Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.4**

Дисциплина: «Информатика»

Тема: Объектно-ориентированное программирование. Простое наследование

Вариант 13

Выполнил:

Студент группы РИС-20-1б

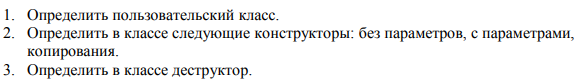
Кузнецов Михаил Сергеевиич

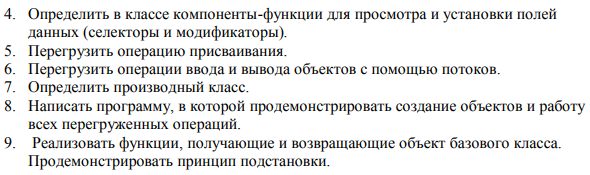
Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Постановка задачи**





**Анализ задачи**

**1.** Для решения задачи необходимо:

**1.1.** Организовать класс Person с полями name и age.

**1.2.** Организовать класс Employee с полями post, salary, который является производным класса Person.

**1.3.** Организовать необходимые методы для ввода данных в поля first и second: гетторы, сетторы, конструкторы, деструктор.

**1.4.** Организовать перегрузку оператора >> дружественной классу Person.

**1.5.** Организовать перегрузку оператора << дружественной классу Person.

**1.6.** Организовать перегрузку оператора - дружественной классу Person.

**1.7.** Организовать перегрузку оператора + дружественной классу Person.

**2.** В ходе работы были использованы следующие типы данных:

2.1 Тип данных int для хранения возраста;



2.2 Тип данных double для хранения зарплаты;



2.3 Тип данных string, для хранения имени.



**Код**

#include "Employee.h"

int main()

{

Person a;

cin >> a;

cout << a;

Person b = a;

cout << b;

Person c("Vasya", 35);

cout << c;

Employee e;

cout << e;

cin >> e;

cout << e;

Employee f = e;

Employee k("Sergey", 32, "software engineer", 100000.99);

cout << k;

cout << "Final salary: " << k.get\_final\_salary(50.25);

}

#include "Person.h"

Person::Person()

{

m\_name = "";

m\_age = 0;

}

Person::Person(string name, int age)

{

m\_name = name;

m\_age = age;

}

Person::Person(const Person& p)

{

m\_name = p.m\_name;

m\_age = p.m\_age;

}

Person::~Person()

{

}

string Person::get\_name()

{

return m\_name;

}

int Person::get\_age()

{

return m\_age;

}

void Person::set\_name(string name)

{

m\_name = name;

}

void Person::set\_age(int age)

{

m\_age = age;

}

Person& Person::operator=(const Person& p)

{

if (&p == this) return \*this;

m\_name = p.m\_name;

m\_age = p.m\_age;

return \*this;

}

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

{

cout << "Enter name: ";

in >> p.m\_name;

cout << "\nEnter age: ";

in >> p.m\_age;

cout << endl;

return in;

}

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

{

out << "\nName: " << p.m\_name;

out << "\nAge: " << p.m\_age << endl;

return out;

}

#include "Employee.h"

Employee::Employee():Person()

{

m\_post = "";

m\_salary = 0.0;

}

Employee::Employee(string name, int age, string post, double salary):Person(name, age)

{

m\_post = post;

m\_salary = salary;

}

Employee::Employee(const Employee& e)

{

m\_name = e.m\_name;

m\_age = e.m\_age;

m\_post = e.m\_post;

m\_salary = e.m\_salary;

}

Employee::~Employee()

{

}

string Employee::get\_post()

{

return m\_post;

}

double Employee::get\_salary()

{

return m\_salary;

}

double Employee::get\_final\_salary(double percent)

{

return m\_salary + (m\_salary \* percent / 100);

}

void Employee::set\_post(string post)

{

m\_post = post;

}

void Employee::set\_salary(double salary)

{

m\_salary = salary;

}

istream& operator>>(istream& in, Employee& e)

{

cout << "Enter name: ";

in >> e.m\_name;

cout << "\nEnter age: ";

in >> e.m\_age;

cout << "\nEnter post: ";

in >> e.m\_post;

cout << "\nEnter salary: ";

in >> e.m\_salary;

cout << endl;

return in;

}

ostream& operator<<(ostream& out, Employee& e)

{

out << "\nName: " << e.m\_name;

out << "\nAge: " << e.m\_age;

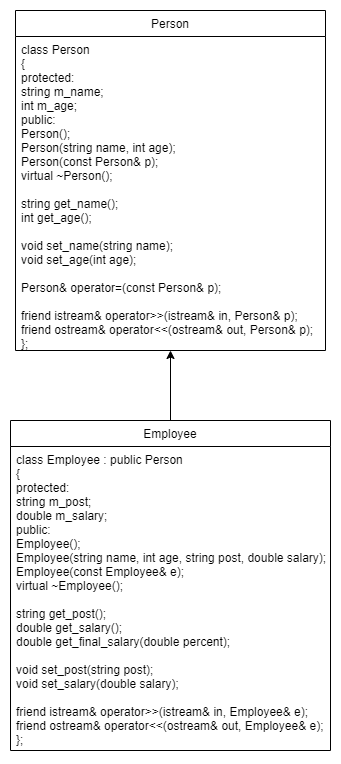
out << "\nPost: " << e.m\_post;

out << "\nSalary: " << e.m\_salary << endl;

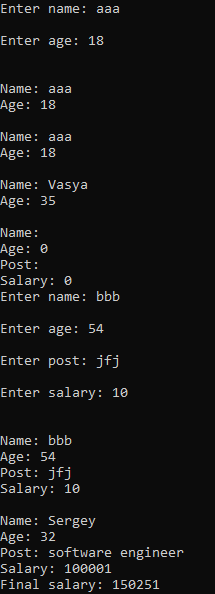
return out;

}

**UML диаграмма**



**Скриншоты тестов**

****

**Контрольные вопросы**



1. Наследование позволяет реализовать типизированное повторное использование объектов, создание иерархии «тип-подтип» с сохранением подтипом всех свойств своего предка. Можно рассматривать наследование для всех трех составляющих объекта.



2. Когда вы открыто наследуете родительский класс, то унаследованные public-члены остаются public.



3. Когда вы открыто наследуете родительский класс, то унаследованные private-члены остаются недоступными для дочернего класса.



4. К protected-членам родительского класса доступ открыт для членов дочернего класса.



5. 



6. Конструкторы не наследуются.



7. Деструкторы не наследуются.



8. 



9. 

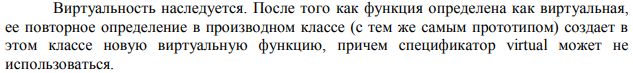


10. 

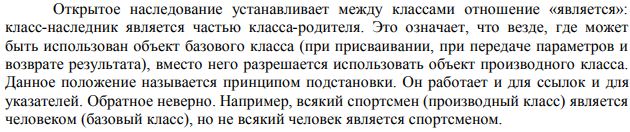


11. 



12. 



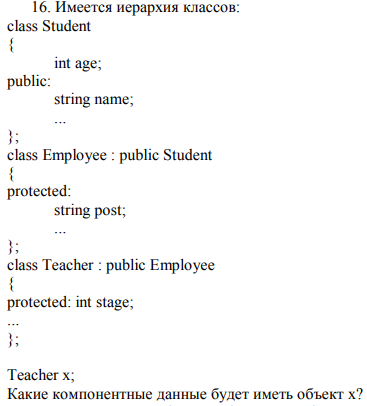
13. 



14. 



15. Пункт 13 и 14 – принцип подстановки.



16. age, name, post, stage.



17. Student(); Employee() : Student(); Teacher() : Employee();