counterWorkers++;

int numberOfDetailsForWorker = 0;

cout << "How many Details to add?" << endl;

cin >> numberOfDetailsForWorker;

for (int j = 1; j <= numberOfDetailsForWorker; j++) {

((Worker&)newFactory.GetPersonal(newFactory.getCapacityOfPersonals() - 1)).AddDetail(Randomization::getDetailName(), Randomization::getRanK1(), Randomization::getRanK2());

}

}

}

else if (numOfChoise == 2) {

cout << "How many Engineers to add?" << endl;

cin >> N;

for (int i = 1; i <= N; i++) {

newFactory.AddPerson(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary(), PersonalPost::PersonalPost::ENGINEER);

counterEngineers++;

int numberOfWorkersForEngineer = 0;

cout << "How many workers do you need to add to this Engineer?" << endl;

cin >> numberOfWorkersForEngineer;

while ((counterWorkers - AddedWorker) < numberOfWorkersForEngineer){

cout << "You are adding too much." << endl;

cin >> numberOfWorkersForEngineer;

}

for (int j = 1; j <= numberOfWorkersForEngineer; j++) {

AddedWorker++;

((Engineer&)newFactory.GetPersonal(newFactory.getCapacityOfPersonals() - 1)).addWorker((Worker&)newFactory.GetPersonalByPost(PersonalPost::PersonalPost::WORKER, AddedWorker));

}

}

}

else if (numOfChoise == 3) {

cout << "How many Foremanrs to add?" << endl;

cin >> N;

for (int i = 1; i <= N; i++) {

newFactory.AddPerson(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary(), PersonalPost::PersonalPost::FOREMAN);

counterForemans++;

int numberOfEngineersForForeman = 0;

cout << "How many engineers do you need to add to this Foreman?" << endl;

cin >> numberOfEngineersForForeman;

while ((counterEngineers - AddedEngineer) < numberOfEngineersForForeman) {

cout << "You are adding too much." << endl;

cin >> numberOfEngineersForForeman;

}

for (int j = 1; j <= numberOfEngineersForForeman; j++) {

AddedEngineer++;

((Foreman&)newFactory.GetPersonal(newFactory.getCapacityOfPersonals() - 1)).addEngineer((Engineer&)newFactory.GetPersonalByPost(PersonalPost::PersonalPost::ENGINEER, AddedEngineer));

}

}

}

else if (numOfChoise == 4) {

cout << "How many Managers to add?" << endl;

cin >> N;

for (int i = 1; i <= N; i++) {

newFactory.AddPerson(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary(), PersonalPost::PersonalPost::MANAGER);

counterManagers++;

int numberOfForemansForManager = 0;

cout << "How many foremans do you need to add to this Manager?" << endl;

cin >> numberOfForemansForManager;

while ((counterForemans - AddedForeman) < numberOfForemansForManager) {

cout << "You are adding too much." << endl;

cin >> numberOfForemansForManager;

}

for (int j = 1; j <= numberOfForemansForManager; j++) {

AddedForeman++;

((Manager&)newFactory.GetPersonal(newFactory.getCapacityOfPersonals() - 1)).addForeman((Foreman&)newFactory.GetPersonalByPost(PersonalPost::PersonalPost::FOREMAN, AddedForeman));

((Foreman&)newFactory.GetPersonalByPost(PersonalPost::PersonalPost::FOREMAN, AddedForeman)).setManager((Manager\*)&newFactory.GetPersonalByPost(PersonalPost::PersonalPost::MANAGER, counterManagers));

}

}

}

else {

cout << "1)Only 1 - 4!" << endl;

cout << "2)Database full" << endl;

}

}

break;

case 2:

cout << "What entities to delete?" << endl;

cout << "1 - Worker and Detail" << endl;

cout << "2 - Engineer" << endl;

cout << "3 - Foreman" << endl;

cout << "4 - Manager" << endl;

cin >> numOfChoise;

if (numOfChoise == 1) {

cout << "What number of Workers to delete?" << endl;

cout << "Enter Personal number" << endl;

cin >> numberOfPersonal;

if (numberOfPersonal > counterWorkers) {

cout << "This number is not in the database." << endl;

}

else {

newFactory.deleteById(numberOfPersonal, PersonalPost::PersonalPost::WORKER);

}

}

else if (numOfChoise == 2) {

cout << "What number of Engineers to delete?" << endl;

cout << "Enter Personal number" << endl;

cin >> numberOfPersonal;

if (numberOfPersonal > counterEngineers) {

cout << "This number is not in the database." << endl;

}

else {

newFactory.deleteById(numberOfPersonal, PersonalPost::PersonalPost::ENGINEER);

}

}

else if (numOfChoise == 3) {

cout << "What number of Foremans to delete??" << endl;

cout << "Enter Personal number" << endl;

cin >> numberOfPersonal;

if (numberOfPersonal > counterForemans) {

cout << "This number is not in the database." << endl;

}

else {

newFactory.deleteById(numberOfPersonal, PersonalPost::PersonalPost::FOREMAN);

}

}

else if (numOfChoise == 4) {

cout << "What number of Managers to delete??" << endl;

cout << "Enter Personal number" << endl;

cin >> numberOfPersonal;

if (numberOfPersonal > counterManagers) {

cout << "This number is not in the database." << endl;

}

else {

newFactory.deleteById(numberOfPersonal, PersonalPost::PersonalPost::MANAGER);

}

}

else {

cout << "1)Only 1 - 4!" << endl;

cout << "2)Number selection error, please try again" << endl;

}

break;

}

cout << "Exit - 0" << endl;

cout << "Info - 1" << endl;

cout << "Proceaed - 2" << endl;

cin >> numOfChoiseE;

} while (numOfChoiseE != 0 && numOfChoiseE != 1);

if (numOfChoiseE == 1) {

cout << endl;

cout << "Total Info: " << endl;

cout << endl;

cout << newFactory.getNameOfFactory() << endl;

cout << endl;

cout << "Payouts: " << endl;

try {

cout << newFactory.getPayouts() << endl;

cout << endl;

cout << "Statistic-Info About Personal: " << endl;

newFactory.printInfoAboutMinMaxSalary();

cout << endl;

cout << "Total-Info About Personal" << endl;

newFactory.printInfoAboutPersonal();

}

catch (Exception& e) {

cout << e.getInfo() << endl;

}

cout << endl;

cout << "The number of people employed in each type of position: " << endl;

cout << "Statistic-Info About Count of Personal: " << endl;

newFactory.printInfoAboutPersonalByPost();

};

cout << endl;

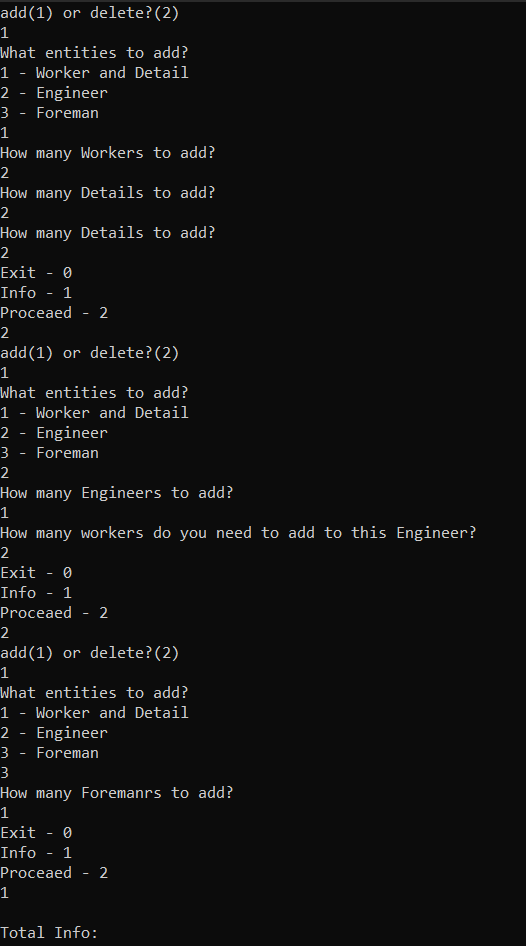
cout << endl;

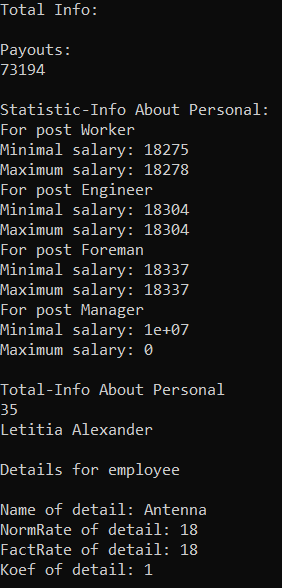
std::cout << "End of Test. URA COMRADS!\n";

return 0;

}

**Результаты работы программы:**

****

****

**Вывод:** Я освоил исключения, контейнерные классы и итераторы STL.