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 № 15.5**

Дисциплина: «Информатика»

Тема: Объектно-ориентированное программирование. Наследование. Виртуальные функции. Полиморфизм.

Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

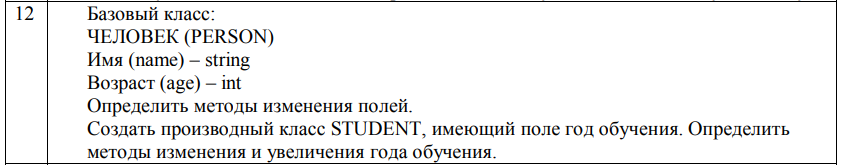
**Пермь 2023**

**Цель работы**

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.



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение абстрактного класса Object и определение необходимых методов данного класса.
   2. Реализовать определение класса Vector и определение необходимых методов и операторов данного класса.
   3. Реализовать определение производного от класса Object класса Person и определение необходимых методов и операторов данного класса.
   4. Реализовать определение производного от класса Person класса Student и определение необходимых методов и операторов данного класса
   5. Реализовать применение этих функций в главной функции.
2. В ходе работы были использованы типы данных:
   1. Для конструктора без параметров Object / Person / Student / Vector класса Object / Person / Student / Vector не используются аргументы:

Сам конструктор заполняет атрибуты объекта класса “пустотами”

* 1. Для конструктора с параметрами Person / Student класса Person / Student используются следующие аргументы:
  2. Тип int: год обучения
  3. Тип int: возраст
  4. Тип string: имя

Сам конструктор заполняет атрибуты объекта класса параметрами.





* 1. Для конструктора копирования Person / Student класса Person / Student используются следующие аргументы:
  2. Адресация типа const Person& / const Student&: объект копирования.

Сам конструктор заполняет атрибуты объекта класса атрибутами объекта копирования.

* 1. Для деструктора Object / Person / Student / Vector класса Object / Person / Student / Vector не используются аргументы:

Сам деструктор удаляет атрибуты объекта класса.

* 1. Для селектора get\_name не используются аргументы.

Сам селектор имеет тип string и возвращает значение поля name.

* 1. Для селектора get\_age не используются аргументы.

Сам селектор имеет тип int и возвращает значение поля age.

* 1. Для модификатора set\_name используются следующие аргументы:

1. Тип string: имя.

Сам модификатор имеет тип void, поскольку при работе селектора не нужно возвращать значение.

* 1. Для модификатора set\_age используются следующие аргументы:

1. Тип int: возраст

Сам модификатор имеет тип void, поскольку при работе селектора не нужно возвращать значение.

* 1. Для метода increase\_year не используются аргументы.

Сам метод имеет тип void, поскольку при работе метода не нужно возвращать значение.

* 1. Для метода перегрузки оператора присваивания используются следующие аргументы:
  2. Тип const Person& / Student&: ссылка на объект.

Сам метод имеет тип Person& / Student& и возвращает указатель на объект this.

* 1. Для дружественной функции перегрузки оператора ввода используются следующие аргументы:
  2. Тип istream&: поток ввода.
  3. Тип const Person& / const Student&: ссылка на объект.

Сама функция имеет тип istream& и возвращает вводимые данные.

* 1. Для дружественной функции перегрузки оператора вывода используются следующие аргументы:
  2. Тип ostream&: поток вывода.
  3. Тип const Person& / const Student&: ссылка на объект.

Сама функция имеет тип ostream& и возвращает выводимые данные.

* 1. Для метода Add используются следующие аргументы:
     + 1. Тип указатель на Object: указатель на добавляемый объект.

Сам метод имеет тип тип void, поскольку при работе метода не нужно возвращать значение.

* 1. Для метода перегрузки оператора присваивания используются следующие аргументы:

1. Тип const Person& / Student&: ссылка на объект.

Сам метод имеет тип Person& / Student& и возвращает указатель на объект this.

* 1. Для дружественной функции перегрузки оператора ввода используются следующие аргументы:
  2. Тип istream&: поток ввода.
  3. Тип const Person& / const Student&: ссылка на объект.

Сама функция имеет тип istream& и возвращает вводимые данные.

* 1. Для дружественной функции перегрузки оператора вывода используются следующие аргументы:
  2. Тип ostream&: поток вывода.
  3. Тип const Person& / const Student& / const Vector&: ссылка на объект.

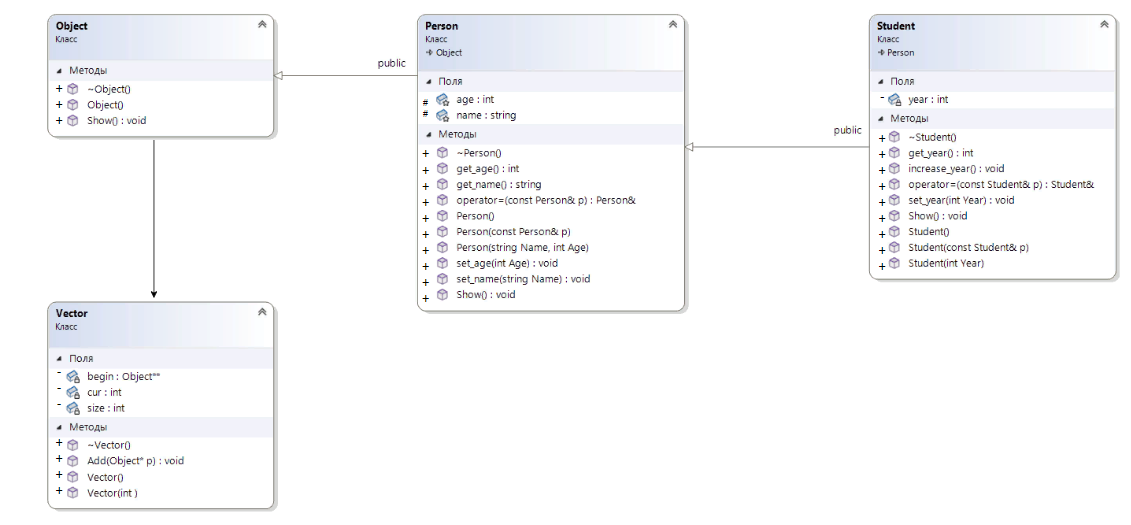
Сама функция имеет тип ostream& и возвращает выводимые данные.

* 1. Для метода Show не используются аргументы.

Сам метод имеет тип void, поскольку при работе метода не нужно возвращать значение.

1. Для решения задачи данные были представлены в следующем виде:
   1. Для работы с данными используются атрибуты класса.
2. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Ввод данных реализован с помощью оператора cin, используемых при реализации в главной функции и функций перегрузки оператора.
   2. Вывод данных реализован с помощью оператора cout, используемого при реализации в главной функции и функций перегрузки операторов ввода-вывода.
3. Поставленные задачи решены следующими действиями:
   1. Определение класса Object / Person / Student было реализовано в заголовочном файле Object.h / Person.h / Student.h, определение конструкторов, деструктора, селекторов и модификаторов класса – в файле Object.cpp / Person.cpp / Student.cpp. Основной блок программы описан в главном файле Лабораторная работа № 18.5.cpp. Абстрактный класс Object имеет виртуальную функцию Show для реализации в других классах.
   2. При работе с объектами класса Vector пользователь имеет возможность добавить объекты других классов, унаследованных от класса Object с помощью метода Add. В качестве параметра метода используется указатель на объект типа Object.
   3. При работе с объектами классов Person и Student обращение к конструктору без параметра аналогично выделению памяти под переменную. Обращение к конструктору с параметрами аналогично обращению к методу Init. Обращение к конструктору копирования аналогично оператору присваивания. Обращение к селекторам и модификаторам аналогично обращению к методам или к полям структуры.
   4. При работе с перегруженными операторами работа без объектов созданного класса происходит по умолчанию. Если при вызове перегруженных операторов используются объекты класса, то операторы работают согласно их определению в этом классе.

**Диаграмма классов**

****

**Реализация задачи на языке С++**

**Заголовочный файл Object.h**

#pragma once

class Object

{

public:

Object() {} // конструктор по умолчанию

~Object() {} //деструктор

virtual void Show() = 0; // чисто виртуальная функция

};

**Заголовочный файл Vector.h**

#pragma once

#include <iostream>

#include "Object.h"

#include <string>

using namespace std;

class Vector

{

private:

Object\*\* begin; // Указатель на первый элемент вектора

int size; // Размер

int cur; // Текущая позиция

public:

Vector();

Vector(int); // конструктор копирования

~Vector();

friend ostream& operator<< (ostream& out, const Vector& Pointer);

void Add(Object\* p); // Добавление элемента в вектор

};

**Файл с описанием методов класса Vector.cpp**

#include "Vector.h"

Vector::Vector()

{

begin = nullptr;

size = 0;

cur = 0;

}

Vector::Vector(int N)

{

begin = new Object \* [N];

size = N;

cur = 0;

}

Vector::~Vector()

{

if (begin != nullptr)

delete[] begin;

begin = nullptr;

}

void Vector::Add(Object\* p)

{

if (cur < size)

{

begin[cur] = p;

cur++;

}

}

ostream& operator<<(ostream& out, const Vector& vec)

{

if (vec.size == 0)

out << "Vector is empty!" << endl;

else

{

Object\*\* ptr = vec.begin;

for (int i = 0; i < vec.cur; i++)

{

(\*ptr)->Show();

ptr++;

}

}

return out;

}

**Заголовочный файл Person.h**

#pragma once

#include "Object.h"

#include <string>

using namespace std;

class Person : public Object

{

protected:

string name;

int age;

public:

Person();

Person(string Name, int Age);

Person(const Person& p);

~Person();

string get\_name();

void set\_name(string Name);

int get\_age();

void set\_age(int Age);

Person& operator= (const Person& p); // Оператор присваивания

friend istream& operator>> (istream& in, Person& p); // Оператор вывода

friend ostream& operator<< (ostream& out, const Person& p); // Оператор ввода

void Show() override;

};

**Файл с описанием методов класса Person.cpp**

#include "Person.h"

#include "Object.h"

#include<string>

#include <iostream>

using namespace std;

Person::Person()

{

name = " ";

age = 0;

}

Person::Person(string Name, int Age)

{

name = Name;

age = Age;

}

Person::Person(const Person& p)

{

name = p.name;

age = p.age;

}

Person::~Person()

{

}

string Person::get\_name()

{

return name;

}

void Person::set\_name(string Name)

{

name = Name;

}

int Person::get\_age()

{

return age;

}

void Person::set\_age(int Age)

{

age = Age;

}

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

{

name = p.name;

age = p.age;

return \*this;

}

istream& operator >> (istream& in, Person& p) // Оператор ввода

{

cout << "INPUT NAME: ";

in >> p.name;

cout << "INPUT AGE: ";

in >> p.age;

return in;

}

ostream& operator << (ostream& out, const Person& p) // Оператор вывода

{

out << "\nNAME: " << p.name << endl;

out << "AGE: " << p.age << endl;

return out;

}

void Person::Show()

{

cout << "\nNAME: " << name;

cout << "\nAGE: " << age << endl;

}

**Заголовочный файл Student.h**

#pragma once

#include "Person.h"

class Student : public Person

{

private:

int year;

public:

Student();

Student(int Year);

Student(const Student& p);

~Student();

int get\_year();

void set\_year(int Year);

Student& operator= (const Student& p); // Оператор присваивания

friend istream& operator>> (istream& in, Student& p); // Оператор вывода

friend ostream& operator<< (ostream& out, const Student& p); // Оператор ввода

void increase\_year();

void Show() override;

};

**Файл с описанием методов класса Student.cpp**

#include "Student.h"

#include <iostream>

using namespace std;

Student::Student()

{

year = 0;

}

Student::Student(int Year)

{

year = Year;

}

Student::Student(const Student& p)

{

year = p.year;

name = p.name;

age = p.age;

}

Student::~Student(){}

int Student::get\_year()

{

return year;

}

void Student::set\_year(int Year)

{

year = Year;

}

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

{

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

name = p.name;

age = p.age;

year = p.year;

return \*this;

}

istream& operator >> (istream& in, Student& p) // Оператор ввода

{

cout << "\nINPUT NAME: ";

in >> p.name;

cout << "INPUT AGE: ";

in >> p.age;

cout << "INPUT YEAR: ";

in >> p.year;

return in;

}

ostream& operator << (ostream& out, const Student& p) // Оператор вывода

{

out << "\nNAME: " << p.name << endl;

out << "AGE: " << p.age<<endl;

out << "YEAR: " << p.year << endl;

return out;

}

void Student::increase\_year()

{

year += 1;

}

void Student::Show()

{

cout << "\nNAME: " << name;

cout << "\nAGE: " << age;

cout << "\nYEAR: " << year << endl;

}

**Файл с главной программой main.cpp**

#include <iostream>

#include <string>

#include "Object.h"

#include "Person.h"

#include "Student.h"

#include "Vector.h"

using namespace std;

int main()

{

/\*

Person A;

cin >> A;

cout << "Operator << " << endl;

cout << A; // через перегруженную функцию

Object\* ptr = &A;

cout << "Show()" << endl;

ptr->Show();

Student B;

cin >> B;

cout << "Operator << " << endl;

cout << B;

ptr = &B;

cout << "Show()" << endl;

ptr->Show();

\*/

Vector m(3);

Person A;

cin >> A;

Student B;

cin >> B;

Object\* p = &A;

m.Add(p);

p = &B;

m.Add(p);

cout << m;

}

**Ответы на вопросы**

1. Функция, равная нулю, ее нельзя вызвать
2. Класс без методов и полей
3. Для наследования от них нормальных классов, симуляция абстрактных сущностей
4. Функции, переопределенные в классах-наследниках
5. При полиморфизме друг друга замещают классы а не функции
6. От «фигуры» наследуются «треугольник» и «круг»
7. Вывод площади для предыдущей иерархии
8. При использовании виртуальных функций