ИСТ-АДМО-202Б

Трубин В.Д.

Орлов К.О.

3241.506515.000 ПЗ

Геоинформационных систем

Языки программирования

Уфа – 2025

Виртуальные функции

Содержание

[Введение 3](#_Toc198795382)

[1 Создание программы 4](#_Toc198795383)

[Заключение 9](#_Toc198795384)

[Список литературы 10](#_Toc198795385)

Введение

Цель работы – изучение принципов наследования в объектно-ориентированной парадигме программирования

# Создание программы

В приведённой программе на C++ реализована иерархия классов с базовым классом Person и производным классом Student. Одним из ключевых элементов в этой иерархии является использование **виртуальной функции** printInfo() в базовом классе.

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <map>

#include <cmath>

using namespace std;

// Базовый класс Person

class Person {

protected:

string fullName;

public:

Person() = default;

Person(const string& name) : fullName(name) {}

virtual ~Person() = default;

string getFullName() const { return fullName; }

void setFullName(const string& name) { fullName = name; }

virtual void printInfo() const {

cout << "Name: " << fullName << endl;

}

};

// Дочерний класс Student

class Student : public Person {

private:

int\* marks;

int subjectCount;

public:

Student(int subjectCount) : subjectCount(subjectCount) {

marks = new int[subjectCount];

}

~Student() {

delete[] marks;

}

void setMarks(const int\* inputMarks) {

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

marks[i] = inputMarks[i];

}

}

void setMark(int index, int value) {

if (index >= 0 && index < subjectCount)

marks[index] = value;

}

int getSubjectCount() const { return subjectCount; }

double average() const {

int sum = 0;

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

sum += marks[i];

return static\_cast<double>(sum) / subjectCount;

}

// Перегрузка оператора ~

double operator~() const {

return average();

}

// Перегрузка оператора % — сравнение среднего балла

friend double operator%(const Student& a, const Student& b) {

return fabs((~a) - (~b));

}

void printInfo() const override {

cout << fullName << ": ";

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

cout << marks[i] << " ";

cout << "| Average score: " << average() << "\n";

}

};

// Класс обработки результатов экзаменов

class ExamResults {

private:

string\* subjects;

Student\*\* students;

int subjectCount;

int studentCount;

int markToNumeric(const string& mark) {

if (mark == "unsatisfactory") return 2;

if (mark == "satisfactory") return 3;

if (mark == "good") return 4;

if (mark == "excellent") return 5;

return 0;

}

public:

ExamResults() = default;

~ExamResults() {

delete[] subjects;

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

delete students[i];

delete[] students;

}

void readFromFile(const string& filename) {

ifstream file(filename);

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

cerr << "Unable to open file: " << filename << endl;

return;

}

string line;

// Считываем предметы

if (getline(file, line)) {

istringstream stream(line);

string tmp;

subjectCount = 0;

while (stream >> tmp) ++subjectCount;

subjects = new string[subjectCount];

stream.clear();

stream.str(line);

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

stream >> subjects[i];

}

// Подготовка массива студентов

int capacity = 100;

students = new Student \* [capacity];

studentCount = 0;

while (getline(file, line)) {

istringstream stream(line);

string name;

stream >> name;

if (name.empty()) continue;

Student\* student = new Student(subjectCount);

student->setFullName(name);

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

string mark;

if (stream >> mark)

student->setMark(i, markToNumeric(mark));

else

student->setMark(i, 0);

}

if (studentCount < capacity)

students[studentCount++] = student;

else

delete student;

}

file.close();

}

void printResults() const {

if (subjectCount == 0 || studentCount == 0) {

cout << "No data available.\n";

return;

}

cout << "Subjects: ";

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

cout << subjects[i] << " ";

cout << "\n";

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

students[i]->printInfo();

}

void writeToFile(const string& filename) const {

ofstream file(filename);

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

cerr << "Unable to open file for writing: " << filename << endl;

return;

}

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

file << subjects[i] << " ";

file << endl;

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

file << students[i]->getFullName() << " ";

for (int j = 0; j < subjectCount; ++j)

file << students[i]->average() << " ";

file << endl;

}

file.close();

}

void compareResults(const ExamResults& other) const {

cout << "\nComparison of results for the current and previous year:\n";

map<string, double> currentResults, previousResults;

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

currentResults[students[i]->getFullName()] = ~(\*students[i]);

for (int i = 0; i < other.studentCount; ++i)

previousResults[other.students[i]->getFullName()] = ~(\*other.students[i]);

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

string name = students[i]->getFullName();

double currentAvg = currentResults[name];

cout << "Student: " << name << "\n";

if (previousResults.find(name) != previousResults.end()) {

double prevAvg = previousResults[name];

double deviation = fabs(currentAvg - prevAvg);

cout << " Current year: " << currentAvg << "\n";

cout << " Previous year: " << prevAvg << "\n";

cout << " Deviation in average score: " << deviation << "\n";

}

else {

cout << " No data for the previous year.\n";

}

}

}

};

// Точка входа

int main() {

ExamResults currentYearResults;

ExamResults lastYearResults;

currentYearResults.readFromFile("current\_year.txt");

lastYearResults.readFromFile("last\_year.txt");

cout << "\nCurrent year results:\n";

currentYearResults.printResults();

cout << "\nPrevious year results:\n";

lastYearResults.printResults();

currentYearResults.compareResults(lastYearResults);

currentYearResults.writeToFile("output\_current\_year.txt");

lastYearResults.writeToFile("output\_last\_year.txt");

return 0;

}

Заключение

В результате работы была улучшена архитектура программы. Введена иерархия классов, использован принцип наследования. Программа стала более гибкой и готовой к дальнейшему расширению. Такие подходы являются основой качественного объектно-ориентированного проектирования.

Список литературы

1. Кубенский, А. А. Функциональное программирование : учебник и практикум для академического бакалавриата / А. А. Кубенский. — М. : Издательство Юрайт, 2019. — 348 с.
2. Трофимов, В. В. Основы алгоритмизации и программирования : учебник для СПО / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. — М. : Издательство Юрайт, 2019. — 137 с