Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №5
«Наследование. Виртуальные функции. Полиморфизм»

Выполнил:
студент первого курса
ЭТФ группы РИС-23-3б
Акбашева Софья Руслановна

Проверила:
Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Наследование. Виртуальные функции. Полиморфизм

Цель задания

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Создание иерархии классов с использованием простого наследования и абстрактного класса.

3) Изучение полиморфизма и виртуальных методов.

Постановка задачи

1. Определить абстрактный класс.

2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4).

3. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.

4. Перегрузить для класса Вектор операцию вывода объектов с помощью потоков.

5. В основной функции продемонстрировать перегруженные операции и полиморфизм Вектора.

Задание

Базовый класс:

ЧЕЛОВЕК (PERSON)

Имя (name) – string Возраст (age) – int

Определить методы изменения полей.

Создать производный класс STUDENT, имеющий поля Предмет – string и Оценка – int. Определить методы изменения полей и метод, выдающий сообщение о неудовлетворительной оценке.

Анализ задачи

1) Использование директивы #pragma once предотвращает повторную загрузку заголовочного файла, что может улучшить производительность программы, особенно при использовании множества включаемых файлов.

2) Необходимо реализовать класс Person и дочерний для него класс Student.

3) Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.

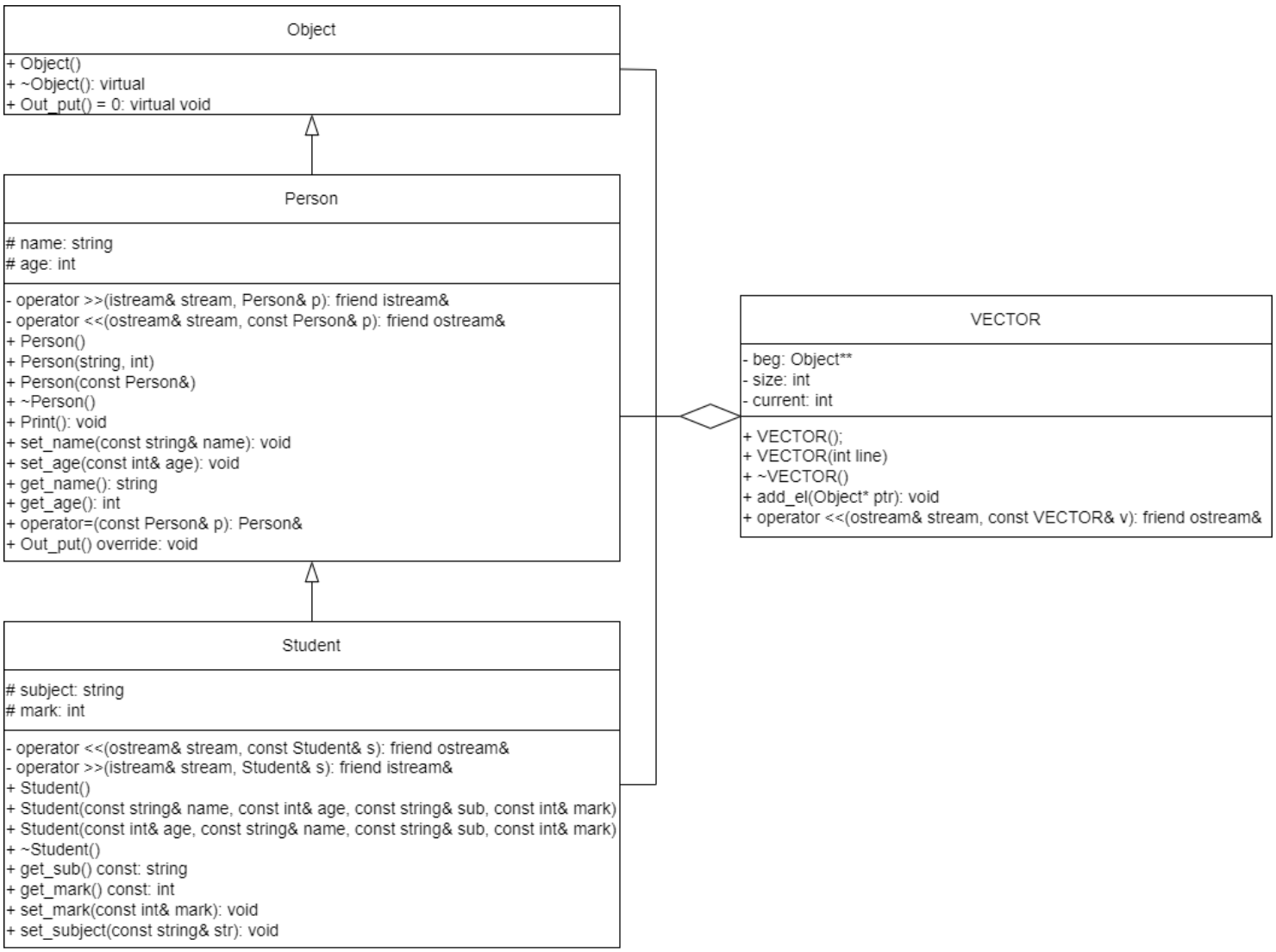
4) Класс Person содержит защищенные данные name и age, что предотвращает их изменение извне класса. Также определены конструктор по умолчанию, конструктор с параметрами, конструктор копирования и деструктор. Предоставлены функции для установки значений и вывода информации о персоне. Определён оператор присваивания и дружественные функции для работы с потоками ввода/вывода.

5) Класс Student является производным от Person. Он добавляет свои собственные защищенные данные subject и mark, а также предоставляет дополнительные функции для установки этих значений и вывода информации о студенте. Определены конструктор с параметрами, функции для установки предмета и оценки, а также дружественные функции для работы с потоками ввода/вывода.

6) Перегруженные операторы +, -, = и << позволяют выполнять различные операции с объектами класса. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса в поток вывода.

7) В главной функции показаны операции, которые можно выполнить с объектом классов.

Блок схема



Код

Файл Class_5.cpp

```
#include <iostream>
#include <string>
#include "person.h"
#include "vector.h"

int main(){
   system("chcp 1251 > Null");
   VECTOR vec(2);

   Person p1;//класс человек
   cin >> p1;
   cout << endl << p1 << endl ;

   Student st1;//класс студент
   cin >> st1;
   cout << st1 << endl ;


   Object* ob = &p1;//класс указателей на классы
   vec.add_el(ob);
   ob = &st1;
   vec.add_el(ob);
```

```
    cout << vec;

    cout << endl;

}
```

Файл object.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>

using namespace std;

class Object {
public:
        Object() {};
        virtual ~Object() {};
        virtual void Out_put() = 0;
};
```

Файл person.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include "object.h"
using namespace std;

class Person : public Object {
        friend istream& operator >>(istream& stream, Person& p){//друж-я ф-я для потока ввода
                string temp;
                int a;

                cout << "Введите имя: ";
                stream.ignore();
                getline(stream, temp);
                p.set_name(temp);

                cout << "Введите возраст: ";
                stream >> a;
                p.set_age(a);
                return stream;
        }

        friend ostream& operator <<(ostream& stream, const Person& p) {//оператор для потока вывода
                stream << "Имя " << p.name << endl;
                stream << "Возраст " << p.age << endl;
                return stream;
        }

protected:
        string name;
        int age;
public:
        Person() {//конструктор по умолчанию
                this->name = "";
                this->age = 0;
        }
        Person(string name, int age) {//конструктор с параметрами
                this->name = name; //сначала передается имя, потом возраст
                this->age = age;
        }
        Person(const Person& p) {//конструктор копирования
```

```
		this->name = p.name;
		this->age = p.age;
	}
	~Person() {};//деструктор

	void set_name(const string& name) {//установка имени
		this->name = name;
	}
	void set_age(const int& age) {//установка возраста
		this->age = age;
	}

	string get_name() {//получение имени
		return name;
	}
	int get_age() {//получение возраста
		return age;
	}

	Person& operator=(const Person& p) {//оператор присваивания
		this->name = p.name;
		this->age = p.age;
		return *this;
	};

	void Out_put() override {//вывод параметров класса person
		cout << "ФИО: " << this->name << endl << "Возраст: " << this->age;
	}
};

class Student : public Person {
	friend ostream& operator <<(ostream& stream, const Student& s) {//оператор для потока вывода
		const Person* human_stud = &s;
		stream << *human_stud;
		stream << "Предмет " << s.subject << endl;
		stream << "Оценка " << s.mark << endl;

		if (s.mark) {
			if (s.mark < 1 || s.mark > 5) {
				cout << "Оценка неправильная!" << endl;
			}
			else if (s.mark < 3) {
				cout << "Отметка неудовлетворительна!" << endl;
			}
			else if (s.mark == 3) {
				cout << "Зачет." << endl;
			}
			else if (s.mark == 4) {
				cout << "Хорошая оценка" << endl;
			}
			else {
				cout << "Отличная оценка" << endl;
			}
		}
		return stream;
	}

	friend istream& operator >>(istream& stream, Student& s) {//оператор для потока ввода
		string temp;
		int a;

		Person* const human_stud = &s;
```

```
                stream >> *human_stud;

                cout << "Введите предмет: ";
                stream >> temp;
                s.set_subject(temp);

                cout << "Введите оценку: ";
                stream >> a;
                s.set_mark(a);
                cout << endl;

                return stream;
        }
protected:
        string subject;
        int mark;
public:
        Student() {//конструктор по умолчанию
                this->mark = 0;
                this->subject = "";
        }
        Student(const Student& p) {//конструктор копирования
                this->subject = p.subject;
                this->mark = p.mark;
        }
        Student(const string& name, const int& age, const string& sub, const int& mark) : Person(name, age)
{//конструктор с параметрами
                this->set_mark(mark);
                this->set_subject(sub);
        }
        Student(const int& age, const string& name, const string& sub, const int& mark) : Student(name, age, sub,
mark) {};
        ~Student() {};//деструктор

        string get_sub() const {//получение предмета
                return this->subject;
        }
        int get_mark() const {//получение оценки
                return this->mark;
        }
        void set_mark(const int& mark) {//установка предмета
                this->mark = mark;
        }
        void set_subject(const string& str) {//установка оценки
                this->subject = str;
        };
};
```

Файл vector.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include "object.h"
using namespace std;

class VECTOR {
        Object** beg;
        int size;//размер вектора
        int current;
public:
        VECTOR() {//конструктор по умолчанию
                beg = 0;
```

```
		size = 0;//всего элементов
		current = 0;//элементов сейчас
	};

	VECTOR(int line) {//конструктор с длиной
		beg = new Object * [line];
		current = 0;
		size = line;
	}

	~VECTOR() {};//деструктор

	void add_el(Object* ptr) {//добавление элемента в вектор
		if (current < size) {//если вектор не заполнен
			beg[current] = ptr;
			current++;//увеличиваю текущее количество элементов
		}
	};

	friend ostream& operator <<(ostream& stream, const VECTOR& v) {//оператор для потока вывода
		if (v.size == 0) {
			cout << endl << "Вектор пустой(";
		}
		else {
			Object** ptr = v.beg;
			for (int i = 0; i < v.current; i++) {
				(*ptr)->Out_put();
				cout << endl << endl;
				ptr++;
			}
		}
		return stream;
	}
};
```

Результаты работы

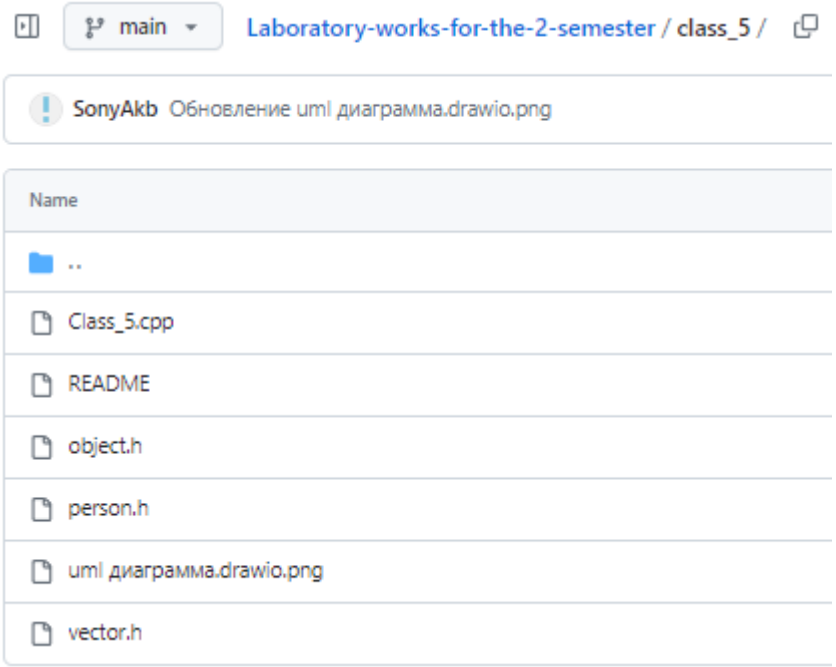
```
Введите имя: Оля
Введите возраст: 19

Имя Оля
Возраст 19

Введите имя: Мефодий
Введите возраст: 123
Введите предмет: информатика
Введите оценку: 5

Имя Мефодий
Возраст 123
Предмет информатика
Оценка 5
Отличная оценка

Имя: Оля
Возраст: 19

Имя: Мефодий
Возраст: 123
```

Вывод

В ходе работы я применила знания о работе с классами. В частности, о простом наследовании с помощью принципа подстановки. Была создана иерархия классов с использованием простого наследования и абстрактного класса. Также был определен класс Вектор, элементами которого являются указатели на объекты иерархии классов. Для класса Вектор была перегружена операция вывода объектов с помощью потоков. В основной функции были продемонстрированы перегруженные операции и полиморфизм Вектора.

GitHub

Ссылка: https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/class_5



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.

2. **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.

3. **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.

4. **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.

5. **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.

6. **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;
}
```

7. **Привести примеры полиморфных функций.**

```
class Abstract { //Абстрактный класс
public:virtual void print_msg() = 0;
};
```

8. **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 C++ 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.