Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 - «Информатика и вычислительная техника»

**О Т Ч Е Т**

**по лабораторной работе “Классы” №8**

**по дисциплине**

**«Основы алгоритмизации и программирования» семестр 2**

Выполнил студент гр. ИВТ-21-1б

Ипатов Дмитрий Сергеевич

Проверил:

Ст. Преподаватель кафедры ИТАС

Яруллин Д.В.

(оценка) (подпись)

г. Пермь-2022

**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.

**Анализ задачи:**

При ОО подходе к проектированию и программированию часто бывает удобно рассматривать программу как набор независимых объектов, обменивающихся сообщениями, которые называют событиями. Особенно это удобно при организации обмена сообщениями с пользователем. В этом случае источниками сообщений могут служить внешние устройства: мышь, клавиатура. События лучше всего представить себе как пакеты информации, которыми обмениваются объекты и которые создаются объектно-ориентированной средой в ответ на те или иные действия пользователя. Нажатие на клавишу или манипуляция мышь порождают событие, которое передается по цепочке объектов, пока не найдется объект, знающий, как обрабатывать это событие. Для того чтобы событие могло передаваться от объекта к объекту, все объекты программы должны быть объединены в группу. Отсюда следует, что прикладная программа должна быть объектом-группой, в которую должны быть включены все объекты, используемые в программе.

**Код программы:**

#include <iostream>

#include <windows.h>

using namespace std;

const int evNothing = 0;

const int evMessage = 100;

const int cmAdd = 1;

const int cmDel = 2;

const int cmGet = 3;

const int cmShow = 4;

const int cmMake = 6;

const int cmQuit = 101;

class Object

{

public:

~Object(void) {};

public:

Object(void){};

virtual int AvgAge() = 0;

virtual void Show() = 0;

virtual void Input() = 0;

};

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

class Person :public Object

{

protected:

string name;

int age;

public:

Person(void);

Person(string, int);

int AvgAge();

Person(const Person&);

~Person(void);

void Show();

void Input();

string GetName() { return name; }

int GetAge() { return age; }

void SetName(string NewName) { name = NewName; }

void SetAge(int NewAge) { age = NewAge; }

Person& operator=(const Person&);

};

int Person::AvgAge()

{

return age;

}

void Person::Show()

{

cout << "Имя человека: " << name;

cout << "\nВозраст человека: " << age;

cout << "\n\n";

}

void Person::Input()

{

cout << "Имя человека? "; cin >> name;

cout << "\nВозраст человека? "; cin >> age;

cout << '\n';

}

Person::Person(void)

{

name = "-";

age = 0;

}

Person::Person(string name, int age)

{

this->name = name;

this->age = age;

}

Person::Person(const Person& p)

{

this->name = p.name;

this->age = p.age;

}

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

{

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

name = p.name;

age = p.age;

return \*this;

}

Person::~Person(void) = default;

class Abiturient :public Person

{

protected:

int points;

string prof;

public:

Abiturient(void);

Abiturient(string, int, int, string);

Abiturient(const Abiturient&);

~Abiturient(void);

int AvgAge();

string GetName() { return name; }

int GetAge() { return age; }

int GetPoints() { return points; }

string GetProf() { return prof; }

void Show();

void Input();

void SetName(string NewName) { name = NewName; }

void SetAge(int NewAge) { age = NewAge; }

void SetPoints(int NewPoints) { points = NewPoints; }

void SetProf(string NewProf) { prof = NewProf; }

Abiturient& operator=(const Abiturient&);

};

int Abiturient::AvgAge()

{

return age;

}

void Abiturient::Show()

{

cout << "Имя абитуриента: " << name;

cout << "\nВозраст абитуриента: " << age;

cout << "\nКол-во баллов: " << points;

cout << "\nСпециальность: " << prof;

cout << '\n';

}

void Abiturient::Input()

{

cout << "Имя абитуриента? "; cin >> name;

cout << "\nВозраст абитуриента? "; cin >> age;

cout << "\nКол-во баллов? "; cin >> points;

cout << "\nСпециальность? "; cin >> prof;

cout << '\n';

}

Abiturient::Abiturient(void)

{

this->name = "-";

this->age = 0;

this->points = 0;

this->prof = "-";

}

Abiturient::Abiturient(string Name, int Age, int Points, string Prof)

{

this->name = Name;

this->age = Age;

this->points = Points;

this->prof = Prof;

}

Abiturient::Abiturient(const Abiturient& p)

{

name = p.name;

age = p.age;

points = p.points;

prof = p.prof;

}

Abiturient::~Abiturient(void) = default;

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

{

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

name = p.name;

age = p.age;

points = p.points;

prof = p.prof;

return \*this;

}

class Vector

{

protected:

Object\*\* beg;

int size;

int cur;

public:

Vector(int);

Vector(void);

~Vector(void);

void AvgAge();

void HandleEvent(const TEvent&);

void Add();

void Del();

void Show();

int operator() ();

};

void Vector::AvgAge()

{

int a=0;

if (cur == 0) cout << "Вектор пуст!";

Object\*\* p = beg;

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

{

a+=(\*p)->AvgAge();

p++;

}

cout << "Средний возраст людей в списке: " << a / cur << '\n';

}

void Vector::HandleEvent(const TEvent& event)

{

if (event.what == evMessage)

{

Object\*\* p = beg;

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

{

(\*p)->Show();

++p;

}

}

}

Vector::Vector(void)

{

beg = 0;

size = 0;

cur = 0;

}

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

Vector::~Vector(void)

{

if (beg != 0) delete[] beg;

beg = 0;

}

void Vector::Add()

{

Object\* p;

cout << "1.Person\n";

cout << "2.Abiturient\n";

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)

{

Abiturient\* b = new Abiturient;

b->Input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else return;

}

}

void Vector::Show()

{

if (cur == 0) cout << "Вектор пуст!";

Object\*\* p = beg;

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

{

(\*p)->Show();

p++;

}

}

int Vector::operator()()

{

return cur;

}

void Vector::Del()

{

if (cur == 0) return;

cur--;

}

class Dialog :public Vector

{

protected:

int EndState;

public:

Dialog();

~Dialog();

void ClearEvent(TEvent&);

int Valid();

void EndExec();

void GetEvent(TEvent&);

void HandleEvent(TEvent&);

int Execute();

};

Dialog::Dialog() :Vector()

{

EndState = 0;

}

Dialog::~Dialog() {}

void Dialog::ClearEvent(TEvent& event)

{

event.what = evNothing;

}

int Dialog::Valid()

{

if (EndState == 0) return 0;

else return 1;

}

void Dialog::EndExec()

{

cout << "Пока!";

EndState = 1;

}

void Dialog::GetEvent(TEvent& event)

{

string OptInt = "+-mszq";

string s;

string param;

char code;

cout << '>';

cin >> s; code = s[0];

if (OptInt.find(code) >= 0)

{

event.what = evMessage;

switch (code)

{

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

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

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

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

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

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

}

if (s.length() > 1)

{

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

int A = atoi(param.c\_str());

event.a = A;

}

}

else event.what = evNothing;

}

void Dialog::HandleEvent(TEvent& event)

{

if (event.what = evMessage)

{

switch (event.command)

{

case cmMake:

size = event.a;

beg = new Object \* [size];

cur = 0;

ClearEvent(event);

break;

case cmAdd:

Add();

ClearEvent(event);

break;

case cmDel:

Del();

ClearEvent(event);

break;

case cmShow:

Show();

ClearEvent(event);

break;

case cmQuit:

EndExec();

ClearEvent(event);

break;

case cmGet:

AvgAge();

ClearEvent(event);

break;

default:

HandleEvent(event);

}

}

}

int Dialog::Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (!Valid());

return EndState;

}

int main()

{

setlocale(LC\_ALL, "rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Добро пожаловать на сервер крутые сиськи!\n";

cout << "МЕНЮШЕЧКА: " << endl << endl;

cout << "-----------------------------------" << endl;

cout << endl << "1. Создать вектор: (m6, где 6-размер)\n2. Добавить элемент: +\n3. Удалить элемент: -\n4. Вывод полей элементов: s\n5. Средний возраст: z\n6. Выход: q\n\n";

cout << "-----------------------------------" << endl;

cout << endl;

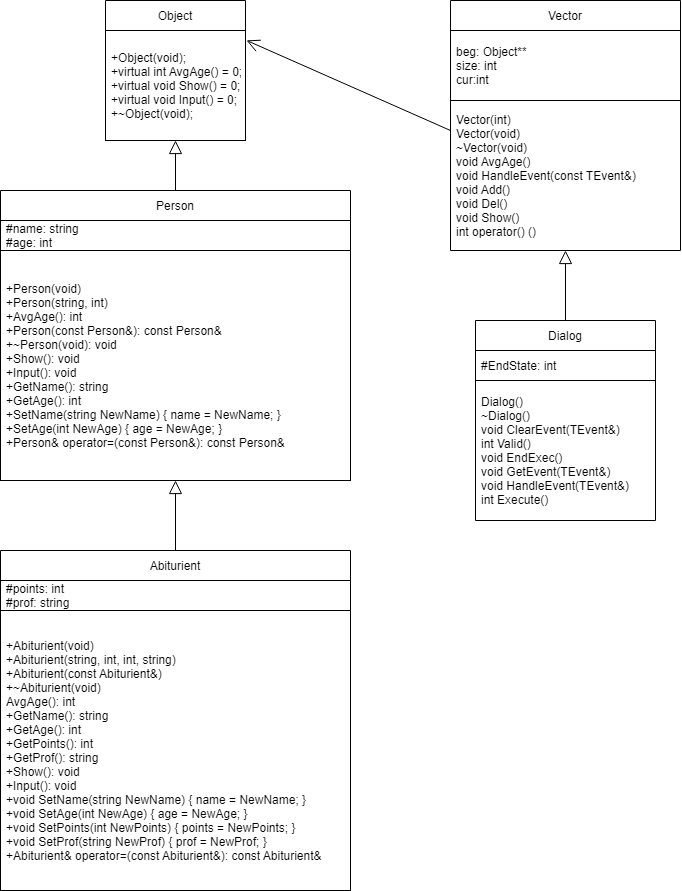
Dialog D;

D.Execute();

return 0;

}

**UML:**

****

**Контрольные вопросы:**

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

Группа - это объект, в который включены другие объекты. В отличие от контейнера мы понимаем группу как класс, который не только хранит объекты других классов, но и обладает собственными свойствами, не вытекающими из свойств его элементов. Группа дает второй вид иерархии - иерархию объектов(иерархию типа целое/часть), построенную на основе агрегации, первый вид - иерархия классов, построенная на основе наследования. **Пример:** Окно в интерактивной программе, которое владеет такими элементами, как поля ввода и редактирования данных, кнопки, списки выбора, диалоговые окна и т.д.

* + 1. Привести пример описания класса-группы Список (List).

template <class Type, class Allocator= allocator<Type>>

class list

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

List(void)

{

beg = 0;

size = 0;

cur = 0;

}

List(int n)

{

beg = new object \*[n];

size = n;

cur = 0;

}

List(const List& l)

{

beg = l.beg;

size = l.size;

cur = l.cur;

}

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

~List(void){}

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

void Info\_All()

{

if (cur == 0) cout << "Empty\n";

object\*\* p = beg;//указатель на указатель типа Object

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

{

(\*p)->show();//вызов метода Show() (позднее срабатывание)

++p;//передвигаем указатель на след объект

}

}

* + 1. Какой вид иерархии дает группа?

Иерархию объектов, построенную на агрегации.

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

Для комфортного наследования.

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

Событие – какое-либо действие(создание, модификация, удаление объекта) как реакция на ввод команды. События используются для системы диалогов между пользователем и программой.

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

Код команды и самое сообщение(содержание).

* + 1. Привести пример структуры, описывающей событие.

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

* + 1. Задана структура события

Struct TEvent

{

int what;

union

{

MouseEventType mouse;

KeyDownEvent keyDown;

MessageEvent message;

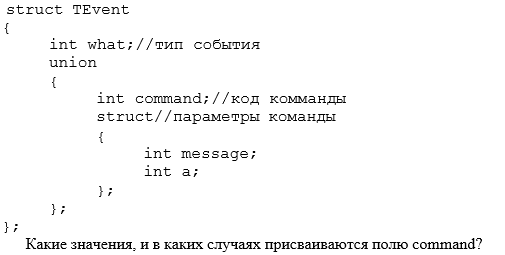
}

};

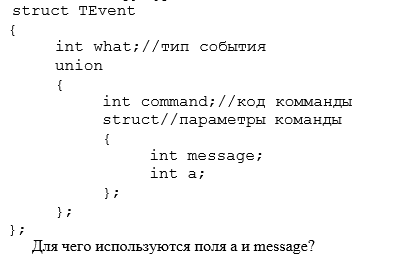
Какие значения, и в каких случаях присваиваются полю what?

0, 1, 2 в соотвествие с порядком полей.

* + 1. Задана структура события

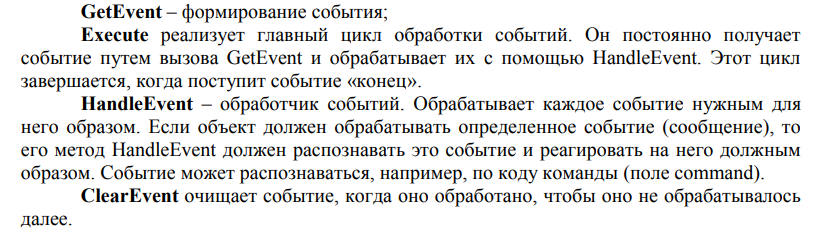


Значение берутся от обработки команды(параметр) в случае действия пользователя.

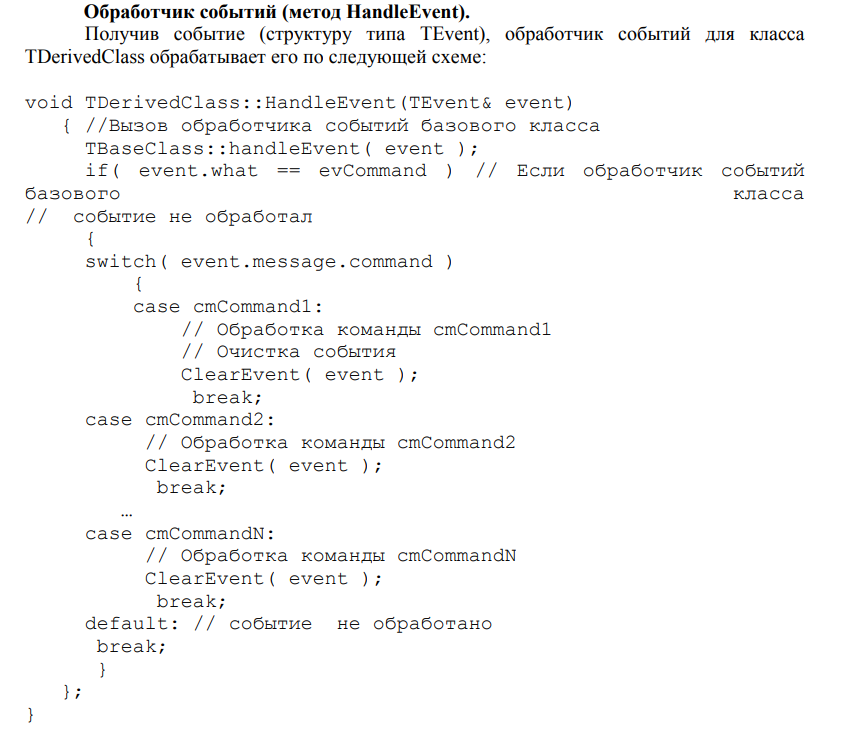


Код ссобщения, а – преобразованный параметр(команда) в число.

* + 1. Какие методы необходимы для организации обработки сообщений?



* + 1. Какой вид имеет главный цикл обработки событий-сообщений?



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

Очищает событие, чтобы оно не обрабатывалось далее. Отправляет в структуру пустое событие.

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

Обработчик событий, работает по коду команды.

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

Формирование события.

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

Конечное состояние программы. Содержится в классе Dialog.

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

Для проверки команды на ее «логичность» и подходит ли она.