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**

**о работе по информатике**

Семестр: 2

На тему: Лабораторная работа №4: «Простое наследование. Принцип подстановки».

**Вариант 13**

Выполнил студент ИВТ-22-2б:

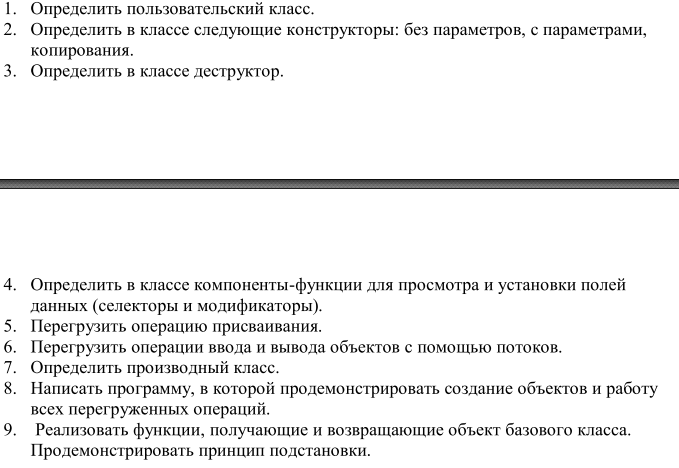
Коняев Александр Сергеевич

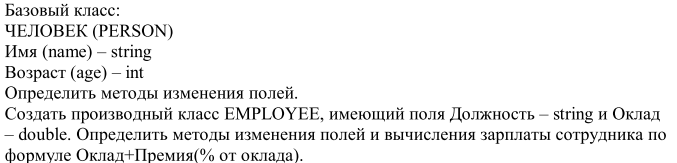
Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**





**Анализ задачи**

Класс Person

* Конструктор с параметрами
* Конструктор копирования
* Метод get\_name – получение фио
* Метод set\_name – запись фио
* Метод get\_age – получение возраста
* Метод set\_age – запись возраста
* Перегруженные операторы: =, >>, <<

Класс Employee

* Конструктор с параметрами
* Конструктор копирования
* Метод get\_dol – получение должности
* Метод set\_dol – запись должности
* Метод get\_okl – получение оклада
* Метод set\_okl – запись оклада
* Перегруженные операторы: =, >>, <<

**UML – диаграмма**

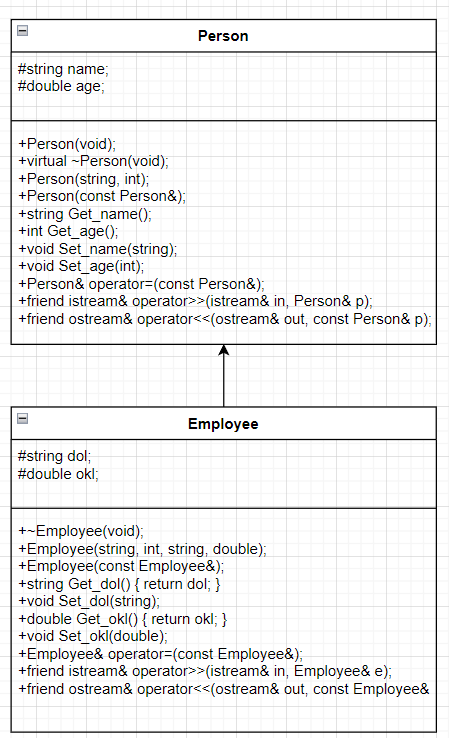


Рисунок 1 – UML-диаграмма.

**Код программы.**

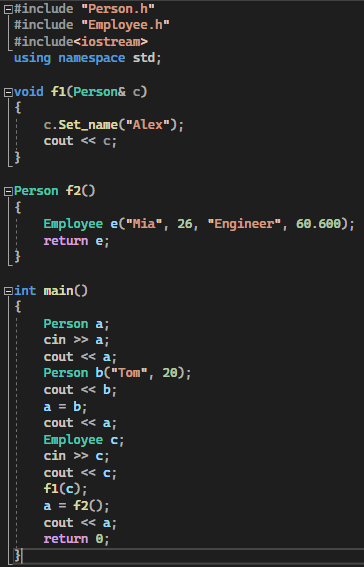


Рисунок 2 – Функция main

**Вывод программы.**

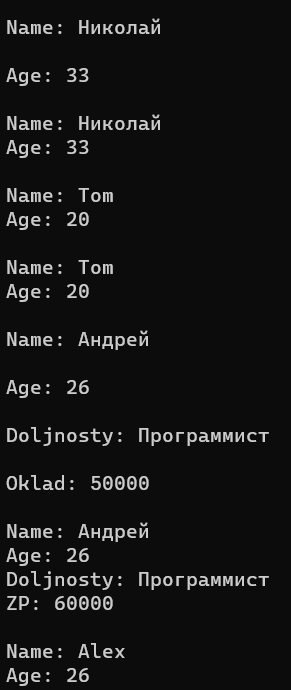
****

Рисунок 3 – Вывод программы

**Ответы на вопросы**

1. Механизм наследования используется для создания новых классов на основе уже существующих. Это позволяет повторно использовать код и уменьшать дублирование, упрощать поддержку и развитие программного обеспечения, а также облегчать его понимание и использование. В результате, наследование позволяет создавать более сложные и мощные программы, состоящие из набора связанных классов, которые могут быть легко расширены и изменены в будущем.
2. Компоненты класса, описанные со спецификатором public, наследуются в подклассах без изменений и могут быть использованы из любого места программы, где доступен объект подкласса. Это означает, что подкласс может использовать все публичные методы, свойства и поля базового класса, как если бы они были определены в самом подклассе.
3. Компоненты класса, описанные со спецификатором private, не наследуются в подклассах и не могут быть использованы извне класса. Они доступны только внутри самого класса и не видны за его пределами. Подкласс не может использовать приватные методы, свойства и поля базового класса.
4. Компоненты класса, описанные со спецификатором protected, наследуются в подклассах и могут быть использованы внутри самого класса и его подклассов. Они не доступны извне класса и не могут быть использованы вне его и его подклассов. Подкласс может использовать защищенные методы, свойства и поля базового класса.
5. Производный класс описывается с помощью ключевого слова "class", за которым следует имя класса, а после двоеточия указывается имя базового класса, от которого производится наследование. Например:

class DerivedClass : BaseClass {

// тело класса

}

В этом примере класс DerivedClass наследует свойства и методы от класса BaseClass. Внутри тела производного класса можно переопределять методы базового класса, добавлять новые свойства и методы.

1. Да, конструкторы также наследуются от базового класса и могут быть вызваны из производного класса с помощью ключевого слова "base". Кроме того, производный класс может иметь свой собственный конструктор, который может вызывать конструктор базового класса с помощью ключевого слова "base".
2. Да, деструкторы также наследуются от базового класса и вызываются автоматически при уничтожении объекта производного класса. Если производный класс определяет свой собственный деструктор, то он может вызвать деструктор базового класса с помощью ключевого слова "base".
3. Объекты производных классов конструируются в порядке наследования, начиная с базового класса и заканчивая самым производным. То есть, сначала вызывается конструктор базового класса, затем конструкторы промежуточных классов (если они есть), и, наконец, конструктор самого производного класса.
4. Объекты производных классов уничтожаются в обратном порядке, начиная с самого производного класса и заканчивая базовым классом. То есть, сначала вызывается деструктор самого производного класса, затем деструкторы промежуточных классов (если они есть), и, наконец, деструктор базового класса.
5. Виртуальные функции - это функции, которые могут быть переопределены в производных классах и вызываться для объектов этих классов через указатель или ссылку на базовый класс. Механизм позднего связывания позволяет вызывать виртуальные функции для объектов производных классов, используя указатель или ссылку на базовый класс, и при этом вызывать соответствующую функцию из производного класса. Это позволяет реализовывать полиморфизм и упрощает разработку и поддержку программного кода.
6. Да, конструкторы и деструкторы могут быть виртуальными. Виртуальный конструктор позволяет создавать объекты производных классов через указатель или ссылку на базовый класс, а виртуальный деструктор гарантирует правильное освобождение памяти при удалении объектов через указатель на базовый класс.
7. Спецификатор virtual не наследуется. Если в базовом классе метод объявлен как виртуальный, то он будет виртуальным и во всех производных классах, но это не означает, что спецификатор virtual будет наследоваться. Каждый класс должен явно указывать, что метод является виртуальным, если это требуется.
8. Открытое наследование устанавливает отношение "является" между классами, то есть производный класс является расширением базового класса и наследует все его свойства и методы.
9. Закрытое наследование устанавливает отношение "реализует" между классами, то есть производный класс реализует интерфейс базового класса, но не наследует его свойства и методы.
10. Принцип подстановки (Liskov substitution principle) заключается в том, что объекты должны быть заменяемыми на экземпляры их подтипов без изменения правильности выполнения программы. То есть, если S является подтипом T, то объекты типа T могут быть заменены объектами типа S без изменения корректности программы. Этот принцип является одним из основных принципов объектно-ориентированного программирования и помогает обеспечить гибкость и расширяемость системы.
11. Объект х будет иметь все компонентные данные, определенные в классах Teacher, Employee и Student: age, name, post и stage.
12. class Student

{

public:

int age;

string name;

Student() {

age = 0;

name = "";

}

};

class Employee : public Student

{

protected:

string post;

public:

Employee() {

post = "";

}

};

class Teacher : public Employee

{

protected:

int stage;

public:

Teacher() {

stage = 0;

}

};

1. class Student

{

public:

int age;

string name;

Student(int a, string n) {

age = a;

name = n;

}

};

class Employee : public Student

{

protected:

string post;

public:

Employee(string p) {

post = p;

}

};

class Teacher : public Employee

{

protected:

int stage;

public:

Teacher(int s) {

stage = s;

}

};

1. class Student

{

public:

int age;

string name;

Student(Student&s) {

age = s.age;

name = s.name;

}

};

class Employee : public Student

{

protected:

string post;

public:

Employee(Employee&e) {

post = e.post;

}

};

class Teacher : public Employee

{

protected:

int stage;

public:

Teacher(Teacher&t) {

stage = t.stage;

}

};

1. Для класса Student:

Student& operator=(const Student& other) {

if (this != &other) {

age = other.age;

name = other.name;

}

return \*this;

}

Для класса Employee:

Employee& operator=(const Employee& other) {

if (this != &other) {

Student::operator=(other);

post = other.post;

}

return \*this;

}

Для класса Teacher:

Teacher& operator=(const Teacher& other) {

if (this != &other) {

Employee::operator=(other);

stage = other.stage;

}

return \*this;

}