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 №18.5**

Дисциплина: основы алгоритмизации и программирования

Тема: “Классы и объекты. Полиморфизм.”

Вариант 15

Выполнил:

студент группы ИВТ-20-2Б

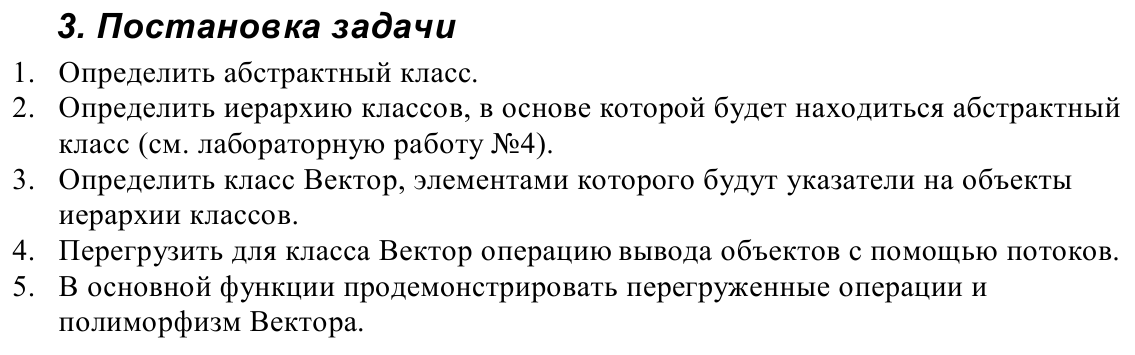
Чувашев Максим

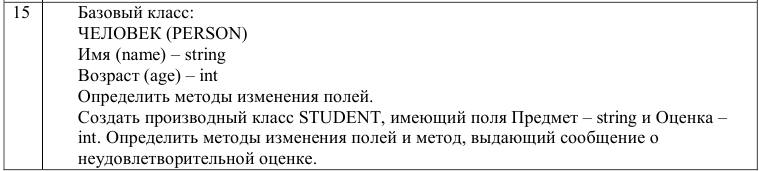
Проверила: доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**





**Анализ задачи**

1. Используемые типы данных
   1. Double
   2. Int
   3. Person
   4. Student
   5. Object
   6. Bool
   7. String
2. Действия над используемыми данными
   1. Int, double – арифметические операции.
3. Структуры
   1. Object – чисто виртуальный класс

#pragma once

class Object

{

public:

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

};

* 1. Person : public Object – класс для хранения сведений о человеке. Конструктор создает сведения о человеке

#pragma once

#include "Object.h"

#include <string>

#include <iostream>

using namespace std;

class PERSON: public Object

{

public:

//конструктор без параметров

PERSON();

//деструктор

virtual ~PERSON();

void Show();// функция для просмотра атрибутов класса с помощью указателя

//констрктор с параметрами

PERSON(string, int);

//конструктор копирования

PERSON(const PERSON&);

//Геттеры

string Get\_name()

{

return name;

}

int Get\_age()

{

return age;

}

//Сеттеры

void Set\_name(string);

void Set\_age(int);

//перегрузка операции присваивания

PERSON& operator=(const PERSON&);

//глобальные операторы-функции ввода-вывода

friend istream& operator>>(istream& in, PERSON& c);

friend ostream& operator<<(ostream& out, const PERSON& c);

//атрибуты

protected:

string name;

int age;

};

#pragma once

#include "Object.h"

#include <string>

#include <iostream>

using namespace std;

class PERSON: public Object

{

public:

//конструктор без параметров

PERSON();

//деструктор

virtual ~PERSON();

void Show();// функция для просмотра атрибутов класса с помощью указателя

//констрктор с параметрами

PERSON(string, int);

//конструктор копирования

PERSON(const PERSON&);

//Геттеры

string Get\_name()

{

return name;

}

int Get\_age()

{

return age;

}

//Сеттеры

void Set\_name(string);

void Set\_age(int);

//перегрузка операции присваивания

PERSON& operator=(const PERSON&);

//глобальные операторы-функции ввода-вывода

friend istream& operator>>(istream& in, PERSON& c);

friend ostream& operator<<(ostream& out, const PERSON& c);

//атрибуты

protected:

string name;

int age;

};

* 1. Student : public Person – класс для хранения сведений о студенте. Наследуется для класса Person.

#pragma once

#include "PERSON.h"

#include "Object.h"

//класс STUDENT наследуется от класса PERSON

class STUDENT :public PERSON

{

public:

STUDENT();//конструктор без параметров

~STUDENT();//деструктор

void Show();// функция для просмотра атрибутов класса с помощью указателя

STUDENT(string, int, int, string);//конструктор с параметрами

STUDENT(const STUDENT&);//конструктор копирования

string Get\_subject() { return subject; }//модификатор

void Set\_subject(string);//селектор

int Get\_mark() { return mark; }//модификатор

void Set\_mark(int);//селектор

STUDENT& operator=(const STUDENT&);//операция присваивания

void Check\_Mark(const STUDENT&);

friend istream& operator>>(istream& in, STUDENT& l);//операция ввода

friend ostream& operator<<(ostream& out, const STUDENT& l); //операция вывода

protected:

int mark;//поле оценки студента

string subject;//поле предмета

};

#include "Student.h"

//конструктор без параметров

STUDENT::STUDENT() :PERSON()

{

mark = 2;

}

//дестрктор

STUDENT::~STUDENT()

{

}

//конструктор с параметрами

STUDENT::STUDENT(string M, int C, int mark, string subject) :PERSON(M, C)

{

this->name = M;

this->age = C;

this->mark = mark;

this->subject = subject;

}

//конструктор копирования

STUDENT::STUDENT(const STUDENT& other)

{

this->name = other.name;

this->age = other.age;

this->mark = other.mark;

this->subject = other.subject;

}

//модификатор

void STUDENT::Set\_mark(int)

{

this->mark = mark;

}

void STUDENT::Set\_subject(string)

{

this->subject = subject;

}

//оперция присваивания

STUDENT& STUDENT::operator=(const STUDENT& l)

{

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

name = l.name;

age = l.age;

mark = l.mark;

subject = l.subject;

return \*this;

}

//операция ввода

istream& operator>>(istream& in, STUDENT& l)

{

cout << "\nName:";

in >> l.name;

cout << "\nAge:";

in >> l.age;

while (l.age < 14 || l.age > 40)

{

cout << "\nНе уверен что в таком возрасте учатся в университете\n";

cout << "\nAge:";

in >> l.age;

}

cout << "\nsubject:";

in >> l.subject;

cout << "\nmark:";

in >> l.mark;

while (l.mark < 2 || l.mark > 5)

{

cout << "\nВведена некорректная оценка!\n";

cout << "\nMark:";

in >> l.mark;

}

return in;

}

//операция вывода

ostream& operator<<(ostream& out, const STUDENT& l)

{

out << "\nName : " << l.name;

out << "\nAge : " << l.age;

out << "\nmark : " << l.mark;

out << "\nsubject:" << l.subject;

out << "\n";

return out;

}

void STUDENT::Check\_Mark(const STUDENT& stud)

{

switch (stud.mark)

{

case 2: cout << "\nОценка студента " << stud.name << " по предмету " << stud.subject << " неудовлетворительна!\n"; break;

case 3:

case 4:

case 5: cout << "\nОценка студента " << stud.name << " по предмету " << stud.subject << " удовлетворительна!\n"; break;

default:

break;

}

}

void STUDENT::Show()

{

cout << "\nName : " << name << endl;

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

cout << "\nMark : " << mark << endl;

cout << "\nSubject : " << subject << endl;

Check\_Mark(\*this);

}

* 1. Vector – класс для хранения нескольких элементов классов Person и Student. Представляет собой динамический массив для хранения вышеупомянутых классов.

#pragma once

#include "PERSON.h"

#include <string>

#include <iostream>

using namespace std;

class Vector

{

public:

Vector(); // конструктор без параметров

Vector(int); // Конструктор копирования

~Vector(); // деструктор

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

friend ostream& operator<<(ostream& out, const Vector&); // операция вывода

private:

Object\*\* first; // указатель на первый элемент вектора

int size; // размер

int current; // текущая позиция

};

#include "Vector.h"

// Конструктор без параметров

Vector::Vector()

{

first = 0;

size = 0;

current = 0;

}

// деструктор

Vector:: ~Vector()

{

if (first != 0)

{

delete[] first;

}

first = 0;

}

// конструктор с параметрами

Vector::Vector(int n)

{

first = new Object \* [n];

current = 0;

size = n;

}

// добавление объект, на который указывает указатель "p" в вектор

void Vector::Add(Object\* p)

{

if (current < size)

{

first[current] = p;

current++;

}

}

// операция вывода

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

{

if (vec.size == 0)

{

out << "Empty" << endl;

}

Object\*\* p = vec.first; // указатель на указатель типа Object

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

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

(\*p)->Show(); // Вызов метода Show() (Позднее связывание)

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

p++;

}

return out;

}

1. Организация ввода-вывода
   1. Оператор ввода cin
   2. Оператор вывода cout
   3. Перегрузка оператора << для класса VectorPerson.

// операция вывода

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

{

if (vec.size == 0)

{

out << "Empty" << endl;

}

Object\*\* p = vec.first; // указатель на указатель типа Object

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

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

(\*p)->Show(); // Вызов метода Show() (Позднее связывание)

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

p++;

}

return out;

}

* 1. Действия для решения задачи. Демонстрация работы класса в функции main().

#include <iostream>

#include "PERSON.h"

#include "Student.h"

#include "Object.h"

#include "Vector.h"

using namespace std;

int main()

{

system("chcp 1251>nul");

Vector vec(5); // вектор состоящий из 5 элементов

PERSON a;// Создание объекта класса Car

cin >> a;// Заполнение полей объекта класса Car

STUDENT b;// Создание объекта класса STUDENT

cin >> b;// Заполнение полей объекта класса STUDENT

Object\* ptr = &a;// Постановка указателя на объект класса Car

vec.Add(ptr);// Добавляем объект в вектор

ptr = &b;// Ставим указатель на объект класса STUDENT

vec.Add(ptr);// Добавляем объект класса STUDENT в вектор

cout << vec;// Вывод всех элементов вектора со всеми полями

return 0;

}

**Блок-схема**

**Sourse.cpp**



**Person.cpp**



**Student.cpp**





**Vector.cpp**



**Object.h**



**Person.h**



**Student.h**



**Vector.h**



**Код**

#include <iostream>

#include "PERSON.h"

#include "Student.h"

#include "Object.h"

#include "Vector.h"

using namespace std;

int main()

{

system("chcp 1251>nul");

Vector vec(5); // вектор состоящий из 5 элементов

PERSON a;// Создание объекта класса Car

cin >> a;// Заполнение полей объекта класса Car

STUDENT b;// Создание объекта класса STUDENT

cin >> b;// Заполнение полей объекта класса STUDENT

Object\* ptr = &a;// Постановка указателя на объект класса Car

vec.Add(ptr);// Добавляем объект в вектор

ptr = &b;// Ставим указатель на объект класса STUDENT

vec.Add(ptr);// Добавляем объект класса STUDENT в вектор

cout << vec;// Вывод всех элементов вектора со всеми полями

return 0;

}

#include "Person.h"

#include <string>

#include <iostream>

using namespace std;

PERSON::PERSON()

{

name = "";

age = 0;

}

PERSON::PERSON(string name, int age)

{

this->name = name;

this->age = age;

}

PERSON::PERSON(const PERSON& person)

{

this->age = person.age;

this->name = person.name;

}

PERSON& PERSON::operator=(const PERSON& copyObject)

{

if (©Object == this) return \*this;

age = copyObject.age;

name = copyObject.name;

return \*this;

}

void PERSON::Show()

{

cout << "Данные о человеке" << endl;

cout << "Имя " << Get\_name() << endl;

cout << "Возраст " << Get\_age() << endl;

}

istream& operator>>(istream& in, PERSON& p)

{

cout << "\nName:";

in >> p.name;

cout << "\nAge:";

in >> p.age;

return in;

}

ostream& operator<<(ostream& out, const PERSON& p)

{

out << "\nName : " << p.name;

out << "\nAge : " << p.age;

out << "\n";

return out;

}

#include "Student.h"

//конструктор без параметров

STUDENT::STUDENT() :PERSON()

{

mark = 2;

}

//дестрктор

STUDENT::~STUDENT()

{

}

//конструктор с параметрами

STUDENT::STUDENT(string M, int C, int mark, string subject) :PERSON(M, C)

{

this->name = M;

this->age = C;

this->mark = mark;

this->subject = subject;

}

//конструктор копирования

STUDENT::STUDENT(const STUDENT& other)

{

this->name = other.name;

this->age = other.age;

this->mark = other.mark;

this->subject = other.subject;

}

//модификатор

void STUDENT::Set\_mark(int)

{

this->mark = mark;

}

void STUDENT::Set\_subject(string)

{

this->subject = subject;

}

//оперция присваивания

STUDENT& STUDENT::operator=(const STUDENT& l)

{

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

name = l.name;

age = l.age;

mark = l.mark;

subject = l.subject;

return \*this;

}

//операция ввода

istream& operator>>(istream& in, STUDENT& l)

{

cout << "\nName:";

in >> l.name;

cout << "\nAge:";

in >> l.age;

while (l.age < 14 || l.age > 40)

{

cout << "\nНе уверен что в таком возрасте учатся в университете\n";

cout << "\nAge:";

in >> l.age;

}

cout << "\nsubject:";

in >> l.subject;

cout << "\nmark:";

in >> l.mark;

while (l.mark < 2 || l.mark > 5)

{

cout << "\nВведена некорректная оценка!\n";

cout << "\nMark:";

in >> l.mark;

}

return in;

}

//операция вывода

ostream& operator<<(ostream& out, const STUDENT& l)

{

out << "\nName : " << l.name;

out << "\nAge : " << l.age;

out << "\nmark : " << l.mark;

out << "\nsubject:" << l.subject;

out << "\n";

return out;

}

void STUDENT::Check\_Mark(const STUDENT& stud)

{

switch (stud.mark)

{

case 2: cout << "\nОценка студента " << stud.name << " по предмету " << stud.subject << " неудовлетворительна!\n"; break;

case 3:

case 4:

case 5: cout << "\nОценка студента " << stud.name << " по предмету " << stud.subject << " удовлетворительна!\n"; break;

default:

break;

}

}

void STUDENT::Show()

{

cout << "\nName : " << name << endl;

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

cout << "\nMark : " << mark << endl;

cout << "\nSubject : " << subject << endl;

Check\_Mark(\*this);

}

#include "Vector.h"

// Конструктор без параметров

Vector::Vector()

{

first = 0;

size = 0;

current = 0;

}

// деструктор

Vector:: ~Vector()

{

if (first != 0)

{

delete[] first;

}

first = 0;

}

// конструктор с параметрами

Vector::Vector(int n)

{

first = new Object \* [n];

current = 0;

size = n;

}

// добавление объект, на который указывает указатель "p" в вектор

void Vector::Add(Object\* p)

{

if (current < size)

{

first[current] = p;

current++;

}

}

// операция вывода

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

{

if (vec.size == 0)

{

out << "Empty" << endl;

}

Object\*\* p = vec.first; // указатель на указатель типа Object

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

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

(\*p)->Show(); // Вызов метода Show() (Позднее связывание)

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

p++;

}

return out;

}

#pragma once

class Object

{

public:

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

};

#pragma once

#include "Object.h"

#include <string>

#include <iostream>

using namespace std;

class PERSON: public Object

{

public:

//конструктор без параметров

PERSON();

//деструктор

virtual ~PERSON();

void Show();// функция для просмотра атрибутов класса с помощью указателя

//констрктор с параметрами

PERSON(string, int);

//конструктор копирования

PERSON(const PERSON&);

//Геттеры

string Get\_name()

{

return name;

}

int Get\_age()

{

return age;

}

//Сеттеры

void Set\_name(string);

void Set\_age(int);

//перегрузка операции присваивания

PERSON& operator=(const PERSON&);

//глобальные операторы-функции ввода-вывода

friend istream& operator>>(istream& in, PERSON& c);

friend ostream& operator<<(ostream& out, const PERSON& c);

//атрибуты

protected:

string name;

int age;

};

#pragma once

#include "PERSON.h"

#include "Object.h"

//класс STUDENT наследуется от класса PERSON

class STUDENT :public PERSON

{

public:

STUDENT();//конструктор без параметров

~STUDENT();//деструктор

void Show();// функция для просмотра атрибутов класса с помощью указателя

STUDENT(string, int, int, string);//конструктор с параметрами

STUDENT(const STUDENT&);//конструктор копирования

string Get\_subject() { return subject; }//модификатор

void Set\_subject(string);//селектор

int Get\_mark() { return mark; }//модификатор

void Set\_mark(int);//селектор

STUDENT& operator=(const STUDENT&);//операция присваивания

void Check\_Mark(const STUDENT&);

friend istream& operator>>(istream& in, STUDENT& l);//операция ввода

friend ostream& operator<<(ostream& out, const STUDENT& l); //операция вывода

protected:

int mark;//поле оценки студента

string subject;//поле предмета

};

#pragma once

#include "PERSON.h"

#include <string>

#include <iostream>

using namespace std;

class Vector

{

public:

Vector(); // конструктор без параметров

Vector(int); // Конструктор копирования

~Vector(); // деструктор

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

friend ostream& operator<<(ostream& out, const Vector&); // операция вывода

private:

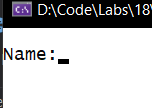
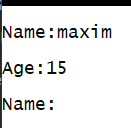
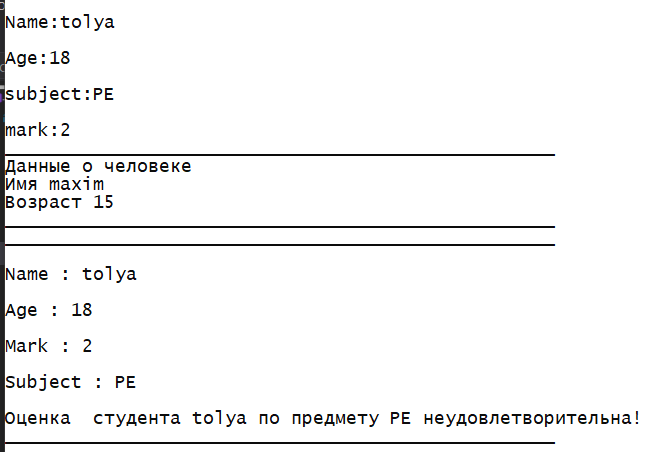
Object\*\* first; // указатель на первый элемент вектора

int size; // размер

int current; // текущая позиция

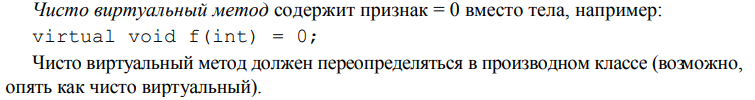
};

**Скриншоты**

1. 
2. 
3. 

**Ответы на контрольные вопросы**

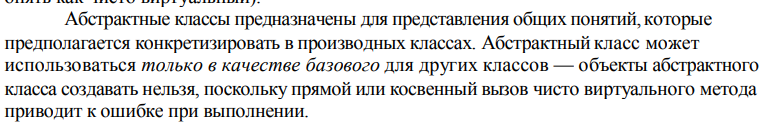


1. 



2. 



3. 



4. Полиморфные функции – это функции, которые работают с объектом любого типа в пределах одной иерархии.



5. При полиморфизме невозможно создать объект, так как базовым классом является абстрактный, когда в принципе подстановки используется наследование, что означает, что везде, где может быть использован объект базового класса, может быть и использован объект производного класса.



6. Пример: класс собака и класс кошка имеют одну из функций: говорить, но делают они это по-разному, так как собака гавкает, а кошка мяукает.



7. Метод Show, который наследуется от абстрактного класса, для вывода на консоль объектов разных классов.