Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

**Тема:** Лабораторная работа № 8 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Солодов Александр Андреевич

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**

1. Определить иерархию пользовательских классов (см. лабораторную работу №5). Во главе иерархии должен стоять абстрактный класс с чисто виртуальными методами для ввода и вывода информации об атрибутах объектов.
2. Реализовать конструкторы, деструктор, операцию присваивания, селекторы и модификаторы.
3. Определить класс-группу на основе структуры, указанной в варианте.
4. Для группы реализовать конструкторы, деструктор, методы для добавления и удаления элементов в группу, метод для просмотра группы, перегрузить операцию для получения информации о размере группы.
5. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.
6. Добавить методы для обработки событий группой и объектами пользовательских классов.
7. Написать тестирующую программу.
8. Нарисовать диаграмму классов и диаграмму объектов.

*Вариант 12*

Базовый класс: ЧЕЛОВЕК (Person)

Имя – string

Возраст – int

Производный класс: СОТРУДНИК (Employee)

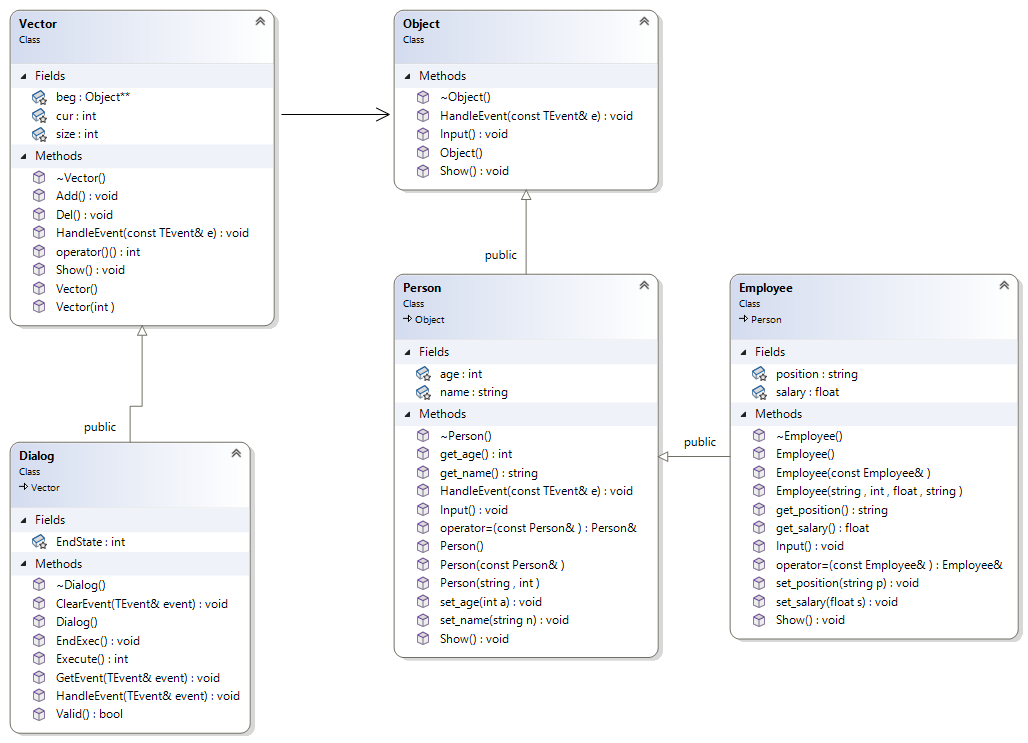
Заработная плата – float

Должность - string

Группа – Дерево (Tree).

Команды:

* Создать группу (формат команды: m количество элементов группы)
* Добавить элемент в группу (формат команды: +)
* Удалить элемент из группы (формат команды -)
* Вывести информацию об элементах группы (формат команды: s)
* Вывести информацию об имени элемента с номером k (формат команды : z k, где k – целое число)
* Конец работы (формат команды: q)

**UML-диаграмма**

**Программное решение**

Object.h

#pragma once

#include "Event.h"

class Object

{

public:

Object();

virtual void Show() = 0;

virtual void Input() = 0;

virtual ~Object();

virtual void HandleEvent(const TEvent& e) = 0;

};

Object.cpp

#include "Object.h"

Object::Object() {}

Object::~Object() {}

Person.h

#pragma once

#include <iostream>

#include <string>

#include "Object.h"

using namespace std;

class Person : public Object

{

protected:

string name;

int age;

public:

Person();

Person(string, int);

Person(const Person&);

~Person();

void Show();

void Input();

string get\_name() { return name; }

void set\_name(string n) { name = n; }

int get\_age() { return age; }

void set\_age(int a) { age = a; }

Person& operator = (const Person&);

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

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

void HandleEvent(const TEvent& e);

};

Person.cpp

#include "Person.h"

Person::Person()

{

name = "";

age = 0;

}

Person::Person(string n, int a)

{

name = n;

age = a;

}

Person::Person(const Person& p)

{

name = p.name;

age = p.age;

}

Person::~Person() {}

void Person::Show() {

cout << "Имя: " << name << endl;

cout << "Возраст: " << age << endl;

}

void Person::Input() {

cout << "\nВведите имя: ";

cin >> name;

cout << "Введите возраст: ";

cin >> age;

}

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

if (&p == this)

return \*this;

name = p.name;

age = p.age;

return \*this;

}

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

{

cout << "\nВведите имя: ";

in >> p.name;

cout << "Введите возраст: ";

in >> p.age;

return in;

}

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

{

out << p.name << " | " << p.age << endl;

return out;

}

void Person::HandleEvent(const TEvent& e) {}

Employee.h

#pragma once

#include "Person.h"

class Employee : public Person {

protected:

float salary;

string position;

public:

Employee();

Employee(string, int, float, string);

Employee(const Employee&);

~Employee();

//методы класса

float get\_salary() { return salary; }

void set\_salary(float s) { salary = s; }

string get\_position() { return position; }

void set\_position(string p) { position = p; }

void Show();

void Input();

Employee& operator = (const Employee&);

//перегрузка операций ввода и вывода

friend istream& operator >> (istream& in, Employee& s);

friend ostream& operator << (ostream& out, const Employee& s);

};

Employee.cpp

#include "Employee.h"

Employee::Employee() : Person()

{

salary = 0;

position = "";

}

Employee::Employee(string n, int a, float s, string p) : Person(n, a)

{

salary = s;

position = p;

}

Employee::Employee(const Employee& e)

{

name = e.name;

age = e.age;

salary = e.salary;

position = e.position;

}

Employee::~Employee() {}

void Employee::Show() {

cout << "Имя: " << name << endl;

cout << "Возраст: " << age << endl;

cout << "Заработная плата: " << salary << endl;

cout << "Должность: " << position << endl;

}

void Employee::Input() {

cout << "\nВведите имя: ";

cin >> name;

cout << "Введите возраст: ";

cin >> age;

cout << "Введите зарплату: ";

cin >> salary;

cout << "Введите должность";

cin >> position;

}

Employee& Employee::operator = (const Employee& e) {

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

name = e.name;

age = e.age;

salary = e.salary;

position = e.position;

return \*this;

}

istream& operator >> (istream& in, Employee& e)

{

cout << "\nВведите имя: ";

in >> e.name;

cout << "Введите возраст: ";

in >> e.age;

cout << "Введите зарплату: ";

in >> e.salary;

cout << "Введите должность";

in >> e.position;

return in;

}

ostream& operator << (ostream& out, const Employee& e)

{

out << e.name << " | " << e.age << " | " << e.salary << " | " << e.position << endl;

return out;

}

Vector.h

#pragma once

#include "Object.h"

#include "Event.h"

class Vector

{

protected:

Object\*\* beg;

int size;

int cur;

public:

Vector();

Vector(int);

~Vector();

void Add();

void Del();

void Show();

void HandleEvent(const TEvent& e);

int operator () ();

};

Vector.cpp

#include "Vector.h"

#include "Employee.h"

#include "Event.h"

#include <iostream>

using namespace std;

Vector::Vector() {

beg = nullptr;

cur = 0;

size = 0;

}

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

Vector::~Vector()

{

if (beg != 0)

delete[] beg;

beg = 0;

}

void Vector::Add()

{

Object\* p;

cout << "1. Человек" << endl;

cout << "2. Сотрудник" << endl;

int y;

cin >> y;

if (y == 1) {

Person\* a = new Person;

a->Input();

p = a;

if (cur < size) {

beg[cur] = p;

cur++;

}

}

else {

if (y == 2) {

Employee\* b = new Employee;

b->Input();

p = b;

if (cur < size) {

beg[cur] = p;

cur++;

}

}

else

return;

}

}

void Vector::Del()

{

if (cur == 0)

return;

cur--;

}

void Vector::Show()

{

if (cur == 0)

cout << "Вектор пустой" << endl;

Object\*\* p = beg;

for (int i = 0; i < cur; i++) {

(\*p)->Show();

p++;

}

}

int Vector::operator () ()

{

return size;

}

void Vector::HandleEvent(const TEvent& e)

{

if (e.what == evMessage)

{

Object\*\* p = beg;

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

{

(\*p)->HandleEvent(e);

p++;

}

switch (e.command)

{

case cmGetName:

int k;

cin >> k;

beg[k - 1]->Show();

break;

}

}

}

Dialog.h

#pragma once

#include "Vector.h"

#include "Event.h"

class Dialog : public Vector

{

protected:

int EndState;

public:

Dialog(void);

virtual ~Dialog(void);

virtual void GetEvent(TEvent& event);

virtual int Execute();

virtual void HandleEvent(TEvent& event);

virtual void ClearEvent(TEvent& event);

bool Valid();

void EndExec();

};

Dialog.cpp

#include "Dialog.h"

#include "Event.h"

#include "Vector.h"

#include <iostream>

using namespace std;

Dialog::Dialog() : Vector()

{

EndState = 0;

}

Dialog::~Dialog() {}

void Dialog::GetEvent(TEvent& event)

{

string OpInt = "m+-szq ";

string s;

string param;

char code;

cout << "Введите команду > ";

cin >> s;

code = s[0];

if (OpInt.find(code) >= 0) {

event.what = evMessage;

switch (code) {

case 'm': event.command = cmMake; break;

case '+': event.command = cmAdd; break;

case '-': event.command = cmDel; break;

case 's': event.command = cmShow; break;

case 'z': event.command = cmGetName; break;

case 'q': event.command = cmQuit; break;

}

//Выделяем параметры команды, если они есть

if (s.length() > 1)

{

param = s.substr(1, s.length() - 1);

int A = atoi(param.c\_str()); //Преобразуем параметр в число

event.a = A; //Записываем в сообщение

}

}

else event.what = evNothing; //Пустое событие

}

int Dialog::Execute()

{

TEvent event;

do {

EndState = 0;

GetEvent(event); //получить событие

HandleEvent(event); //обработать событие

} while (!Valid());

return EndState;

}

bool Dialog::Valid()

{

if (EndState == 0)

return 0;

else

return 1;

}

void Dialog::ClearEvent(TEvent& event)

{

event.what = evNothing; //пустое событие

}

void Dialog::EndExec()

{

EndState = 1;

}

//обработчик событий

void Dialog::HandleEvent(TEvent& event)

{

if (event.what == evMessage)

{

switch (event.command)

{

case cmMake: //создание группы

size = event.a; //размер группы

beg = new Object \* [size]; //выделяем память под массив указателей

cur = 0; //текущая позиция

ClearEvent(event); //очищаем событие

break;

case cmAdd: //добавление

Add();

ClearEvent(event);

break;

case cmDel:

Del(); //удаление

ClearEvent(event);

break;

case cmShow:

Show(); //просмотр

ClearEvent(event);

break;

case cmQuit:

EndExec(); //выход

ClearEvent(event);

break;

default: Vector::HandleEvent(event);

}

}

}

Event.h

#pragma once

const int evNothing = 0;

const int evMessage = 100;

const int cmAdd = 1;

const int cmDel = 2;

const int cmGetName = 3;

const int cmShow = 4;

const int cmMake = 6;

const int cmQuit = 101;

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

main.cpp

/\* Вариант 12.

Базовый класс: ЧЕЛОВЕК (Person)

Имя – string

Возраст – int

Производный класс СОТРУДНИК (Employee)

Заработная плата – float

Должность - string

Группа – Дерево (Tree). Команды:

• Создать группу (формат команды: m количество элементов группы).

• Добавить элемент в группу (формат команды: +)

• Удалить элемент из группы (формат команды -)

• Вывести информацию об элементах группы (формат команды: s)

• Вывести информацию об имени элемента с номером k (формат команды : z k, где k – целое число)

• Конец работы (формат команды: q) \*/

#include <iostream>

#include "Person.h"

#include "Employee.h"

#include "Vector.h"

#include "Dialog.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "RU");

cout << "Введите команду > " << endl;

cout << "m(число элементов): создать группу" << endl;

cout << "+: добавить элемент в группу" << endl;

cout << "-: удалить элемент из группы" << endl;

cout << "s: вывести информацию об элементах группы" << endl;

cout << "z k: вывести информацию об имени элемента с номером k" << endl;

cout << "q: конец работы " << endl << endl;

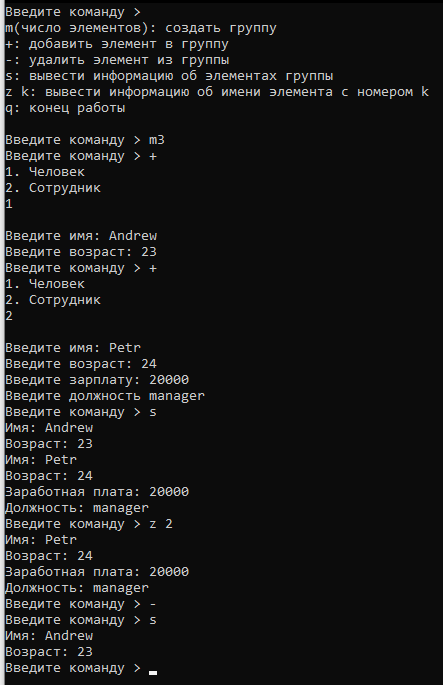
Dialog D;

D.Execute();

return 0;

}

**Результат работы программы**



**Контрольные вопросы**

1. *Что такое класс-группа? Привести примеры таких классов.*

Группа – это объект, в который включены другие объекты. Объекты, входящие в группу, называются элементами группы. Элементы группы, в свою очередь, могут быть группой.

*Примеры:*

1. Дом, состоящий из окон дверей и т.д.
2. Огород, состоящий из растений, системы полива и плана выращивания.
3. Человек, состоящий из внутренних органов, внешних данных.
4. *Привести пример описания класса-группы Список (List).*

class List {

private:

Object\* c1;//Указатель на голову списка

int size;//Размер списка

};

1. *Привести пример конструктора (с параметром, без параметров, копирования) для класса-группы Список.*

//Конструктор без параметров

List::List() {

c1 = NULL;

size = 0;

}

//Конструктор с параметрами

List::List(int n) {

if (n < 1) {

return;

}

else {

size = n;

Object\* buff = new Object;//Первый элемент списка

buff->v1 = NULL;

buff->cc = 0;

c1 = buff;

for (int i = 1; i < n; i++) {//СОЗДАНИЕ СЛЕДУЮЩИХ УЗЛОВ СПИСКА

Object\* buff = new Object;

buff->v1 = c1;

buff->cc = 0;

c1 = buff;

}

}

}

//Конструктор копирования

List::List(const List& v1) {

size = v1.size;

c1 = v1.c1;

}

1. *Привести пример деструктора для класса-группы Список.*

List::~List() {

for (int s = 0; s < size; s++) {

Object\* b1 = c1;

c1 = c1->v1;

delete b1;

}

size = 0;

}

1. *Привести пример метода для просмотра элементов для класса-группы Список.*

int List::check(int vv) {

if (vv >= size) {

return 0;

}

else {

Object\* b1 = c1;//Буферный указатель на узел списка

for (int i = 0; i < vv; i++) {//Перевод указателя на нужный элемент

b1->v1;

}

return b1->cc;//Возвращает поле найденного объекта

}

}

1. *Какой вид иерархии дает группа?*

Группа дает иерархию объектов(иерархию типа целое/часть), построенную на основе агрегации, первый вид иерархия классов, построенная на основе наследования.

1. *Почему во главе иерархии классов, содержащихся в группе объектов должен находиться абстрактный класс?*

Абстрактный класс позволяет определять методы индивидуально для каждого производного класса.

1. *Что такое событие? Для чего используются события?*

Событие с точки зрения языка С++ – это объект, отдельные поля которого характеризуют те или иные свойства передаваемой информации.

События нужны для более интерактивного взаимодействия программы и пользователя(создание окон, меню и пр.), независимо от действий, которые будут связаны с ним.

1. *Какие характеристики должно иметь событие-сообщение?*

Command – код команды, которую необходимо выполнить при появлении данного события;

Передаваемая с событием информация (сообщение).

1. *Привести пример структуры, описывающей событие.*

struct TEvent

{

int what;//тип события

union

{

int command;//код команды

int message;//Сообщение события

};

};

*11. Задана структура события*

struct TEvent

{

int what;

union

{

MouseEventType mouse;

KeyDownEvent keyDown;

MessageEvent message;

}

};

*Какие значения, и в каких случаях присваиваются полю what?*

What задаёт тип события.

Существует два типа событий evNothing – это пустое событие и evMessage – это событие-сообщение от объекта.

* evNothing присваеивается what, после обработки очередного события, а также what изначально присвоено evNothing.
* evMessage присваеивается what, когда в обработку поступает событие.

*12. Задана структура события*

struct TEvent

{

int what;//тип события

union

{

int command;//код комманды

struct//параметры команды

{

int message;

int a;

};

};

};

*Какие значения, и в каких случаях присваиваются полю command?*

Command будет иметь значение равное какому-нибудь событию, то есть, например, программист задал три события (Eventzero=0, Eventone=1, Eventtwo=2), то command может иметь значение 1,2,3.

*13. Задана структура события*

struct TEvent

{

int what;//тип события

union

{

int command;//код команды

struct//параметры команды

{

int message;

int a;

};

};

};

*Для чего используются поля a и message?*

Message нужно для определения сообщения определённого события, а “a” это параметр определённого события.

1. *Какие методы необходимы для организации обработки сообщений?*
2. Нужен метод формирования событий.
3. Главный цикл обработки событий. Он постоянно получает

событие путем вызова метода формирования событий и обрабатывает их с помощью обработчика событий. Этот цикл завершается, когда поступит событие «конец».

1. Обработчик событий. Обрабатывает каждое событие нужным для

него образом. Если объект должен обрабатывать определенное событие (сообщение), то его «обработчик» должен распознавать это событие и реагировать на него должным образом. Событие может распознаваться, например, по коду команды (поле command).

1. Метод очистки событий. Когда событие обработано, то его нужно очистить, чтобы оно не обрабатывалось далее.
2. *Какой вид имеет главный цикл обработки событий-сообщений?*

int TMyApp::Execute()

{

do

{

endState = 0;

GetEvent(event); //получить событие

HandleEvent(event); //обработать событие

//событие осталось не обработано

if (event.what != evNothing)

EventError(event);

} while (!Valid());

return endState;

}

1. *Какую функцию выполняет метод ClearEvent()? Каким образом?*

ClearEvent очищает событие, когда оно обработано, чтобы оно не обрабатывалось далее. Присваивает what пустое событие.

1. *Какую функцию выполняет метод HandleEvent ()?Каким образом?*

Обрабатывает каждое событие нужным для него образом. HandleEvent распознает, например, по коду команды (поле command), какое действие ему нужно совершить и выполняет его.

1. *Какую функцию выполняет метод GetEvent ()?*

Формирование событий.

1. *Для чего используется поле EndState? Какой класс (объект) содержит это поле?*

Используется, как флаг окончания диалога. Его содержит класс Dialog.

1. *Для чего используется функция Valid()?*

Для проверки поля EndState, или же проверки конца диалога.