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: «Информатика»

Тема:

Вариант 13

Выполнил:

Студент группы ИВТ-20-2б

Сафронов Владислав Владиславович

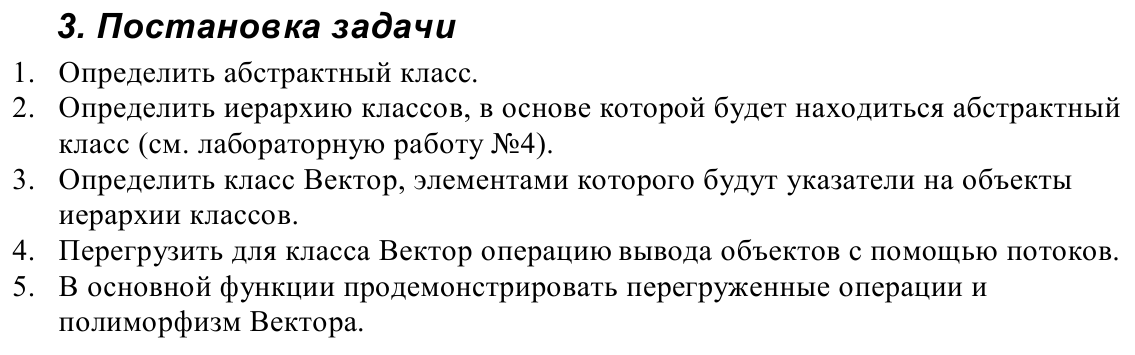
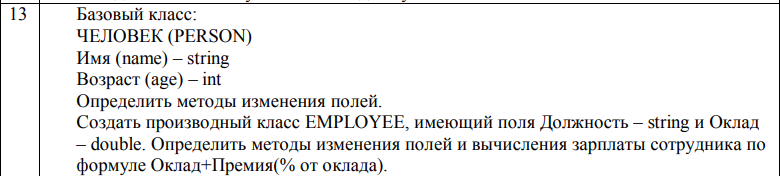
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

**Пермь, 2021**

**Постановка задачи**

****

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Разработать класс Object – базовый класс, который содержит чисто виртуальную функцию Show для вывода экземпляров классов-наследников в консоль.
   2. Разработать класс Person для хранения сведений о человеке.
   3. Разработать класс Employee для хранения сведений о работнике. Наследуется от класса Person.
   4. Разработать класс Vector с полями Object\*\* begin\_p и int len.
   5. Разработать необходимые методы для ввода данных в поля классов, конструкторы, деструкторы.
   6. Организовать перегрузку операторов >>, <<, =.
2. В ходе работы были использованы следующие типы данных:
   1. Класс Object, который содержит чисто виртуальную функцию Show для вывода экземпляров классов-наследников в консоль.
3. class Object
4. {
5. public:
6. Object() {}
7. ~Object() {}
8. virtual void Show() = 0;
9. };
   1. Класс Person, наследуется от класса Object, хранит сведения о человеке. Имеет 2 поля: строковое поле name хранит ФИО, целочисленное поле age – возраст.

class Person : public Object

{

string name;

int age;

* + 1. Конструкторы класса Person.

1. Person()
2. {
3. name = "None";
4. age = 0;
5. }
6. Person(string name, int age)
7. {
8. this->name = name;
9. this->age = age;
10. }
11. Person(const Person& obj)
12. {
13. name = obj.name;
14. age = obj.age;
15. }
    * 1. Деструктор описывать явно не требуется, так как в классе не используется динамическая память.
16. ~Person() {}
    * 1. Сеттеры для присвоения значений полям класса Person.
17. void SetName(string a)
18. {
19. this->name = a;
20. }
21. void SetAge(int n)
22. {
23. this->age = n;
24. }
    * 1. Геттеры для возвращения значений полей класса Person.
25. string GetName() { return name; }
26. int GetAge() { return age; }
    * 1. Метод Show для вывода значений полей в консоль.
27. void Show()
28. {
29. cout << "ФИО: " << name << endl;
30. cout << "Возраст: " << age << endl;
31. }
    * 1. Перегрузка операции присваивания = для класса Person.
32. Person& Person::operator=(Person& obj)
33. {
34. if (this == &obj) return \*this;
35. name = obj.name;
36. age = obj.age;
37. return \*this;
38. }
    * 1. Перегрузка операторов << и >> для класса Person.
39. istream& operator>> (istream& cin, Person& obj)
40. {
41. cout << "Введите ФИО: ";
42. getline(cin, obj.name);
43. cout << "Введите возраст: ";
44. cin >> obj.age;
45. return cin;
46. }
47. ostream& operator<< (ostream& cout, Person& obj)
48. {
49. return (cout << "Имя: " << obj.name << endl << "Возраст: " << obj.age << endl);
50. }
    1. Класс Employee наследуется от класса Person и имеет 2 поля: строковое поле position хранит должность работника, вещественное поле salary хранит оклад.
51. class Employee : public Person
52. {
53. string position;
54. double salary;
    * 1. Конструкторы класса Employee
55. Employee() :Person(), position("None"), salary(0) {}
56. Employee(string pos, double sal) :Person(), position(pos), salary(sal) {}
57. Employee(string pos) :Person(), position(pos), salary(0) {}
58. Employee(string name, int age, string pos, double sal) :Person(name, age), position(pos), salary(sal) {}
59. Employee(double sal) :Person(), salary(sal) {}
60. Employee(Person p, string pos, double sal) :Person(p), position(pos), salary(sal) {}
    * 1. Сеттеры и геттеры для изменения и возвращения значений полей класса.
61. void SetPosition(string pos) { this->position = pos; }
62. void SetSalary(double sal) { this->salary = sal; }
63. string GetPosition() { return position; }
64. double GetSalary() { return salary; }
    * 1. Метод GetPay возвращает вещественное число: оклад + премия работника. Премия составляет 25% от оклада.
65. double GetPay() { return salary \* 1.25; }
    * 1. Метод All\_Info для вывода значений полей класса Employee. Сначала вызывается метод Show класса Person, затем выводятся поля класса Employee, а также оклад + премия.
66. void All\_Info()
67. {
68. this->Show();
69. cout << "Должность: " << position << endl;
70. cout << "Оклад: " << salary << endl;
71. cout << "Оклад + премия: " << GetPay() << endl;
72. }
    * 1. Деструктор описывать явно не требуется, так как в классе не используется динамическая память.
73. ~Employee() {}
    * 1. Перегрузка оператора присваивания = для класса Employee
74. Employee& operator=(Person& p)
75. {
76. this->SetName(p.GetName());
77. this->SetAge(p.GetAge());
78. return \*this;
79. }
80. };
    1. Класс Vector с полями Object\*\* begin\_p и int len.
81. class Vector
82. {
83. Object\*\* begin\_p = nullptr;
84. int len = 0;
    * 1. Конструкторы для класса Vector
85. Vector()
86. {
87. begin\_p = nullptr;
88. len = 0;
89. }
90. Vector(int size)
91. {
92. begin\_p = new Object \* [size];
93. len = size;
94. }
95. Vector(Vector& tmp)
96. {
97. if (begin\_p != nullptr) delete[] begin\_p;
98. else
99. {
100. int size = tmp.size();
101. begin\_p = new Object \* [size];
102. for (int i = 0; i < size; i++)
103. {
104. begin\_p[i] = tmp.begin\_p[i];
105. }
106. }
107. }
     * 1. Деструктор класса Vector.
108. ~Vector()
109. {
110. if (begin\_p != nullprt)
111. {
112. delete[] begin\_p;
113. begin\_p = nullptr;
114. }
115. }
     * 1. Метод size выводит поле, хранящее размер вектора.
116. int size() { return len; }
     * 1. Метод push\_back для добавления элемента в вектор.
117. void push\_back(Object\* a)
118. {
119. Object\*\* tmp = new Object \* [len + 1];
120. for (int i = 0; i < len; i++)
121. {
122. tmp[i] = begin\_p[i];
123. }
124. tmp[len] = a;
125. delete[] begin\_p;
126. begin\_p = tmp;
127. len++;
129. tmp = nullptr;
130. }
     * 1. Перегрузка оператора << для класса Vector
131. osream& operator << (ostream& out, Vecotor& lst)
132. {
133. if (lst.len == 0) return out << "Список пуст\n";
134. Object\*\* tmp = lst.begin\_p;
135. for (int i = 0; i < lst.len; i++)
136. {
137. out << "Элемент " << (i + 1) << endl;
138. (\*tmp)->Show();
139. tmp++;
140. }
141. return out;
142. }

**Блок-схема программы**

**Код программы**

**Файл Object.h:**

1. #pragma once
2. #include <iostream>
3. using namespace std;
5. class Object
6. {
7. public:
8. Object() {}
9. ~Object() {}
10. virtual void Show() = 0;
11. };

**Файл Person.h:**

1. #pragma once
2. #include <iostream>
3. #include <string>
4. #include "Object.h"
5. using namespace std;
7. class Person : public Object
8. {
9. string name;
10. int age;
11. public:
12. Person()
13. {
14. name = "None";
15. age = 0;
16. }
17. Person(string name, int age)
18. {
19. this->name = name;
20. this->age = age;
21. }
22. Person(const Person& p)
23. {
24. this->name = p.name;
25. this->age = p.age;
26. }
28. ~Person() {}
30. void SetName(string name)
31. {
32. this->name = name;
33. }
34. void SetAge(int age)
35. {
36. this->age = age;
37. }
38. string GetName() { return name; }
39. int GetAge() { return age; }
41. void Show() override
42. {
43. cout << "Имя: " << name << endl;
44. cout << "Возраст: " << age << endl;
45. }
47. Person& operator=(Person& p)
48. {
49. if (this == &p) return \*this;
50. this->name = p.name;
51. this->age = p.age;
52. return \*this;
53. }
55. friend istream& operator >> (istream& in, Person& p);
56. friend ostream& operator << (ostream& out, Person& p);
57. };
59. istream& operator >> (istream& in, Person& p)
60. {
61. cout << "Введите ФИО: ";
62. getline(in, p.name);
63. cout << "Введите возраст: ";
64. in >> p.age;
65. return in;
66. }
68. ostream& operator << (ostream& out, Person& p)
69. {
70. return out << "ФИО: " << p.name << endl << "Возраст: " << p.age << endl;
71. }

**Файл Employee.h:**

1. #pragma once
2. #include <iostream>
3. #include "Person.h"
4. using namespace std;
6. class Employee : public Person
7. {
8. string speciality;
9. double salary;
10. public:
11. Employee() :Person(), speciality("None"), salary(0) {}
12. Employee(string pos, double sal) :Person(), speciality(pos), salary(sal) {}
13. Employee(string pos) :Person(), speciality(pos), salary(0) {}
14. Employee(string name, int age, string pos, double sal) :Person(name, age), speciality(pos), salary(sal) {}
15. Employee(double sal) :Person(), salary(sal) {}
16. Employee(Person p, string pos, double sal) :Person(p), speciality(pos), salary(sal) {}
18. void Setspeciality(string pos) { this->speciality = pos; }
19. void SetSalary(double sal) { this->salary = sal; }
20. string Getspeciality() { return speciality; }
21. double GetSalary() { return salary; }
23. double GetPay() { return salary \* 1.25; }
25. void All\_Info()
26. {
27. this->Show();
28. cout << "Должность: " << speciality << endl;
29. cout << "Оклад: " << salary << endl;
30. cout << "Оклад + премия: " << GetPay() << endl;
31. }
33. ~Employee() {}
35. Employee& operator=(Person& p)
36. {
37. this->SetName(p.GetName());
38. this->SetAge(p.GetAge());
39. return \*this;
40. }
42. };

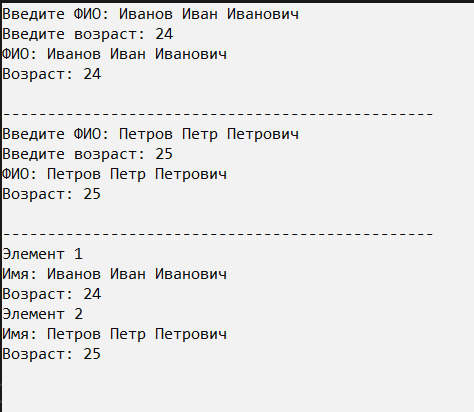
**Файл Vector.h:**

1. #pragma once
2. #include <iostream>
3. #include "Object.h"
4. using namespace std;
6. class Vector
7. {
8. Object\*\* begin\_p = nullptr;
9. int len = 0;
10. public:
11. Vector()
12. {
13. begin\_p = nullptr;
14. len = 0;
15. }
16. Vector(int size)
17. {
18. begin\_p = new Object \* [size];
19. len = size;
20. }
21. Vector(Vector& tmp)
22. {
23. if (begin\_p != nullptr) delete[] begin\_p;
24. else
25. {
26. int size = tmp.size();
27. begin\_p = new Object \* [size];
28. for (int i = 0; i < size; i++)
29. {
30. begin\_p[i] = tmp.begin\_p[i];
31. }
32. }
33. }
35. ~Vector()
36. {
37. if (begin\_p != nullptr)
38. {
39. delete[] begin\_p;
40. begin\_p = nullptr;
41. }
42. }
44. int size() { return len; }
45. void push\_back(Object\* a)
46. {
47. Object\*\* tmp = new Object \* [len + 1];
48. for (int i = 0; i < len; i++)
49. {
50. tmp[i] = begin\_p[i];
51. }
52. tmp[len] = a;
53. delete[] begin\_p;
54. begin\_p = tmp;
55. len++;
57. tmp = nullptr;
58. }
60. friend ostream& operator << (ostream& out, Vector& lst);
61. };
63. ostream& operator << (ostream& out, Vector& lst)
64. {
65. if (lst.len == 0) return out << "Список пуст\n";
66. Object\*\* tmp = lst.begin\_p;
67. for (int i = 0; i < lst.len; i++)
68. {
69. out << "Элемент " << (i + 1) << endl;
70. (\*tmp)->Show();
71. tmp++;
72. }
73. return out;
74. }

**Файл main.cpp:**

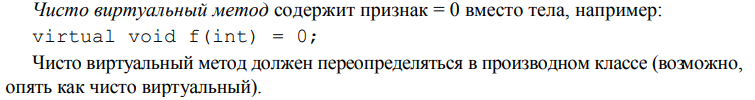
1. #include <iostream>
2. #include "Object.h"
3. #include "Person.h"
4. #include "Employee.h"
5. #include "Vector.h"
6. using namespace std;
8. int main()
9. {
10. setlocale(LC\_ALL, "Russian");
11. system("chcp 1251>nul");
13. Person pers;
14. cin >> pers;
15. cout << pers;
16. cout << "\n------------------------------------------------\n";
17. Employee emp;
18. cin >> emp;
19. cout << emp;
20. cout << "\n------------------------------------------------\n";
21. Object\* p1 = &pers;
22. Person\* p2 = &emp;
24. Vector list;
25. list.push\_back(p1);
26. list.push\_back(p2);
27. cout << list << endl;
28. cout << "\n------------------------------------------------\n";
29. }

**Скриншоты результатов работы программы**

****

**Ответы на вопросы**

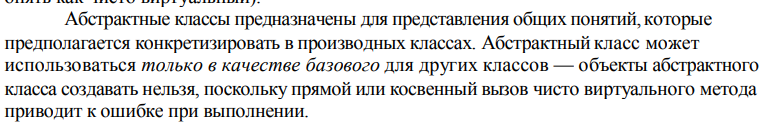


1. 



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.