Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Факультет электротехнический

Кафедра ИТАС

ОТЧЁТ

о лабораторной работе №4 по классам

Выполнил:
Студент группы ИВТ-23-1Б
Пискунов Д. А.

_____

Проверил:
Доцент кафедры ИТАС
Яруллин Д.В.

_____

_____

Пермь 2024

Вариант 15

Базовый класс:
ЧЕЛОВЕК (PERSON)
Имя (name) – string
Возраст (age) – int
Определить методы изменения полей.
Создать производный класс STUDENT, имеющий поля Предмет – string и Оценка – int. Определить методы изменения полей и метод, выдающий сообщение о неудовлетворительной оценке.

Текст программы
Файл Lab_Main.cpp

```cpp
#include <iostream>
#include"Person.h"
#include "Student.h"

 using namespace std;

void f1(Person& p) {
    p.set_name("Толик\n");
}
Person f2() {
    Student a("Артем", 18, "Физическая культура и спорт", 2);

    return a;
}

int main() {
    system("chcp 1251>null");

    Person a;
    cin >> a;
    cout << a;
    cout << endl;

    Person b("Саня", 45);
    cout << b;
    cout << endl;

    a = b;
    cout << a;
    cout << endl;
    cout << endl;

    Student c;
    cin >> c;
    cout << c;
    cout << endl;


    f1(c);
    a = f2();
    cout << a;
    cout << endl;


    return 0;
}
```

Файл Person.h

```h
#pragma once
#include<iostream>

using namespace std;

class Person {
protected:
    string name;
    int age;
public:
    Person();
    Person(string, int);
    Person(const Person&);
    virtual ~Person(void);
    string get_name() { return name; };
    int get_age() { return age; }

    void set_name(string);
    void set_age(int);

    Person& operator=(const Person&);

    friend istream& operator >>(istream& in, Person& p);
    friend ostream& operator <<(ostream& out, const Person& p);


};

```

Файл Person.cpp

```cpp
#include "Person.h"

Person::Person(void) {
    name = "";
    age = 0;
}
Person::Person(string n, int a) {
    name = n;
    age = a;
}
Person::Person(const Person& p) {
    name = p.name;
    age = p.age;
}
 Person::~Person(){}
void Person::set_name(string n) {
    name = n;
}
void Person::set_age(int a) {
    age = a;
}

Person& Person::operator=(const Person& p) {
    if (this == &p) {
        return *this;
    }
    name = p.name;
    age = p.age;
    return *this;
}

istream& operator >>(istream& in, Person& p) {
    cout << "name: "; in >> p.name;
    cout << "age: "; in >> p.age;
    return in;
}
ostream& operator <<(ostream& out, const Person& p) {
    out << "\nname: " << p.name;
    out << "\nage: " << p.age << " y.o.";
    return out;
}
```

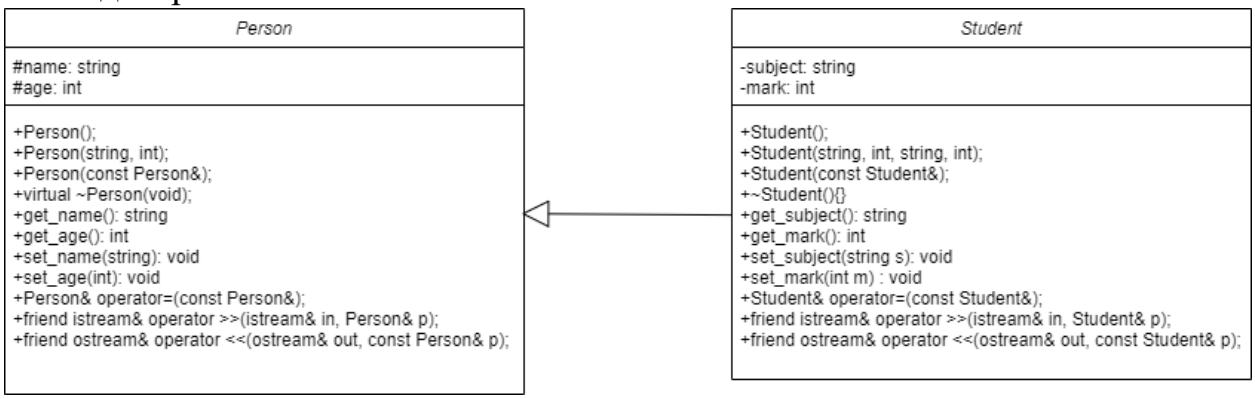
Файл Student.h

```h
#pragma once
#include "Person.h"

class Student : public Person {
private:
    string subject;
    int mark;
public:
    Student();
    Student(string, int, string, int);
    Student(const Student&);
    ~Student(){}

    string get_subject() { return subject; }
    int get_mark() { return mark; }

    void set_subject(string s) { subject = s; }
    void set_mark(int m) { mark = m; }

    Student& operator=(const Student&);

    friend istream& operator >>(istream& in, Student& p);
    friend ostream& operator <<(ostream& out, const Student& p);


};

```

Файл Studen.cpp

```cpp
#include"Student.h"

Student::Student():Person(){
    subject = "";
    mark = 0;
}
Student::Student(string n, int a, string s, int m):Person(n,a) {
    subject = s;
    mark = m;
}
Student::Student(const Student& s) {
    name = s.name;
    age = s.age;
    subject = s.subject;
    mark = s.mark;
}

Student& Student::operator=(const Student& s) {
    if (&s == this) {
        return* this;
    }
    name = s.name;
    age = s.age;
    subject = s.subject;
    mark = s.mark;
    return *this;
}

istream& operator >>(istream& in, Student& p) {
    cout << "name: "; in >> p.name;
    cout << "age: "; in >> p.age;
    cout << "subject: "; in >> p.subject;
    cout << "mark: "; in >> p.mark;
    return in;
}
ostream& operator <<(ostream& out, const Student& p) {
    out << "\nname: " << p.name;
    out << "\nage: " << p.age <<" y.o.";
    out << "\nsubject: " << p.subject;
    out << "\nmark: " << p.mark;
    if (p.mark <= 2) {
        out << "\nНа кол двоечника";
    }

    return out;
}
```

UML-диаграмма



Тесты

```
name: Саша
age: 7

name: Саша
age: 7 y.o.

name: Саня
age: 45 y.o.

name: Саня
age: 45 y.o.

name: Дима
age:
12
subject: math
mark: 2

name: Дима
age: 12 y.o.
subject: math
mark: 2
На кол двоечника

name: Артем
age: 18 y.o.

C:\Users\MOkASiH\Desktop\Lab_Main\x64\Debug\Lab_Main.exe (процесс 6424) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно…
```

Ответы на вопросы

1. Для чего используется механизм наследования?

Ответ: для создания новых классов на основе уже существующих.

2. Каким образом наследуются компоненты класса, описанные со спецификатором public?

Ответ: наследуются открыто и доступны для использования в производном классе.

3. Каким образом наследуются компоненты класса, описанные со спецификатором private?

Ответ: не наследуются.

4. Каким образом наследуются компоненты класса, описанные со спецификатором protected?

Ответ: наследуются защищенно и доступны для использования в производном классе и его потомках.

5. Каким образом описывается производный класс?

Ответ: class Название_Производного_Класса : Тип_Наследуеммых_Данных Название_Класса_Родителя.

6. Наследуются ли конструкторы?

Ответ: конструкторы наследуются, но не переопределяются.

7. Наследуются ли деструкторы?

Ответ: деструкторы наследуются, но не переопределяются.

8. В каком порядке конструируются объекты производных классов?

Ответ: сначала конструируется базовый класс, затем производный класс.

9. В каком порядке уничтожаются объекты производных классов?

Ответ: сначала уничтожается производный класс, затем базовый класс.

10. Что представляют собой виртуальные функции и механизм позднего связывания?

Ответ: виртуальная функция представляет собой функцию, которая может быть в полиморфном состоянии – состоянии, при котором вызов нужной функции из набора виртуальных формируется на этапе *позднего связывания*. Понятие позднее связывание означает, что код вызова нужной функции формируется при выполнении программы.

11. Могут ли быть виртуальными конструкторы? Деструкторы?

Ответ: конструкторы не могут быть виртуальными, деструкторы могут быть виртуальными.

12. Наследуется ли спецификатор virtual?

Ответ: нет.

13. Какое отношение устанавливает между классами открытое наследование?

Ответ: производный класс является расширением базового класса.

14. Какое отношение устанавливает между классами закрытое наследование?

Ответ: производный класс реализует интерфейс базового класса, но не является его расширением.

15. В чем заключается принцип подстановки?

Ответ: объекты производного класса могут использоваться везде, где ожидается объект базового класса, не нарушая при этом корректности программы.

16. Имеется иерархия классов:

```
class Student
{
        int age;
public:
        string name;
        ...
};
class Employee : public Student
{
protected:
        string post;
        ...
};
class Teacher : public Employee
{
protected: int stage;
...
};

Teacher x;
```

Какие компонентные данные будет иметь объект x?

Ответ: объект x будет иметь компоненты данных age, name, post и stage.

17. Для классов Student, Employee и Teacher написать конструкторы без параметров.

Student() : age(0), name("") {}

Employee() : post("") {}

Teacher : stage(0) {}

18. Для классов Student, Employee и Teacher написать конструкторы с параметрами.

```
Student(int a, string n) : age(a), name(n) {}
Employee(int a, string n, string p) : Student(a, n), post(p) {}
Teacher(int a, string n, string p, int s) : Employee(a, n, p), stage(s) {}
```

19. Для классов Student, Employee и Teacher написать конструкторы копирования.

```
Student(const Student& copied) : age(copied.age), name(copied.name) {}
Employee(const Employee7 copied) : Student(copied), post(copied.post) {}
Teacher(const Teacher& copied) : Employee(copied), stage(copied.stage) {}
```

20. Для классов Student, Employee и Teacher определить операцию присваивания.

```
Student& operator=(const Student& s) {
	if (this != &s) {
		age = s.age;
		name = s.name;
	}
	return *this;
}

Employee& operator=(const Employee& e) {
	if (this != &e) {
		Student::operator=(e);
		post = e.post
	}
	return *this;
}

Teacher& operator=(const Teacher& t) {
	if (this != &t) {
		Employee::operator=(t);
		stage = other.stage;
	}
	return *this;
}
```