Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП**

**«№5»**

Выполнил:

студент группы РИС-23-2б

Ившин Максим Сергеевич

Проверила:

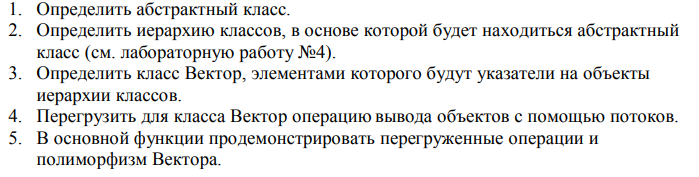
доцент кафедры ИТАС

О.А. Полякова

2024 г.

**Разработка алгоритма**

**Постановка задачи:**

** Анализ задачи:**

1. Абстрактный класс - это класс, который содержит чисто виртуальные функции, то есть функции без определения, и предназначен для обеспечения базовой структуры для производных классов.
2. Для построения иерархии классов с использованием абстрактного класса необходимо создать базовый абстрактный класс Object, от которого будут наследоваться класс Person и Student.
3. В основной функции программы создадим объекты классов Student и Person, добавим их в вектор, выведем содержимое вектора с и продемонстрируем работу полиморфизма, вызвав виртуальную функцию output() для объектов вектора.

**Код программы на C++:**

main.cpp:

#include <iostream>

#include <vector>

#include "Object.h"

#include "Object.cpp"

using namespace std;

int main()

{

    vector<Object \*> vect;

    Person person;

    Student student;

    cout << "Инициализация обьекта класса Person:\n";

    cin >> person;

    cout << "Введенные данные:\n";

    cout << person;

    cout << "Инициализация обьекта класса Student:\n";

    cin >> student;

    cout << "Введенные данные:\n";

    cout << student;

    vect.push\_back(&person);

    vect.push\_back(&student);

    cout << endl

         << endl;

    for (int i = 0; i < int(vect.size()); i++)

    {

        vect[i]->output();

    }

}

Object.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Object

{

public:

    Object(){};

    virtual ~Object(){};

    virtual void output() = 0;

    virtual void setName(const string) = 0;

    virtual void setAge(const int) = 0;

    virtual string getName() = 0;

    virtual int getAge() = 0;

};

class Person : public Object

{

protected:

    string name;

    int age;

public:

    Person();

    Person(string, int);

    Person(const Person &);

    ~Person();

    void operator=(const Person &);

    friend ostream &operator<<(ostream &, Person &);

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

    void setName(const string) override;

    void setAge(const int) override;

    void output() override;

    string getName() override;

    int getAge() override;

};

class Student : public Person

{

protected:

    int year\_of\_study;

public:

    Student();

    Student(string, int, int);

    void setYoS(int);

    string getName() override;

    int getAge() override;

    Student &operator++();

    Student &operator++(int);

    void setName(const string) override;

    void setAge(const int) override;

    void output() override;

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

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

};

Object.cpp:

#include <iostream>

#include <string>

using namespace std;

Student &Student::operator++()

{

    ++this->year\_of\_study;

    return \*this;

}

Student &Student::operator++(int)

{

    this->year\_of\_study++;

    return \*this;

}

Person::Person()

{

    cout << "Вызвался конструктор без параметров для объекта " << this << endl;

    this->name = "-";

    this->age = 0;

}

Person::Person(string name, int age)

{

    cout << "Вызвался конструктор с параметрами для объекта " << this << endl;

    this->name = name;

    this->age = age;

}

Person::Person(const Person &other)

{

    cout << "Вызвался конструктор копирования для объекта " << this << endl;

    this->name = other.name;

    this->age = other.age;

}

Person::~Person()

{

    cout << "Вызвался деструктор для объекта " << this << endl;

}

void Person::operator=(const Person &person)

{

    this->name = person.name;

    this->age = person.age;

}

ostream &operator<<(ostream &stream, Person &person)

{

    stream << "ФИО: " << person.name << endl

           << "Возраст: " << person.age << "\n\n";

    return stream;

}

istream &operator>>(istream &stream, Person &person)

{

    string name;

    int age;

    cout << "Введите имя: ";

    cin.seekg(cin.eof());

    getline(cin, name);

    cout << "Введите возраст: ";

    cin >> age;

    person.setName(name);

    person.setAge(age);

    return stream;

}

Student::Student()

{

    this->name = "-";

    this->age = 0;

    this->year\_of\_study = 0;

}

Student::Student(string name, int age, int yos)

{

    this->name = name;

    this->age = age;

    this->year\_of\_study = yos;

}

void Student::setYoS(int yos)

{

    this->year\_of\_study = yos;

}

ostream &operator<<(ostream &stream, const Student &student)

{

    stream << "Имя: " << student.name << endl

           << "Возраст: " << student.age << endl

           << "Год обучения: " << student.year\_of\_study << endl

           << endl;

    return stream;

}

istream &operator>>(istream &stream, Student &student)

{

    string name;

    int yos, age;

    cout << "Введите ФИО: ";

    cin.seekg(cin.eof());

    getline(stream, student.name);

    cout << "Введите возраст: ";

    stream >> age;

    student.age = age;

    cout << "Введите год обучения: ";

    stream >> yos;

    student.setYoS(yos);

    return stream;

}

void Student::setName(const string name)

{

    this->Person::name = name;

}

void Student::setAge(const int age)

{

    this->Person::age = age;

}

void Person::setName(const string name)

{

    this->name = name;

}

void Person::setAge(const int age)

{

    this->age = age;

}

void Person::output()

{

    cout << "\tДанные о человеке:\nФИО: " << this->name << endl

         << "Возраст: " << this->age << "\n\n";

}

void Student::output()

{

    cout << "\tДанные о студенте:\nФИО: " << this->name << endl

         << "Возраст: " << this->age << endl

         << "Год обучения: " << this->year\_of\_study << "\n\n";

}

string Person::getName()

{

    return this->name;

}

int Person::getAge()

{

    return this->age;

}

string Student::getName()

{

    return this->name;

}

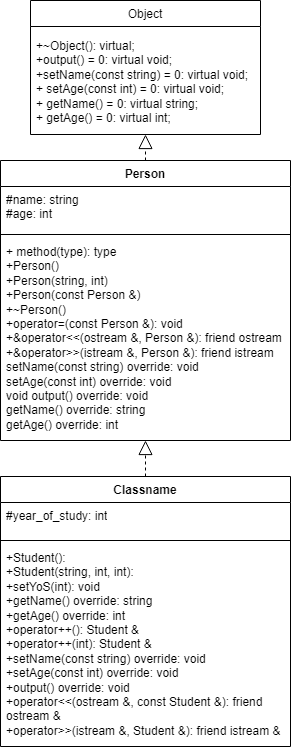
int Student::getAge()

{

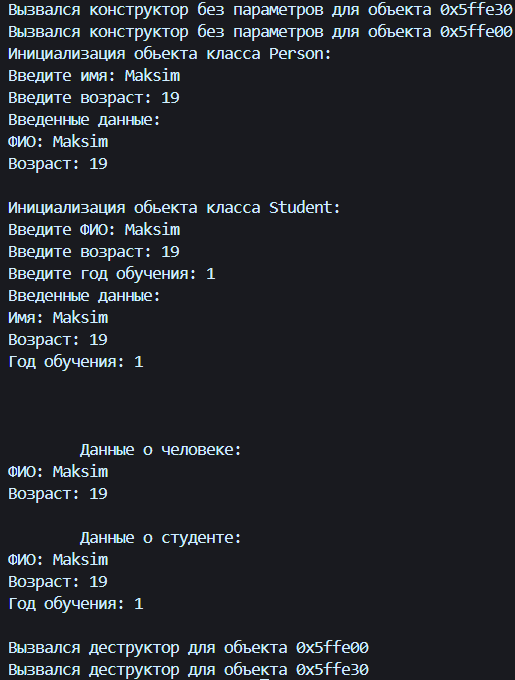
    return this->age;

}

**Диаграмма класса:**



**Пример работы программы:**



**Контрольные вопросы**

1. **Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?**

Чисто виртуальный метод (также известный как pure virtual method) - это метод, который объявлен в базовом классе, но его реализация не предоставлена. Вместо этого, он имеет только декларацию (т.е. прототип функции), и в конце этой декларации стоит ключевое слово = 0;.

<тип> <имя функции> (<список параметров>) = 0;

Это означает, что производные классы обязаны реализовать этот метод, иначе они сами станут абстрактными и не могут быть инстанцированы. Виртуальный метод, в отличие от чисто виртуального, может иметь реализацию в базовом классе, которая может быть переопределена в производном классе.

Основное различие между виртуальной функцией и чисто виртуальной функцией является то, что виртуальная функция является функцией в базовом классе, который объявлен с использованием виртуального ключевого слова в то время, как чистая виртуальная функция является виртуальной функцией в базовом классе без определения функции.

1. **Какой класс называется абстрактным?**

Абстрактный класс - это класс, который содержит хотя бы один чисто виртуальный метод. Такой класс сам по себе не может быть инстанцирован, так как не предоставляет полную реализацию всех методов. Он служит в качестве основы для определения интерфейса для других классов.

1. **Для чего предназначены абстрактные классы?**

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс может использоваться только в качестве базового для других классов — объекты абстрактного класса создавать нельзя, поскольку прямой или косвенный вызов чисто виртуального метода приводит к ошибке при выполнении.

1. **Что такое полиморфные функции?**

Полиморфные функции - это функции, которые могут быть вызваны с различными типами аргументов, благодаря механизму позднего связывания. Это позволяет использовать одну и ту же функцию с разными объектами, даже если они принадлежат к разным классам. Понятие позднее связывание означает, что код вызова нужной функции фор-мируется при выполнении программы.

1. **Чем полиморфизм отличается от принципа подстановки?**

При полиморфизме функция может изменять свой функционал. Полиморфизм - это способность объекта одного класса вести себя как

объект другого класса. Это достигается через наследование и виртуальные функции. Принцип подстановления говорит о том, что объекты можно заменять объектами их типов без изменения программы. Полиморфизм идет дальше, позволяя объектам разных классов заменяться друг другом в определенных контекстах.

1. **Привести примеры иерархий с использованием абстрактных**

**классов.**

#include <iostream>

* Абстрактный класс Animal class Animal {

public:

virtual void makeSound() = 0; // Чисто виртуальная функция, делает класс абстрактным

};

* Класс Dog, наследуется от Animal

class Dog : public Animal {

public:

void makeSound() override {

std::cout << "Woof" << std::endl;

}

};

* Класс Cat, наследуется от Animal class Cat : public Animal {

public:

void makeSound() override { std::cout << "Meow" << std::endl;

}

};

int main() {

Dog dog;

Cat cat;

dog.makeSound(); // Выводит "Woof"

cat.makeSound(); // Выводит "Meow"

return 0;

}

1. **Привести примеры полиморфных функций.**

class Abstract { //Абстрактный класс

public:virtual void print\_msg() = 0;

};

1. **В каких случаях используется механизм позднего связывания?**

Позднее связывание означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше. Позднее связывание достигается в С++ с помощью использования виртуальных функций и производных классов.

Механизм позднего связывания используется в следующих случаях:

* Когда вызывается виртуальный метод объекта.
* Когда происходит вызов функции через указатель или ссылку на базовый класс, где фактический вызов осуществляется для производного класса.

В шаблонах, когда конкретный тип определяется во время компиляции.

**GitHub:** *https://github.com/geroineee/PNRPU*