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 №5: «Наследование. Виртуальные функции. Полиморфизм».

**Вариант 13**

Выполнил студент ИВТ-22-2б:

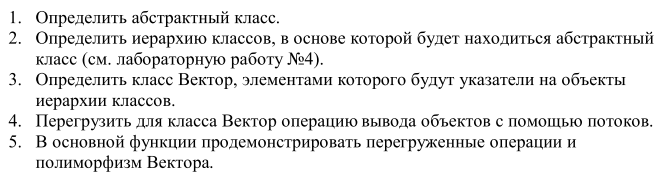
Коняев Александр Сергеевич

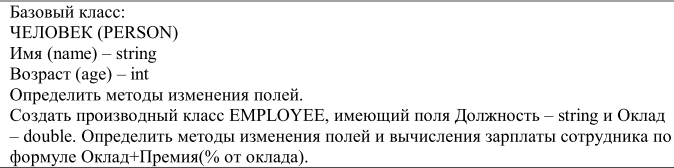
Проверил доцент кафедры ИТАС:

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: =, >>, <<

Класс Vector

* Конструкторы
* Метод Add – добавление элемента
* Перегруженный оператор <<

Класс Object

* Конструктор без параметров
* Метод Show – вывод информации

**UML – диаграмма**

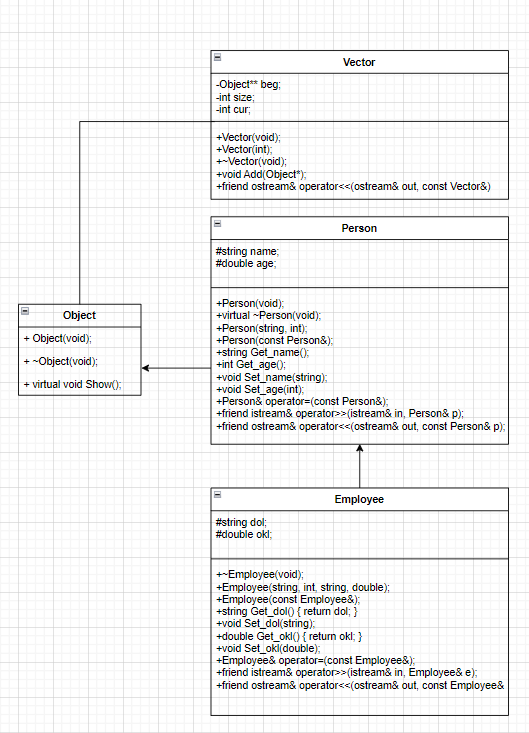


Рисунок 1 – UML-диаграмма.

**Код программы.**

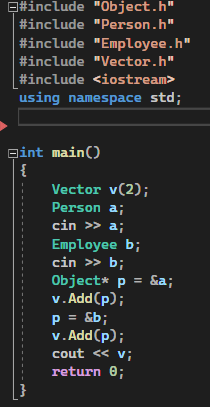


Рисунок 2 – Функция main

**Вывод программы.**

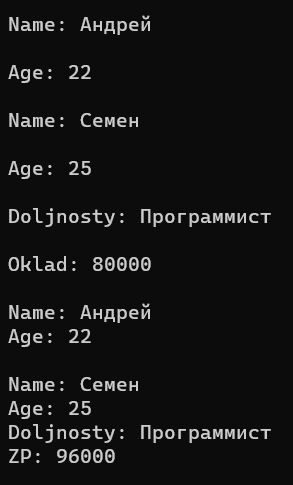
****

Рисунок 3 – Вывод программы

**Ответы на вопросы**

1. Чисто виртуальный метод называется методом-заглушкой (или абстрактным методом) и объявляется в базовом классе с помощью ключевого слова virtual и = 0 в конце объявления. Он не имеет реализации в базовом классе и должен быть переопределен в классах-наследниках.

Виртуальный метод, в отличие от чисто виртуального, имеет реализацию в базовом классе, но может быть переопределен в классах-наследниках. Он объявляется также с помощью ключевого слова virtual, но без = 0 в конце объявления.

Использование чисто виртуальных методов позволяет создавать абстрактные классы, которые могут служить только для создания объектов классов-наследников. Это позволяет создавать более гибкую архитектуру программы и упрощает поддержку кода.

1. Абстрактным классом называется класс, который содержит хотя бы один чисто виртуальный метод и не может быть использован для создания объектов. Он служит только для наследования и реализации его методов в классах-наследниках.\
2. Абстрактный класс представляет базовый класс, который может содержать как абстрактные, так и реализованные методы и свойства. Он создан для того, чтобы быть унаследованным другими классами, которые расширят его функциональность с помощью реализации его абстрактных методов.

Абстрактные классы являются мощным инструментом для проектирования приложений, которые могут быть удобно расширены и изменены в будущем. Они дают разработчикам возможность создавать сильную иерархию наследования, повышая повторное использование кода и облегчая понимание кода другим разработчикам.

1. Полиморфные функции в C++ - это функции, которые могут принимать и обрабатывать объекты различных типов. Они позволяют использовать одну и ту же функцию для работы с разными объектами, имеющими различные типы. Это делает код более гибким и удобным для использования, так как не требуется создавать отдельные функции для обработки каждого типа данных.
2. В C++ полиморфные функции реализуются с помощью механизма виртуальных функций. Для того, чтобы функция стала полиморфной, ее необходимо объявить виртуальной в базовом классе. Затем производные классы могут переопределить эту функцию и предоставить свою собственную реализацию. При вызове такой функции у объекта, компилятор выбирает соответствующую реализацию, в зависимости от типа объекта.

Пример полиморфной функции в C++:

class Shape {

public:

virtual void draw() {

cout << "Drawing a shape" << endl;

}

};

class Circle: public Shape {

public:

void draw() {

cout << \"Drawing a circle" << endl;

}

};

class Rectangle: public Shape {

public:

void draw() {

cout << \"Drawing a rectangle" << endl;

}

};

void drawShape(Shape \*s) {

s->draw();

}

int main() {

Circle c;

Rectangle r;

drawShape(&c);

drawShape(&r);

return 0;

}

В данном примере базовый класс Shape содержит виртуальную функцию draw(), которая выводит на экран информацию о том, что рисуется фигура. Классы Circle и Rectangle наследуют эту функцию и предоставляют свою реализацию.

В функции drawShape() принимается указатель на объект типа Shape. При вызове этой функции для объектов классов Circle и Rectangle, выбирается соответствующая реализация функции draw(). Таким образом, эта функция становится полиморфной и может работать с объектами разных классов.

1. Принцип подстановки (Liskov substitution principle, LSP) является одним из пяти принципов SOLID и определяет свойства, которыми должны обладать классы-наследники в отношении своих базовых классов. А именно, объекты классов-наследников должны быть полностью совместимы с объектами базового класса в том смысле, что они должны подставляться везде, где используются объекты базового класса, не нарушая корректность программы.\n\nПолиморфизм – это способность объекта иметь несколько форм (типов) одновременно. В программировании полиморфизм проявляется через использование виртуальных функций, которые могут быть переопределены наследниками базового класса.

Таким образом, принцип подстановки определяет правила наследования в С++, а полиморфизм позволяет выбирать нужную функцию для объекта в зависимости от его текущего типа. Оба этих понятия тесно связаны между собой и используются для обеспечения гибкости и расширяемости программного кода.

1. Концепция полиморфизма в С++ может быть реализована через виртуальные функции. Вот несколько примеров:

class Animal {

public:

virtual void makeSound() { cout << "Animal Sound" << endl; }

};

class Dog : public Animal {

public:

void makeSound() { cout << \"Bark" << endl; }

};

class Cat : public Animal {

public:

void makeSound() { cout << "Meow" << endl; }

};

class Cow : public Animal {

public:

void makeSound() { cout << \"Moo" << endl; }

};

В этом коде класс Animal является базовым классом, а классы Dog, Cat и Cow наследуются от него. Главная особенность этого кода заключается в том, что функция makeSound() объявлена как виртуальная в базовом классе Animal. Это позволяет замещать ее наследующими классами и использовать их в зависимости от текущего типа объекта.

1. еханизм позднего связывания (late binding) в C++ используется во время выполнения программы для определения, какая версия виртуальной функции должна быть вызвана для конкретного объекта в зависимости от его типа.

Это особенно полезно, когда у вас есть иерархия классов с виртуальными функциями и вам нужно вызвать функцию конкретного объекта в зависимости от его типа, который может не быть известен на этапе компиляции.

Например, в системе управления автомобилями может быть базовый класс Vehicle, с подклассами Car, Truck и Motorcycle. В базовом классе Vehicle может быть виртуальная функция drive(), которая определяет, как движется транспортное средство. Каждый подкласс может иметь свою собственную реализацию этой функции.

Когда вы вызываете функцию drive() на объекте Vehicle, которому был присвоен объект типа Car, будет вызвана версия функции drive() в классе Car. Если же тот же объект был бы присвоен объекту типа Truck или Motorcycle, будет вызвана соответствующая версия drive() в каждом из этих классов.

Таким образом, механизм позднего связывания используется в C++, чтобы обеспечить полиморфизм и правильное выполнение виртуальных функций в зависимости от конкретного объекта.