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

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**Тема:** Лабораторная работа №4 по ООП

Семестр: 2

Выполнил студент ИВТ-23-2б:

Хомутов Сергей Алексеевич

(дата, подпись)

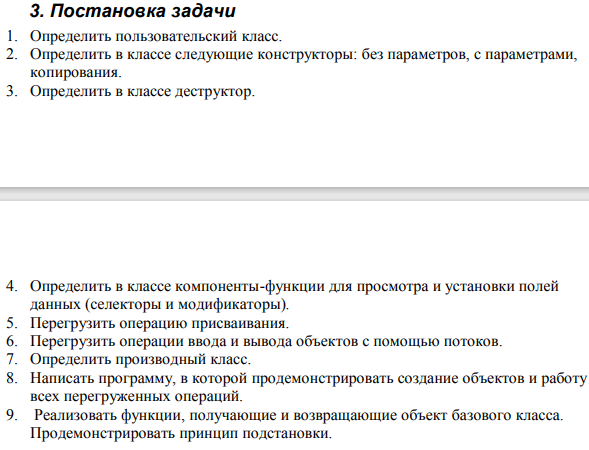
Проверила:

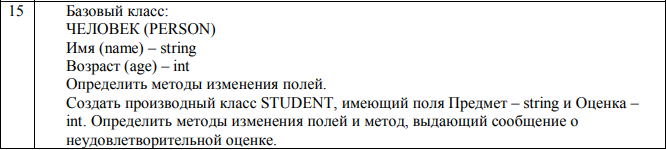
Полякова Ольга Андреевна

(дата, подпись)

Пермь 2024

**Задание**





**Анализ задачи**

**Класс Person:**

1. Person() – конструктор без параметров

2. Person(string name, int age) – конструктор с параметрами

3. Person(const Person& copied) – конструктор копирования

4. istream& operator>>(istream& in, Person& p) – перегрузка оператора ввода

5. ostream& operator<<(ostream& out, const Person& p) – перегрузка оператора вывода

6. void setName(string name) – сеттер для поля имени

7. void setAge(int age) – сеттер для поля возраст

8. string getName() – геттер для поля имя

9. int getAge() – геттер для поля возраст

10. Person& operator=(const Person& m\_person) - перегрузка оператора присваивания

**Класс Student:**

1. Student() – конструктор без параметров

2. Student(string, int, string, int) – конструктор с параметрами

3. Student(const Student&)- конструктор копирования

4. void setSubject(string) – сеттер для поля предмет

5. void setMark(int) – сеттер для поля оценка

6. string getSubject() – геттер для поля предмет

7. int getMark() – геттер для поля оценка

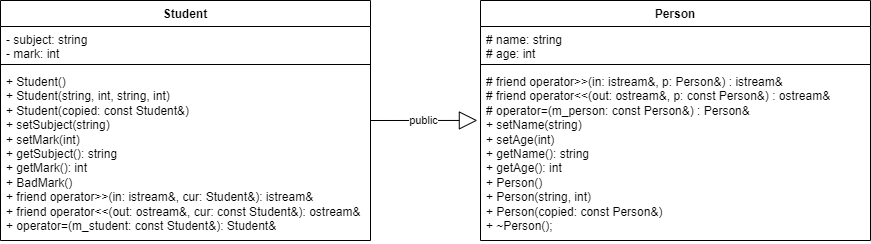
8. void BadMark() – метод выдающий сообщение о неудовлетворяющей оценке

9. friend istream& operator>>(istream&, Student&) – перегрузка оператора ввода

10. friend ostream& operator<<(ostream&, const Student&) – перегрузка оператора вывода

11. Student& operator=(const Student&) – перегрузка оператора присваивания

**UML-Диаграмма**

****

**Программное решение**

**Person.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Person {//Базовый класс

protected:

string name;

int age;

friend istream& operator>>(istream& in, Person& p);

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

Person& operator=(const Person&);

public:

void setName(string);

void setAge(int);

string getName();

int getAge();

Person();

Person(string, int);

Person(const Person&);

~Person() {};

};

**Person.cpp**

#include "Person.h"

Person::Person() {

this->name = "";

this->age = 0;

}

Person::Person(string name, int age) {

this->name = name;

this->age = age;

}

Person::Person(const Person& copied) {

age = copied.age;

name = copied.name;

}

istream& operator>>(istream& in, Person& p) {

cout << "Введите данные в формате:\nимя возвраст\n";

return (in >> p.name >> p.age);

}

ostream& operator<<(ostream& out, const Person& p) {

return (out << p.name << ' ' << p.age << '\n');

}

void Person::setName(string name) {

this->name = name;

}

void Person::setAge(int age) {

this->age = age;

}

string Person::getName() {

return name;

}

int Person::getAge() {

return age;

}

Person& Person::operator=(const Person& m\_person) {//перегрузка оператора присваивания

if (&m\_person != this) {

name = m\_person.name;

age = m\_person.age;

}

return \*this;

}

**Student.h**

#pragma once

#include <iostream>

#include "Person.h"

using namespace std;

class Student : public Person {

private:

string subject;

int mark;

public:

Student();

Student(string, int, string, int);

Student(const Student&);

void setSubject(string);

void setMark(int);

string getSubject();

int getMark();

void BadMark();

friend istream& operator>>(istream&, Student&);

friend ostream& operator<<(ostream&, const Student&);

Student& operator=(const Student&);

};

**Student.cpp**

#include "Student.h"

Student::Student() {

name = "";

subject = "";

age = 0;

mark = 0;

}

Student::Student(string name, int age, string subject, int mark) {

this->name = name;

this->age = age;

this->subject = subject;

this->mark = mark;

}

Student::Student(const Student& copied) {

name = copied.name;

age = copied.age;

subject = copied.subject;

mark = copied.mark;

}

void Student::setSubject(string subject) {

this->subject = subject;

}

void Student::setMark(int mark) {

this->mark = mark;

}

string Student::getSubject() {

return subject;

}

int Student::getMark() {

return mark;

}

void Student::BadMark() {

if (mark < 4) {

cout << this->name << " имеет плохую оценку\n";

}

}

istream& operator>>(istream& in, Student& cur) {

cout << "Введите данные в формате:\nимя возвраст предмет оценка\n";

return (in >> cur.name >> cur.age >> cur.subject >> cur.mark);

}

ostream& operator<<(ostream& out, const Student& cur) {

return (out << cur.name << ' ' << cur.age << ' ' << cur.subject << ' ' << cur.mark << '\n');

}

Student& Student::operator=(const Student& m\_student) {//перегрузка оператора присваивания

if (&m\_student != this) {

name = m\_student.name;

age = m\_student.age;

subject = m\_student.subject;

mark = m\_student.mark;

}

return \*this;

}

**OOPLab4.cpp**

#include "Person.h"

#include "Student.h"

void getObject(Person& object) {//функция принимающая объект

object.setName("Дима");

object.setAge(15);

}

Person ReturnObject() {//функция возвращающая объект

Student s("Антон", 21, "Информатика", 5);

return s;

}

int main() {

system("chcp 1251 > NULL");

Person p1("Миша", 3);//Миша ещё не ученик

cout << p1;//выводим значения

cin >> p1;//вводим новые знаения

cout << p1;//проверяем значения

Student s1("Олег", 23, "inf", 4);//У Олега, как и у любого человека есть имя и возвраст, а как у ученика - предмет и оценка

Student s2;

getObject(s2);//установка значений через функцию

cout << s2;//проверка значений

Person p2 = ReturnObject();//создание объекта через функцию

cout << p2;//проверка значений

Student s3;

cin >> s3;//считываем значений для s3

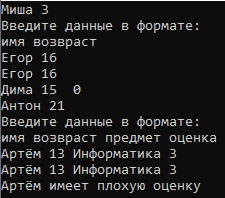
s2 = s3;//заменяем значения s2 соответствующими значениями s3

cout << s2;//проверяем значение

s2.BadMark();//проверяем оценку (выведет сообщение, если плохая)

return 0;

}



**Контрольные вопросы**

1. Для чего используется механизм наследования?

Для создания новых классов на основе уже существующих.

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

Наследуются открыто и доступны для использования в производном классе.

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

Не наследуются.

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

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

5. Каким образом описывается производный класс?

class Название\_Производного\_Класса : Тип\_Наследуеммых\_Данных Название\_Класса\_Родителя.

6. Наследуются ли конструкторы?

Конструкторы наследуются, но не переопределяются.

7. Наследуются ли деструкторы?

Деструкторы наследуются, но не переопределяются.

8. В каком порядке конструируются объекты производных классов?

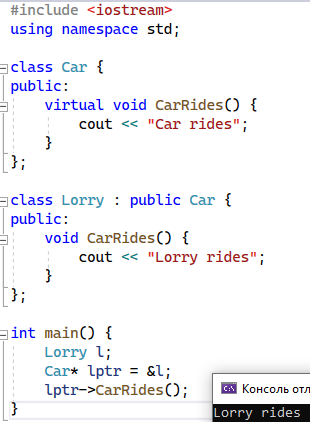
Сначала конструируется базовый класс, затем производный класс.

9. В каком порядке уничтожаются объекты производных классов?

Сначала уничтожается производный класс, затем базовый класс.

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

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?

Конструкторы не могут быть виртуальными, деструкторы могут быть виртуальными.

12. Наследуется ли спецификатор virtual?

Нет.

13. Какое отношение устанавливает между классами открытое наследование?

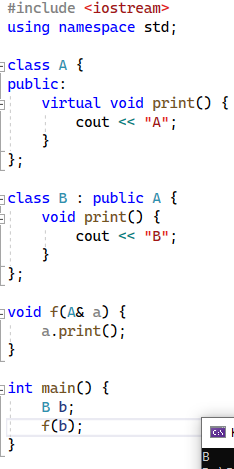
Производный класс является расширением базового класса.

14. Какое отношение устанавливает между классами закрытое наследование?

Производный класс реализует интерфейс базового класса, но не является его расширением.

15. В чем заключается принцип подстановки?

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 х?

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;

}