Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

**Лабораторная работа «Класс №8»**

**по дисциплине**

**«Основы алгоритмизации и программирования»**

**(Семестр** 2)

Выполнил студент гр. ИВТ-21-1б

Гребнев Алексей Дмитриевич

Проверил:

Яруллин Денис Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

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 информацииоб атрибутах объектов.

2. Реализовать конструкторы, деструктор, операцию присваивания, селекторы имодификаторы.

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 размерегруппы.

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

**Исходные данные**

Вариант 10

Базовый класс:

ПЕЧАТНОЕ ИЗДАНИЕ(PRINT)

Название - string

Автор - string

Производный класс

ЖУРНАЛ (MAGAZIN)

Кол-во страниц - int

Группа - Список(List)

Команды:

-Создать группу(формат команды -> m кол-во элементов группы)

-Добавить элемент в группу(формат команды -> +)

-Удалить элемент из группы(формат команды -> -)

-Вывести информацию об элементах группы(формат команды -> s)

-Вывести информацию о названиях всех элементов группы (формат команды -> z)  
  
-Конец работы(формат команды -> q)

#include <iostream>

using namespace std;

const int evNothing = 0;//пустое событие

const int evMessage = 100;//непустое событие

const int cmAdd = 1;//добавить объект в группу

const int cmDel = 2;//удалить объект из группы

const int cmGet = 3;//вывести атрибут всех объектов

const int cmShow = 4;//вывести всю группу

const int cmMake = 6;//создать группу

const int cmQuit = 101;//выход

//класс событие

struct TEvent

{

int what;//тип события

union

{

int command;//код команды

struct

{

int message;

int a;//параметр команды

};

};

};

class Object

{

public:

Object();

virtual void Show() = 0;

virtual void Input() = 0;

virtual ~Object();

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

};

Object::Object() = default; // тоже самое что и {}

Object::~Object() = default; // тоже самое что и {}

class Print : public Object

{

protected:

string name;

string author;

public:

Print();//конструктор без параметров

virtual ~Print();//деструктор

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

void Input() override;//функция для ввода значений атрибутов

Print(string, string);//конструктор с параметрами

Print(const Print&);//конструктор копирования

//геттеры

string get\_name();

string get\_author();

//сеттеры

void set\_name(string);

void set\_author(string);

Print& operator=(const Print&);//перегрузка операции присваивания

virtual void HandleEvent(const TEvent& e);

};

//конструктор без параметров

Print::Print()

{

name = "";

author = "";

}

//деструктор

Print::~Print() {}

//конструктор с параметрами

Print::Print(string a, string b)

{

name = a;

author = b;

}

//конструктор копирования

Print::Print(const Print & a)

{

name = a.name;

author = a.author;

}

//селекторы

void Print::set\_name(string newName)

{

name = newName;

}

void Print::set\_author(string newAuthor)

{

author = newAuthor;

}

//геттеры

string Print::get\_name() { return name; }

string Print::get\_author() { return author; }

//оператор присваивания

Print& Print::operator=(const Print & a)

{

if (&a == this)

{

return \*this;

}

name = a.name;

author = a.author;

return \*this;

}

//метод для просмотра атрибутов

void Print::Show()

{

cout << "Название: " << name;

cout << "Автор: " << author;

cout << "\n";

}

//метод для ввода значений атрибутов

void Print::Input()

{

cout << "\nНазвание: "; cin >> name;

cout << "\nАвтор: "; cin >> author;

}

void Print::HandleEvent(const TEvent & e)

{

if (e.what == evMessage)//событие-сообщение

{

switch (e.command)

{

case cmGet:

cout << "name=" << get\_name() << endl;

break;

}

}

}

class Magazin :public Print

{

protected:

int sizebook;

public:

Magazin();

~Magazin() override;

void Show() override;

void Input() override;

Magazin(string, string, int);

Magazin(const Magazin&);

int get\_size();

void set\_size(int);

Magazin& operator=(const Magazin&);

};

//без параметров

Magazin::Magazin() : Print()

{

sizebook = 0;

}

//деструктор

Magazin::~Magazin() {}

//с параметрами

Magazin::Magazin(string a, string b, int c) : Print(a, b)

{

sizebook = c;

}

//копирование

Magazin::Magazin(const Magazin & a)

{

name = a.name;

author = a.author;

sizebook = a.sizebook;

}

//Сеттер

void Magazin::set\_size(int newSize)

{

sizebook = newSize;

}

//Геттер

int Magazin::get\_size()

{

return sizebook;

}

//оператор присваивания

Magazin& Magazin::operator=(const Magazin & a)

{

if (&a == this)

{

return \*this;

}

name = a.name;

author = a.author;

sizebook = a.sizebook;

return \*this;

}

//метод для просмотра атрибутов

void Magazin::Show()

{

cout << "Название: " << name;

cout << "Автор: " << author;

cout << "Размер по стр: " << sizebook;

cout << "\n";

}

//метод для ввода значений атрибутов

void Magazin::Input()

{

cout << "\nНазвание: "; cin >> name;

cout << "\nАвтор: "; cin >> author;

cout << "\nРазмер по стр: "; cin >> sizebook;

}

class List :public Magazin

{

protected:

Object\*\* beg;//указатель на первый элемент List

int size;//размер

int cur;//текущая позиция

public:

List();

List(int);

List(const List&);

~List();

void Add();

void Delete();

void Show();

int operator()();

virtual void HandleEvent(const TEvent& e);

};

//конструктор без параметров

List::List()

{

beg = 0;

size = 0;

cur = 0;

}

//конструктор копирования

List::List(const List & a)

{

beg = a.beg;

size = a.size;

cur = a.cur;

}

//деструктор

List::~List()

{

if (beg != 0)

{

delete[] beg;

}

beg = 0;

}

//конструктор с параметрами

List::List(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

//добавление объекта, на который указывает указатель p в вектор

void List::Add()

{

Object\* p;

//выбор из объектов двух возможных классов

cout << "1.Print" << endl;

cout << "2.Magazin" << endl << "Enter comand.num(y): " << endl;

int y;

cin >> y;

if (y == 1)//добавление объекта класса Print

{

Print\* a = new (Print);

a->Input();//ввод значений атрибутов

p = a;

if (cur < size)

{

beg[cur] = p;//добавление в вектор

cur++;

}

}

else

if (y == 2)//добавление объекта класса Magazin

{

Magazin\* b = new (Magazin);

b->Input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else { return; }

}

//просмотр List

void List::Show()

{

if (cur == 0)

{

cout << "Empty" << endl;

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

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

{

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

p++;//передвигаем указатель на следующий объект

}

}

}

//операция, которая возвращает размер List

int List::operator()()

{

return cur;

}

//удаление элемента из List, память не освобождается

void List::Delete()

{

if (cur == 0)

{

return;//пустенький

}

cur--;

}

void List::HandleEvent(const TEvent & e)

{

if (e.what == evMessage)

{

Object\*\* p = beg;

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

{

(\*p)->HandleEvent(e);//вызов метода(позднее связывание)

p++;//передвигаем указатель на следующий объект

}

}

}

class Dialog : public List

{

protected:

int EndState;

public:

Dialog();//конструктор

virtual ~Dialog();//деструктор

virtual void GetEvent(TEvent& event);//получить событие

virtual int Execute();//главный цикл обработки событий

virtual void HandleEvent(TEvent& event);//обработчик

virtual void ClearEvent(TEvent& event);//очистить событие

int Valid();//проверка атрибута EndState

void EndExec();//обработка события "конец работы"

};

//конструктор

Dialog::Dialog() :List()

{

EndState = 0;

}

//деструктор

Dialog::~Dialog() {}

//получение события

void Dialog::GetEvent(TEvent & event)

{

string OpInt = "+-szqam";//строка содержит коды операции

string s;

string param;

char code;

cout << '>';

cin >> s; code = s[0];//первый символ команды

if (OpInt.find(code) >= 0)//является ли символ кодом операции

{

event.what = evMessage;

switch (code)

{

case 'm'://создать группу

event.command = cmMake;

break;

case '+'://добавить объект в группу

event.command = cmAdd;

break;

case '-'://удалить объект из группы

event.command = cmDel;

break;

case 's'://вывести список

event.command = cmShow;

break;

case 'z'://вывести атрибуты 1 объекта

event.command = cmGet;

break;

case 'q'://выход

event.command = cmQuit;

break;

}

if (s.length() > 1) //выделяем параметры команды, ели они есть

{

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

}

}

//Главный цикл обработки событий

int Dialog::Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);//получить событие

HandleEvent(event);//обработать событие

} while (!Valid());

return EndState;

}

void Dialog::ClearEvent(TEvent & event)

{

event.what = evNothing;//пустое событие

}

int Dialog::Valid()

{

if (EndState == 0)

{

return 0;

}

else { return 1; }

}

void Dialog::EndExec()

{

EndState = 1;

}

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;

ClearEvent(event);//очищаем событие

break;

case cmAdd://добавление

Add();

ClearEvent(event);

break;

case cmDel:

Delete();//удаление

ClearEvent(event);

break;

case cmShow:

Show();//просмотр

ClearEvent(event);

break;

case cmQuit:

EndExec();//выход

ClearEvent(event);

break;

default:

HandleEvent(event);

};

};

}

int main()

{

setlocale(LC\_ALL, "RUS");

cout << "МЕНЮ: " << endl << endl;

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

cout << endl << "1. создать группу: m\n2. добавить объект в группу: +\n3. удалить объект из группы: -\n4. вывести список: s\n5. вывести атрибуты 1 объекта: z\n6. выход: q\n\n";

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

cout << endl;

cout << "В начале нужно создать вектор для взаимодействия" << endl << endl;

cout << "Для этого выберите символ 'm'(1 - работа с Print, 2 - работа с Magazin;" << endl << endl;

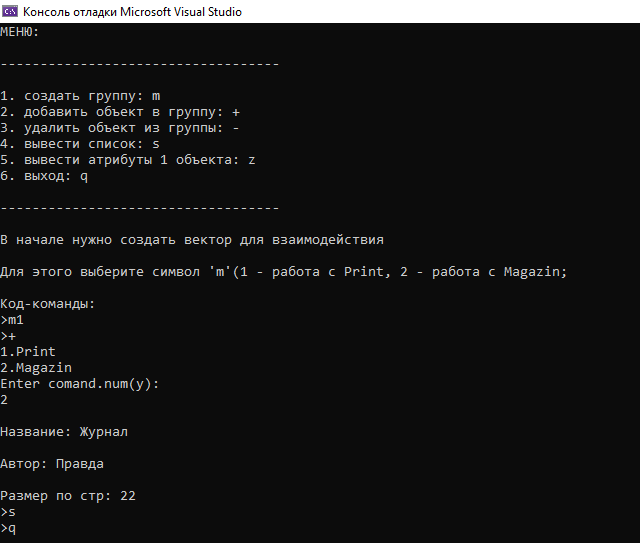
cout << "Код-команды:\n";

Dialog D;

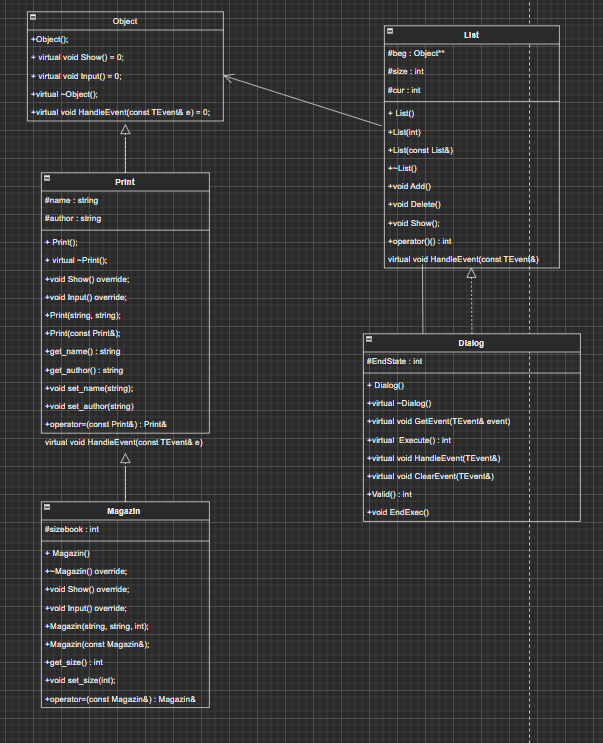
D.Execute();

return 0;

}

**Результат**  


**UML - диаграма**



**Контрольные вопросы:**

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

Понятия "группа" от "контейнер" отличаются. Контейнер используется для хранения других данных. Пример контейнеров: объекты контейнерных классов библиотеки STL в C++ (массивы, списки, очереди).

В отличие от контейнера группа есть класс, который не только хранит объекты других классов, но и обладает собственными свойствами, не вытекающими из свойств его элементов.

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

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

|  |
| --- |
| #include<list> |

Далее используем конструкцию ниже:

|  |  |
| --- | --- |
| 1 | list<типданных><имяконтейнера>; |

* < тип данных > — сюда мы должны указать тип, который хотим использовать.
* <имя контейнера> — это будет нашим именем контейнера. Лучше указывать такое имя, которое будет *говорить*, за что этот контейнер отвечает.

Вот пример создания списка с типом string:

|  |  |
| --- | --- |
| 1 | list<string>listok; |

* + 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 = newobject \*[n];

size = n;

cur = 0;

}

List(constList& 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 show\_elem()

{

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

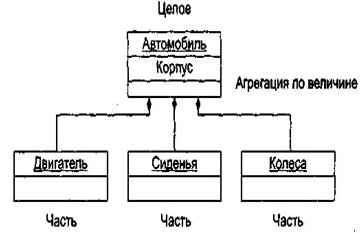
}

}

* + 1. **Какой вид иерархии дает группа?**

Иерархию объектов, построенную на агрегации.

Связи обозначают равноправные (клиент-серверные) отношения между объектами. Агрегация обозначает отношения объектов в иерархии «целое/часть». Агрегация обеспечивает возможность перемещения от целого (агрегата) к его частям (свойствам).

Агрегация может обозначать, а может и не обозначать физическое включение части в целое.  


* + 1. **Почему во главе иерархии классов, содержащихся в группе**

**объектов должен находиться абстрактный класс?**

Базовый класс используется для наследования интерфейса. Производные классы будут конкретизировать и реализовать этот интерфейс.  
  
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 и само сообщение (содержание)»

Объект инициирует событие и вызываются все объекты-обработчики. Т.е. от одного объекта к нескольким. Причем объект инициатор события может ничего не «знать» об его обработчиках, поэтому событие называют исходящим вызовом. Раз уж в С++ события на уровне языка не поддерживаются, значит стоит организовать их на уровне библиотеки.

* + 1. **Привести пример структуры, описывающей событие.**

structTEvent

{

int what;

union

{

int command;

struct

{

int message;

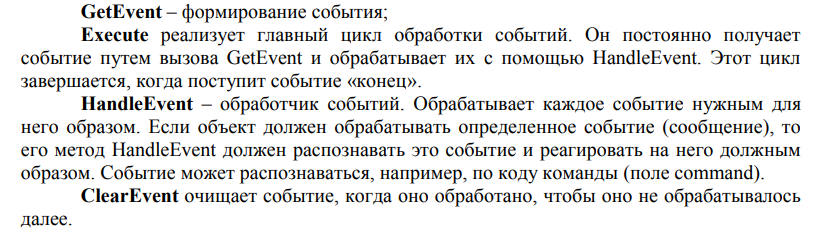
int a;

};

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

Функция обработки, сообщения опознает, какое именно сообщение поступило для обработки и выполняет соответствующие действия.  
Сообщения распознаются по его коду. Обычно функция содержит оператор switch, который служит для определения кода сообщений.

* + 1. **Какой видим ее главный цикл обработки событий-сообщений?**

Execute реализует главный цикл обработки событий. Он постоянно получает событие путем вызова GeEvent и обрабатывает их с помощью HandleEvent. Этот цикл завершается, когда поступит событие «конец». HandleEvent – обработчик событий. Обрабатывает каждое событие нужным для него образом.   


* + 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. Обрабатывает каждое событие нужным для него образом. Если объект должен обрабатывать определенное событие (сообщение), то его метод HandleEvent должен распознавать это событие и реагировать на него должным образом. Событие может распознаваться, например, по коду команды (поле command).

* + 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