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 №4

по дисциплине «Объектно-ориентированное программирование»

на тему «Полиморфизм, абстрактные классы»

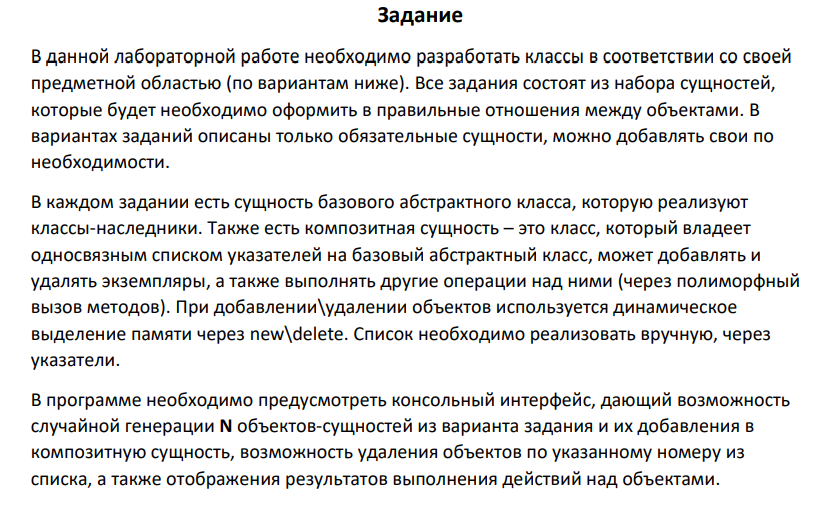
Выполнил: студент гр.1413

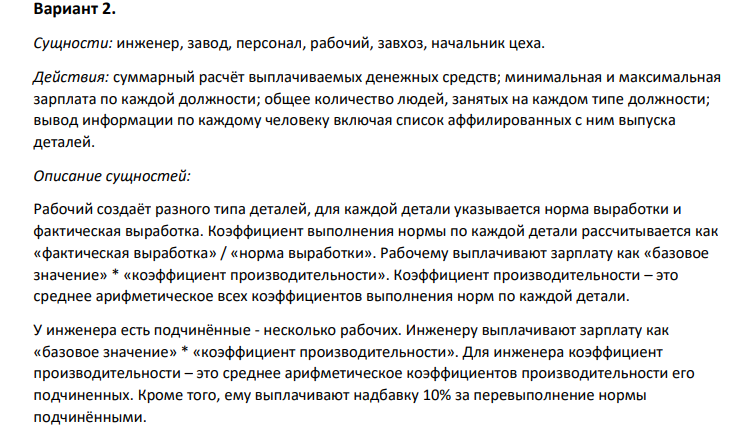
Аверкин М.Е.

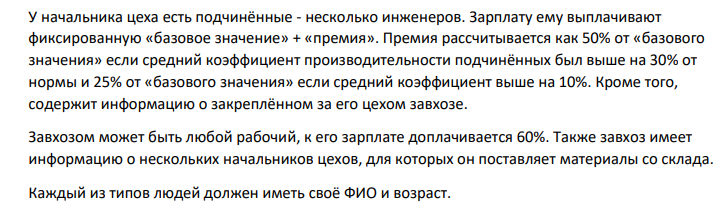
Проверил:

доц. Антипов О.В.

**Цель работы**: освоить реализацию принципа полиморфизма, поработать с абстрактными классами. Выстроить правильную иерархию наследования.

**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, getBaseValue, getAverageKoef и calculateSalary, которые будут переопределяться в классах-наследниках.

Поля: pricate: int \_BaseValue;

int \_age;

char\* \_fio = nullptr;

const char\* \_post = nullptr;

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).

Поля: private: char\* \_name = nullptr;

int \_numberOfDetails;

public: int normRate;

float factRate;

float koef;

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.

Поля: private: Worker\*\* listOfWorkers = nullptr;

int numberOfWorkers;

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.

Поля: private: Engineer\*\* listOfEngineers = nullptr;

Manager\* manager = nullptr;

int numberOfEngineers;

Manager – класс – наследник от класса Worker. 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.

Поля: private: Engineer\*\* listOfEngineers = nullptr;

Manager\* manager = nullptr;

int numberOfEngineers;

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.

Поля: private: Detail\*\* listOfDetails;

int numberOfDetails;

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

Поля: private: Personal\*\* listOfPersonals;

char\* \_name = nullptr;

int capacityOfPersonals;

**Листинг программы:**

**Factory.h**

#pragma once

#include "Personal.h"

#include "Worker.h"

#include "Engineer.h"

#include "GeneralOfDepartment.h"

#include "Manager.h"

class Factory{

private:

Personal\*\* listOfPersonals;

char\* \_name = nullptr;

int capacityOfPersonals;

public:

~Factory();

Factory(const char\* name);

void extendListOfPersonals(int capacity);

void addNewPersonal(Personal& personal);

float getPayouts();

void printInfoAboutMinMaxSalary();

void printInfoAboutPersonal();

void printInfoAboutPersonalByPost();

void deleteById(int id);

const char\* getNameOfFactory() {

return \_name;

}

int getCapacityOfPersonals(){

return capacityOfPersonals;

}

};

**Factory.cpp**

#include "Factory.h"

#include "Personal.h"

Factory::~Factory(){

for (size\_t i = 0; i < capacityOfPersonals; i++) {

delete listOfPersonals[i];

}

delete[] listOfPersonals;

delete[] \_name;

cout << "Factory dtor" << endl;

}

Factory::Factory(const char\* name) {

size\_t len = strlen(name) + 1;

\_name = new char[len];

strcpy\_s(\_name, len, name);

}

float Factory::getPayouts(){

float summPayout = 0.0f;

for (size\_t i = 0; i < capacityOfPersonals; i++) {

summPayout += listOfPersonals[i]->calculateSalary();

}

return summPayout;

}

void Factory::printInfoAboutMinMaxSalary()

{

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

cout << "For post " << arrayOfPosts[i] << endl;

float minSalary = 10000000;

for (int j = 0; j < capacityOfPersonals; j++) {

if (listOfPersonals[j]->getPost() == arrayOfPosts[i]) {

if (listOfPersonals[j]->calculateSalary() < minSalary) {

minSalary = listOfPersonals[j]->calculateSalary();

}

}

}

if (minSalary == 10000000){

minSalary = 0;

}

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

float maxSalary = 0;

for (int j = 0; j < capacityOfPersonals; j++) {

if (listOfPersonals[j]->getPost() == arrayOfPosts[i]) {

if (listOfPersonals[j]->calculateSalary() > maxSalary) {

maxSalary = listOfPersonals[j]->calculateSalary();

}

}

}

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

}

}

void Factory::printInfoAboutPersonal()

{

for (int i = 0; i < capacityOfPersonals; i++) {

listOfPersonals[i]->printInfoAboutPersonal();

}

}

void Factory::printInfoAboutPersonalByPost()

{

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

int totalCounter = 0;

for (int j = 0; j < capacityOfPersonals; j++) {

if (arrayOfPosts[i] == listOfPersonals[j]->getPost()) {

totalCounter++;

}

}

cout << "Total number of personal on post " << arrayOfPosts[i] << " : " << totalCounter << endl;

}

}

void Factory::deleteById(int id)

{

Personal\*\* newListOfPersonals = new Personal \* [capacityOfPersonals - 1];

for (int i = 0; i < id ; i++) {

newListOfPersonals[i] = listOfPersonals[i];

}

for (int j = id + 1; j < capacityOfPersonals; j++) {

newListOfPersonals[j-1] = listOfPersonals[j];

}

delete[] listOfPersonals;

listOfPersonals = newListOfPersonals;

capacityOfPersonals--;

}

void Factory::extendListOfPersonals(int capacity)

{

if (capacity <= capacityOfPersonals) {

return;

}

else {

Personal\*\* newListOfPersonals = new Personal \* [capacity];

for (int i = 0; i < capacityOfPersonals; i++) {

newListOfPersonals[i] = listOfPersonals[i];

}

delete[] listOfPersonals;

listOfPersonals = newListOfPersonals;

}

}

void Factory::addNewPersonal(Personal& personal)

{

this->extendListOfPersonals(capacityOfPersonals + 1);

listOfPersonals[capacityOfPersonals] = &personal;

capacityOfPersonals++;

}

**Personal.h**

#pragma once

#include <stdlib.h>

#include "Detail.h"

using namespace std;

static const char\* arrayOfPosts[]

{ "Default",

"Worker",

"Engineer",

"Foreman",

"Manager"

};

enum posts {

DEFAULT,

WORKER,

ENGINEER,

FOREMAN,

MANAGER

};

class Personal

{

private:

int \_BaseValue;

int \_age;

char\* \_fio = nullptr;

const char\* \_post = nullptr;

public:

virtual void printInfoAboutPersonal() const;

virtual ~Personal();

virtual float getBaseValue() const;

virtual float getAverageKoef() const = 0;

virtual const char\* getPost() const;

void setPost(const char\* post){

this->\_post = post;

}

virtual float calculateSalary() const;

Personal(const char\* fio, int age, int BaseValue);

char\* getFio() {

return \_fio;

}

void setBaseValue(int BaseValue) {

this->\_BaseValue = BaseValue;

}

int getAge() {

return \_age;

}

void setAge(int age) {

this->\_age = age;

}

};

**Personal.cpp**

#include "Personal.h"

void Personal::printInfoAboutPersonal() const

{

cout << "Job title: " << \_post << endl;

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

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

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

cout << endl;

}

Personal::~Personal()

{

delete[] \_fio;

cout << "Personal dtor" << endl;

}

float Personal::getBaseValue() const

{

return \_BaseValue;

}

const char\* Personal::getPost() const

{

return \_post;

}

float Personal::calculateSalary() const{

return getAverageKoef() \* getBaseValue();

}

Personal::Personal(const char\* fio, int age, int BaseValue)

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

{

size\_t len = strlen(fio) + 1;

\_fio = new char[len];

strcpy\_s(\_fio, len, fio);

this->\_post = arrayOfPosts[DEFAULT];

}

**Worker.h**

#pragma once

#include "Detail.h"

#include "Personal.h"

class Worker:public Personal

{private:

Detail\*\* listOfDetails;

int numberOfDetails;

public:

~Worker() override;

Worker(const char\* name, int age, int BaseValue);

void extendListOfDetails(int capacity);

void addNewDetail(Detail& detail);

Detail\*\* getListOfDetails() {

return listOfDetails;

};

float getAverageKoef() const override;

void printInfoAboutPersonal() const override;

};

**Worker.cpp**

#include "Worker.h"

Worker::~Worker(){

for(size\_t i = 0; i < numberOfDetails; i++) {

delete listOfDetails[i];

}

delete[] listOfDetails;

cout << "Worker dtor" << endl;

}

Worker::Worker(const char\* fio, int age, int BaseValue)

:Personal(fio, age, BaseValue){

this->setPost(arrayOfPosts[WORKER]);

numberOfDetails = 0;

}

void Worker::extendListOfDetails(int capacity){

if (capacity <= numberOfDetails) {

return;

}

else {

Detail\*\* newListOfDetails = new Detail \* [capacity];

for (int i = 0; i < numberOfDetails; i++) {

newListOfDetails[i] = listOfDetails[i];

}

delete[] listOfDetails;

listOfDetails = newListOfDetails;

}

}

void Worker::addNewDetail(Detail& detail){

this->extendListOfDetails(numberOfDetails + 1);

this->listOfDetails[numberOfDetails] = &detail;

numberOfDetails++;

}

float Worker::getAverageKoef() const {

float averageKoef = 0.0f;

float summaryKoef = 0.0f;

for (int i = 0; i < numberOfDetails; i++) {

summaryKoef += listOfDetails[i]->getKoef();

}

averageKoef = summaryKoef / numberOfDetails;

return averageKoef;

}

void Worker::printInfoAboutPersonal() const{

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

Personal::printInfoAboutPersonal();

for (int i = 0; i < numberOfDetails; i++) {

listOfDetails[i]->toString();

}

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

cout << endl;

}

**Detail.h**

#pragma once

#include <stdlib.h>

#include <string>

#include <iostream>

using namespace std;

class Detail

{

private:

char\* \_name = nullptr;

int \_numberOfDetails;

public:

int normRate;

float factRate;

float koef;

Detail(const char\* name, int normRate, float factRate);

~Detail();

const char\* getName() {

return \_name;

}

void toString() {

cout << endl;

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(char\* 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 char\* name, int normRate, float factRate) {

size\_t len = strlen(name) + 1;

\_name = new char[len];

strcpy\_s(\_name, len, name);

this->normRate = normRate;

this->factRate = factRate;

this->koef = factRate / normRate;

}

Detail::~Detail()

{

delete[] \_name;

cout << "Detail dtor" << endl;

}

**Engineer.h**

#pragma once

#include "Worker.h"

class Engineer:public Personal

{

private:

Worker\*\* listOfWorkers = nullptr;

int numberOfWorkers;

public:

~Engineer() override;

Engineer(const char\* fio, int age, int BaseValue);

Worker\*\* getListOfWorkers() {

return listOfWorkers;

};

void extendListOfWorkers(int capacity);

void addNewWorker(Worker& worker);

int getNumberOfWorkers() {

return numberOfWorkers;

}

float calculateSalary() const override;

void printInfoAboutPersonal() const override;

float getAverageKoef() const override;

};

**Engineer.cpp**

#include "Engineer.h"

Engineer::~Engineer(){

delete[] listOfWorkers;

cout << "Engineer dtor" << endl;

}

Engineer::Engineer(const char\* fio, int age, int BaseValue)

:Personal(fio, age, BaseValue) {

this->setPost(arrayOfPosts[ENGINEER]);

numberOfWorkers = 0;

}

void Engineer::extendListOfWorkers(int capacity)

{

if (capacity <= numberOfWorkers) {

return;

}

else {

Worker\*\* newListOfWorkers = new Worker \* [capacity];

for (int i = 0; i < numberOfWorkers; i++) {

newListOfWorkers[i] = listOfWorkers[i];

}

delete[] listOfWorkers;

listOfWorkers = newListOfWorkers;

}

}

void Engineer::addNewWorker(Worker& worker)

{

this->extendListOfWorkers(numberOfWorkers + 1);

this->listOfWorkers[numberOfWorkers] = &worker;

numberOfWorkers++;

}

void Engineer::printInfoAboutPersonal() const{

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

Personal::printInfoAboutPersonal();

for (int i = 0; i < numberOfWorkers; i++) {

listOfWorkers[i]->Personal::printInfoAboutPersonal();

}

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

cout << endl;

}

float Engineer::getAverageKoef() const{

float averageKoef = 0.0f;

float summaryKoef = 0.0f;

for (int i = 0; i < numberOfWorkers; i++) {

summaryKoef += listOfWorkers[i]->getAverageKoef();

}

averageKoef = summaryKoef / numberOfWorkers;

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:

Engineer\*\* listOfEngineers = nullptr;

Manager\* manager = nullptr;

int numberOfEngineers;

public:

~Foreman() override;

Foreman(const char\* fio, int age, int BaseValue);

Engineer\*\* getListOfEngineers() {

return listOfEngineers;

}

int getNumberOfEngineers() {

return numberOfEngineers;

}

void setManager(Manager\* manager) {

this->manager = manager;

}

void extendListOfEngineers(int capacity);

void addNewEngineer(Engineer& engineer);

float calculateSalary() const override;

void printInfoAboutPersonal() const override;

float getAverageKoef() const override;

};

**GeneralOfDepartment.cpp**

#include "GeneralOfDepartment.h"

#include "Manager.h"

Foreman::~Foreman()

{

delete[] listOfEngineers;

cout << "Foreman dtor" << endl;

}

Foreman::Foreman(const char\* fio, int age, int BaseValue)

:Personal(fio, age, BaseValue) {

this->setPost(arrayOfPosts[FOREMAN]);

numberOfEngineers = 0;

}

void Foreman::printInfoAboutPersonal() const{

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

Personal::printInfoAboutPersonal();

for (int i = 0; i < numberOfEngineers; i++) {

listOfEngineers[i]->Personal::printInfoAboutPersonal();

}

manager -> Personal::printInfoAboutPersonal();

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

cout << endl;

}

void Foreman::extendListOfEngineers(int capacity)

{

if (capacity <= numberOfEngineers) {

return;

}

else {

Engineer\*\* newListOfEngineer = new Engineer \* [capacity];

for (int i = 0; i < numberOfEngineers; i++) {

newListOfEngineer[i] = listOfEngineers[i];

}

delete[] listOfEngineers;

listOfEngineers = newListOfEngineer;

}

}

void Foreman::addNewEngineer(Engineer& engineer)

{

this->extendListOfEngineers(numberOfEngineers + 1);

this->listOfEngineers[numberOfEngineers] = &engineer;

numberOfEngineers++;

}

float Foreman::getAverageKoef() const {

float averageKoef = 0.0f;

float summaryKoef = 0.0f;

for (int i = 0; i < numberOfEngineers; i++) {

summaryKoef += listOfEngineers[i]->getAverageKoef();

}

averageKoef = summaryKoef / numberOfEngineers;

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:

Foreman\*\* listOfForemans = nullptr;

int numberOfForemans;

public:

~Manager() override;

Manager(const char\* fio, int age, int BaseValue)

:Worker(fio, age, BaseValue) {

this->setPost(arrayOfPosts[MANAGER]);

numberOfForemans = 0;

}

Foreman\*\* getListOfForemans() {

return listOfForemans;

}

void extendListOfForemans(int capacity);

void addNewForeman(Foreman& foreman);

int getNumberOfForemans() {

return numberOfForemans;

}

float calculateSalary() const override;

void printInfoAboutPersonal() const override;

float getAverageKoef()const override;

};

**Manager.cpp**

#include "Manager.h"

#include "GeneralOfDepartment.h"

Manager::~Manager(){

delete[] listOfForemans;

cout << "Manager dtor" << endl;

}

void Manager::extendListOfForemans(int capacity){

if (capacity <= numberOfForemans) {

return;

}

else {

Foreman\*\* newListOfForemans = new Foreman \* [capacity];

for (int i = 0; i < numberOfForemans; i++) {

newListOfForemans[i] = listOfForemans[i];

}

delete[] listOfForemans;

listOfForemans = newListOfForemans;

}

}

void Manager::addNewForeman(Foreman& foreman){

this->extendListOfForemans(numberOfForemans + 1);

this->listOfForemans[numberOfForemans] = &foreman;

numberOfForemans++;

}

void Manager::printInfoAboutPersonal() const {

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

Personal::printInfoAboutPersonal();

for (int i = 0; i < numberOfForemans; i++) {

listOfForemans[i]->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>

using namespace std;

static const char\* ranFackName[10] = {

"Microelectronics",

"LabaevArms",

"RadioEngineering",

"Avto",

"RedFlag",

"Automobile",

"Millitary",

"Metell",

"Litie",

"RanNaMe"

};

static const char\* ranDetailName[10] = {

"Microelectronic",

"Plata",

"Software",

"Toyota",

"Barrel",

"Gun",

"RanDetAIl",

"Transistor",

"Antenna",

"Refrezeraitor"

};

static const char\* 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 char\* 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 char\* getFackName() {

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

const char\* fackName = ranFackName[i];

return fackName;

}

static const char\* getRanFIO() {

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

static char FIO[256];

const char\* name = ranName[i];

const char\* surname = ranSurname[i];

sprintf\_s(FIO, "%s %s", 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 char\* getDetailName() {

int i = rand() % (10);

const char\* detailName = ranDetailName[i];

return detailName;

}

};

**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());

Worker\* arrayOfWorkers[150]{};

Engineer\* arrayOfEngineer[150]{};

Detail\* arrayOfDetails[750]{};

Foreman\* arrayOfForemans[75]{};

int counterWorkers = 0;

int counterEngineers = 0;

int counterForemans = 0;

int counterManagers = 0;

int AddedDetail = -1;

int AddedWorker = -1;

int AddedEngineer = -1;

int AddedForeman = -1;

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 && counterWorkers < 150 && AddedDetail < 750) {

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

cin >> N;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> N;

}

while (N + counterWorkers > 150);

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

Worker& newWorker = \*(new Worker(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary()));

newFactory.addNewPersonal(newWorker);

int numberOfDetailsForWorker = 0;

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

cin >> numberOfDetailsForWorker;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> numberOfDetailsForWorker;

} while (numberOfDetailsForWorker + AddedDetail > 750);

for (int j = 0; j < numberOfDetailsForWorker; j++) {

Detail& newDetail = \*(new Detail(Randomization::getDetailName(), Randomization::getRanK1(), Randomization::getRanK2()));

AddedDetail++;

arrayOfDetails[AddedDetail] = &newDetail;

newWorker.addNewDetail(\*arrayOfDetails[AddedDetail]);

}

arrayOfWorkers[counterWorkers++] = &newWorker;

}

}

else if (numOfChoise == 2 && counterEngineers < 150 && AddedWorker < 150) {

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

cin >> N;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> N;

} while (N + counterEngineers > 150);

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

Engineer& newEngineer = \*(new Engineer(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary()));

newFactory.addNewPersonal(newEngineer);

int numberOfWorkersForEngineer = 0;

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

cin >> numberOfWorkersForEngineer;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> numberOfWorkersForEngineer;

} while (numberOfWorkersForEngineer + AddedWorker > 150);

for (int j = 0; j < numberOfWorkersForEngineer; j++) {

AddedWorker++;

newEngineer.addNewWorker(\*arrayOfWorkers[AddedWorker]);

}

arrayOfEngineer[counterEngineers++] = &newEngineer;

}

}

else if (numOfChoise == 3 && counterForemans < 75 && AddedEngineer < 150) {

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

cin >> N;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> N;

} while (N + counterForemans > 75);

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

Foreman& newForeman = \*(new Foreman(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary()));

newFactory.addNewPersonal(newForeman);

int numberOfEngineersForForeman = 0;

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

cin >> numberOfEngineersForForeman;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> numberOfEngineersForForeman;

} while (numberOfEngineersForForeman + AddedEngineer > 150);

for (int j = 0; j < numberOfEngineersForForeman; j++) {

AddedEngineer++;

newForeman.addNewEngineer(\*arrayOfEngineer[AddedEngineer]);

}

arrayOfForemans[counterForemans++] = &newForeman;

}

}

else if (numOfChoise == 4 && counterManagers < 75 && AddedForeman < 150) {

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

cin >> N;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> N;

} while (N + counterManagers > 75);

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

Manager& newManager = \*(new Manager(Randomization::getRanFIO(), Randomization::getRanAge(), Randomization::getRanSalary()));

newFactory.addNewPersonal(newManager);

counterManagers++;

int numberOfForemansForManager = 0;

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

cin >> numberOfForemansForManager;

do {

cout << "The database is full, please enter a lower number: " << endl;

cin >> numberOfForemansForManager;

} while (numberOfForemansForManager + AddedForeman > 75);

for (int j = 0; j < numberOfForemansForManager; j++) {

AddedForeman++;

newManager.addNewForeman(\*arrayOfForemans[AddedForeman]);

arrayOfForemans[AddedForeman]->setManager(&newManager);

}

}

}

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 (AddedWorker+1 < numberOfPersonal) {

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

}

else {

newFactory.deleteById(numberOfPersonal - 1);

}

}

else if (numOfChoise == 2) {

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

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

cin >> numberOfPersonal;

if (AddedEngineer+1 < numberOfPersonal) {

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

}

else {

numberOfPersonal = numberOfPersonal + counterWorkers;

newFactory.deleteById(numberOfPersonal - 1);

}

}

else if (numOfChoise == 3) {

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

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

cin >> numberOfPersonal;

if (AddedForeman+1 < numberOfPersonal) {

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

}

else {

numberOfPersonal = numberOfPersonal + counterWorkers + counterEngineers;

newFactory.deleteById(numberOfPersonal - 1);

}

}

else if (numOfChoise == 4) {

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

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

cin >> numberOfPersonal;

if (counterManagers < numberOfPersonal) {

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

}

else {

numberOfPersonal = numberOfPersonal + counterWorkers + counterEngineers + counterForemans;

newFactory.deleteById(numberOfPersonal - 1);

}

}

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;

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

cout << endl;

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

newFactory.printInfoAboutMinMaxSalary();

cout << endl;

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

newFactory.printInfoAboutPersonal();

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;

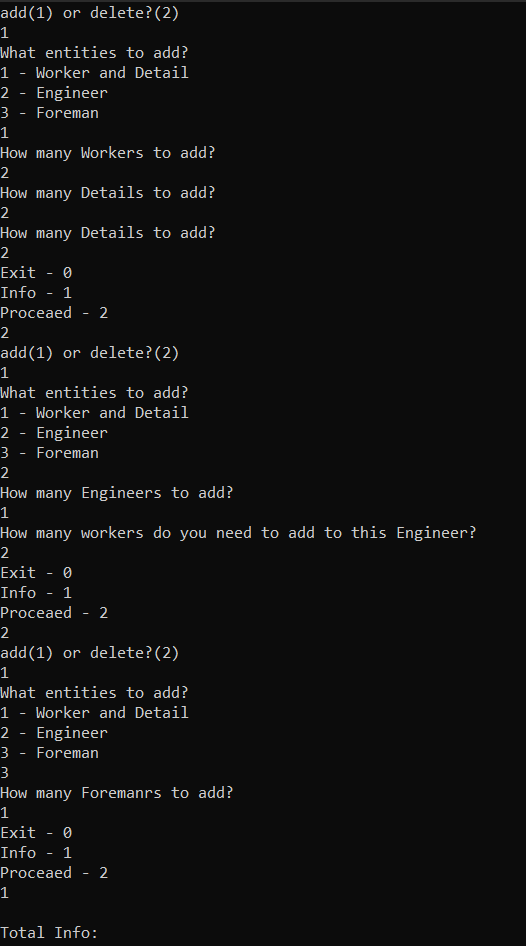
cout << endl;

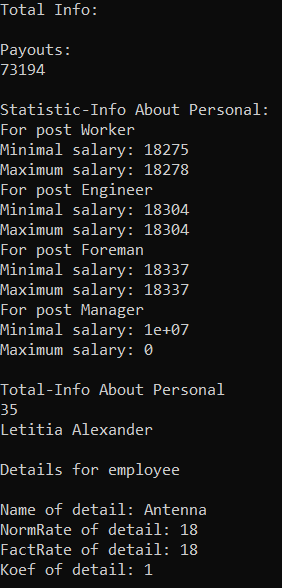
std::cout << "End of Test. URA COMRADS!\n";

return 0;

}

**Результаты работы программы:**

****

****

**Вывод:** Я освоил принципы полиморфизма, научился строить иерархию классов.