Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский

политехнический университет»

Лабораторная работа №8

«Программа, управляемая событиями»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Программа, управляемая событиями

**Цель задания**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Разработка программы, управляемой событиями.

**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.

* 1. 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.
  2. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.
  3. Добавить методы для обработки событий группой и объектами пользовательских классов.
  4. Написать тестирующую программу.
  5. Нарисовать диаграмму классов и диаграмму объектов

**Задание**

Базовый класс:

ПЕЧАТНОЕ\_ИЗДАНИЕ(PRINT)

Название– string

Автор – string

Производный класс

ЖУРНАЛ (MAGAZIN)

Количество страниц - int 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 о названиях всех элементов группы

(формат команды : z) Конец работы (формат команды: q)

**Анализ задачи**

* 1. Необходимо определить иерархию пользовательских классов, где во главе стоит абстрактный класс с чисто виртуальными методами для ввода
* вывода информации об атрибутах объектов.
  1. Для класса необходимо реализовать конструкторы, деструктор,

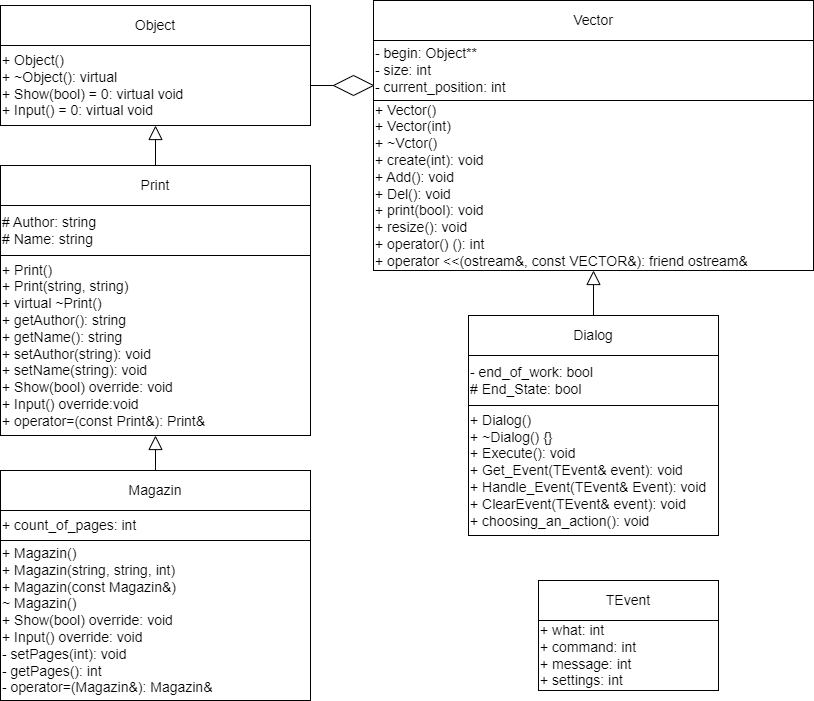
операцию присваивания, селекторы и модификаторы.

1. Определить класс-группу на основе структуры, Print
2. 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.

1. Определить класс Диалог – наследника группы, в котором реализовать методы для обработки событий.
2. Добавить методы для обработки событий группой и объектами пользовательских классов.

**UML диаграмма**

****

**Код**

Файл ООП8.cpp

#include <iostream>

using namespace std;

#include "Dialog.h"

int main()

{

system("chcp 1251>null");

Dialog dialog;

dialog.Execute();

return 0;

}

Файл Object.h

#pragma once

class Object

{

public:

Object() {}

virtual ~Object() {}

virtual void Show(bool) = 0;

virtual void Input() = 0;

};

Файл Print.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

#include "Object.h"

class Print : public Object

{

protected:

string Author;

string Name;

public:

Print();

Print(string, string);

Print(const Print&);

virtual ~Print() {}

void Show(bool) override;

void Input() override;

Print& operator=(const Print&);

void setAuthor(string& author) { this->Author = author; }

void setName(string& name) { this->Name = name; }

string getAuthor() { return this->Author; }

string getName() { return this->Name; }

};

Файл Print.cpp

#pragma once

#include "Print.h"

Print::Print()

{

this->Author = "noAuthor";

this->Name = "NoName";

}

Print::Print(string author, string name)

{

this->Author = author;

this->Name = name;

}

Print::Print(const Print& other)

{

this->Author = other.Author;

this->Name = other.Name;

}

void Print::Show(bool names = false)

{

if (!names)

{

cout << "Автор: " << this->Author

<< "\nНазвание: " << this->Name << "\n\n";

}

else

{

cout << "Название: " << this->Name << "\n\n";

}

}

void Print::Input()

{

cout << "\nВведите данные\n";

cout << "Автор: ";

cin.seekg(cin.eof());

getline(cin, this->Author);

cout << "Название: ";

cin.seekg(cin.eof());

getline(cin, this->Name);

}

Print& Print::operator=(const Print& other\_Print)

{

if (&other\_Print == this)

{

return \*this;

}

this->Author = other\_Print.Author;

this->Name = other\_Print.Name;

return \*this;

}

Файл Magazin.h

#pragma once

#include "Print.h"

class Magazin : public Print

{

protected:

int count\_of\_page;

public:

Magazin();

Magazin(string, string, int);

Magazin(const Magazin&);

~Magazin() {}

void Show(bool) override;

void Input() override;

Magazin& operator = (const Magazin&);

void setPages(int& pages) { this->count\_of\_page = pages; }

int getPages() { return this->count\_of\_page; }

};

Файл Magazin.cpp

#pragma once

#include "Magazin.h"

Magazin::Magazin() : Print()

{

this->count\_of\_page = 0;

}

Magazin::Magazin(string author, string name, int pages) : Print(author, name)

{

this->count\_of\_page = pages;

}

Magazin::Magazin(const Magazin& other\_magazin)

{

this->Author = other\_magazin.Author;

this->Name = other\_magazin.Name;

this->count\_of\_page = other\_magazin.count\_of\_page;

}

void Magazin::Show(bool names = false)

{

if (!names)

{

cout << "Автор: " << this->Author

<< "\nНазвание: " << this->Name

<< "\nКоличество страниц: " << this->count\_of\_page << "\n\n";

}

else

{

cout << "Название: " << this->Name << "\n\n";

}

}

void Magazin::Input()

{

cout << "\nВведите данные\n";

cout << "Автор: ";

cin.seekg(cin.eof());

getline(cin, this->Author);

cout << "Название: ";

cin.seekg(cin.eof());

getline(cin, this->Name);

cout << "Количество страниц: "; cin >> this->count\_of\_page;

}

Magazin& Magazin::operator=(const Magazin& other\_magazin)

{

if (this == &other\_magazin)

{

return \*this;

}

Print::operator=(other\_magazin);

this->count\_of\_page = other\_magazin.count\_of\_page;

return \*this;

}

Файл Vector.h

#pragma once

#include <iostream>

using namespace std;

#include "Object.h"

#include "Print.h"

#include "Magazin.h"

class Vector

{

protected:

Object\*\* begin;

int size;

int current\_position;

public:

Vector();

Vector(int);

~Vector();

void Add();

void Del();

void print(bool);

void create(int);

void resize(int);

int operator() ();

friend ostream& operator << (ostream&, const Vector&);

};

Файл Vector.cpp

#pragma once

#include "Vector.h"

Vector::Vector()

{

this->begin = 0;

this->size = 0;

this->current\_position = 0;

}

Vector::Vector(int size)

{

this->begin = new Object \* [size];

this->size = size;

this->current\_position = 0;

}

Vector::~Vector()

{

if (begin != 0)

{

delete[] begin;

}

begin = 0;

}

void Vector::Add()

{

if (this->current\_position < this->size)

{

Object\* ptr;

cout << "Объект какого класса?\n1) Print\n2) Magazin\n";

int choice;

do

{

cout << "Ваш выбор: ";

cin >> choice;

} while (choice < 1 && choice > 2);

if (choice == 1)

{

Print\* print = new Print;

print->Input();

cout << "\n";

ptr = print;

this->begin[this->current\_position] = ptr;

this->current\_position++;

}

else if (choice == 2)

{

Magazin\* magazin = new Magazin;

magazin->Input();

cout << "\n";

ptr = magazin;

this->begin[this->current\_position] = ptr;

this->current\_position++;

}

}

else

{

cout << "Больше некуда =(\n\n";

}

}

void Vector::Del()

{

if (current\_position > 0)

{

this->current\_position--;

}

else

{

cout << "Меньше некуда =(\n\n";

}

}

void Vector::print(bool names = false)

{

if (this->current\_position == 0)

{

cout << "Пусто =(\n\n";

}

else

{

Object\*\* ptr = this->begin;

for (int i = 0; i < this->current\_position; i++)

{

(\*ptr)->Show(names);

ptr++;

}

}

}

void Vector::create(int size)

{

if (this->begin != 0)

{

for (int i = 0; i < this->size; i++)

{

delete this->begin[i];

}

delete[] this->begin;

}

this->begin = new Object \* [size];

this->size = size;

this->current\_position = 0;

}

void Vector::resize(int new\_size)

{

int temp\_size = this->size;

this->size = new\_size;

if (this->current\_position != 0)

{

if (new\_size < temp\_size)

{

this->current\_position--;

for (int i = temp\_size; i > new\_size; i--)

{

if(this->current\_position > 0)

{

delete begin[this->current\_position];

this->current\_position--;

}

}

if(this->size != 0)

{

this->current\_position++;

}

}

}

}

int Vector::operator() ()

{

return this->size;

}

ostream& operator<<(ostream& stream, const Vector& vector)

{

if (vector.current\_position == 0)

{

cout << "Пусто =(\n\n";

}

else

{

Object\*\* ptr = vector.begin;

for (int i = 0; i < vector.current\_position; i++)

{

(\*ptr)->Show(false);

ptr++;

}

}

return stream;

}

Файл Event.h

#pragma once

const int EvNothing = 0;

const int EvMessage = 100;

const int CmCreateGroup = 1; //m создать группу

const int CmAddToGroup = 2; //+ добавить в группу элемент

const int CmRemoveFromGroup = 3; //- удалить элемент из группы

const int CmShow = 4; //s Вывести информацию об элементах группы

const int CmShowName = 5; //z Вывести информацию о названиях всех элементов группы

const int CmResize = 6; //r Изменить размер группы

const int CmExit = 7; //q Конец работы

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int settings;

};

};

};

Файл Dialog.h

#pragma once

#include "Vector.h"

#include "Event.h"

#include "Print.h"

#include "Magazin.h"

class Dialog : public Vector

{

protected:

int EndState;

public:

Dialog();

~Dialog() {}

void GetEvent(TEvent&);

bool is\_command(char);

int Execute();

void HandleEvent(TEvent&);

void ClearEvent(TEvent&);

int Valid();

void EndExec();

};

Файл Dialog.cpp

#pragma once

#include "Dialog.h"

Dialog::Dialog() : Vector()

{

this->EndState = 0;

}

void Dialog::GetEvent(TEvent&event)

{

cout << "Выберите действие\n"

<<"m: Создать группу (Пример команды: m10)\n"

<< "+: Добавить элемент в группу\n"

<< "-: Удалить элемент из группы\n"

<< "s: Вывести информацию об элементах группы\n"

<< "z: Вывести информацию о названиях всех элементов группы\n"

<< "r: Изменить размер группы (Пример команды: r7)\n"

<< "q: Конец работы\n";

cout << "> ";

string str, set; cin >> str;

char code = str[0];

if (is\_command(code))

{

event.what = EvMessage;

switch (code)

{

case 'm':

event.command = CmCreateGroup;

break;

case '+':

event.command = CmAddToGroup;

break;

case '-':

event.command = CmRemoveFromGroup;

break;

case 's':

event.command = CmShow;

break;

case 'z':

event.command = CmShowName;

break;

case 'r':

event.command = CmResize;

break;

case 'q':

event.command = CmExit;

break;

}

if (str.size() > 1)

{

set = str.substr(1, str.size() - 1);

int settings = atoi(set.c\_str());

event.settings = settings;

}

}

else

{

event.what = EvNothing;

}

}

bool Dialog::is\_command(char action)

{

int i = 0;

string str = "m+-szrq";

while (i < str.size())

{

if (action == str[i])

{

return true;

}

i++;

}

return false;

}

int Dialog::Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);

cout << "\n";

HandleEvent(event);

} while (!Valid());

return EndState;

}

void Dialog::HandleEvent(TEvent& event)

{

if (event.what == EvNothing)

{

cout << "Такой команды нет\n\n";

}

else

{

switch (event.command)

{

case CmCreateGroup:

create(event.settings);

ClearEvent(event);

break;

case CmAddToGroup:

Add();

ClearEvent(event);

break;

case CmRemoveFromGroup:

Del();

ClearEvent(event);

break;

case CmShow:

print(false);

ClearEvent(event);

break;

case CmShowName:

print(true);

ClearEvent(event);

break;

case CmResize:

resize(event.settings);

ClearEvent(event);

break;

case CmExit:

EndExec();

ClearEvent(event);

break;

default:

break;

}

}

}

void Dialog::ClearEvent(TEvent& event)

{

event.what = EvNothing;

}

int Dialog::Valid()

{

if (EndState == 0)

{

return 0;

}

return 1;

}

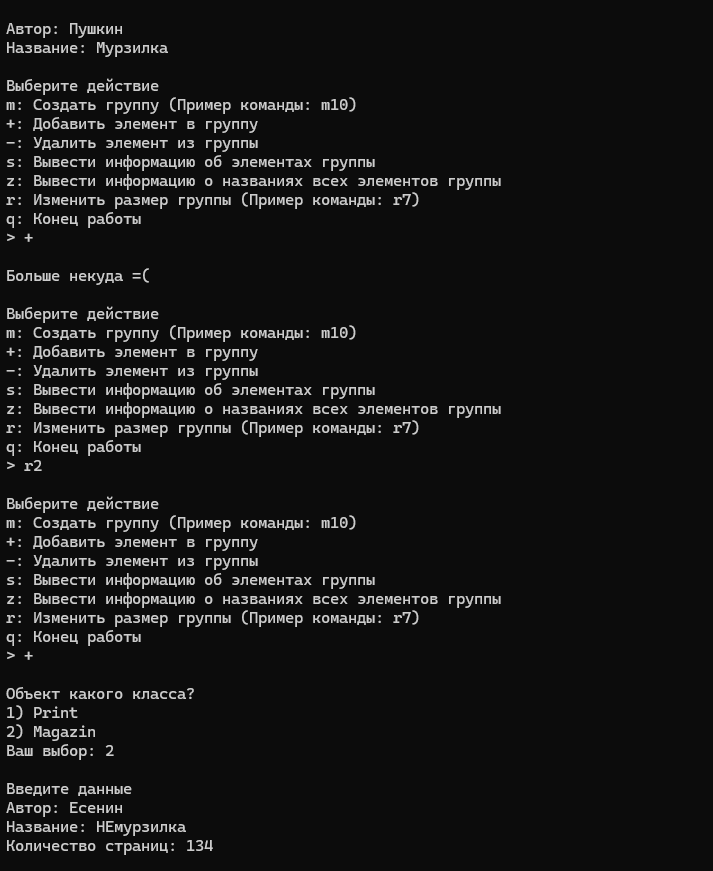
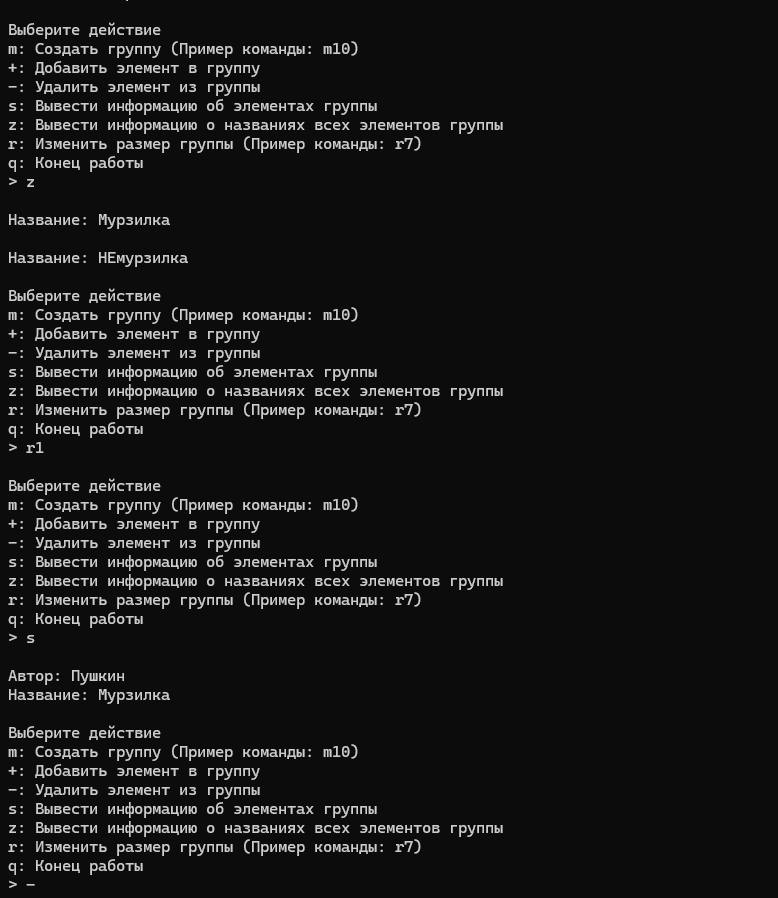
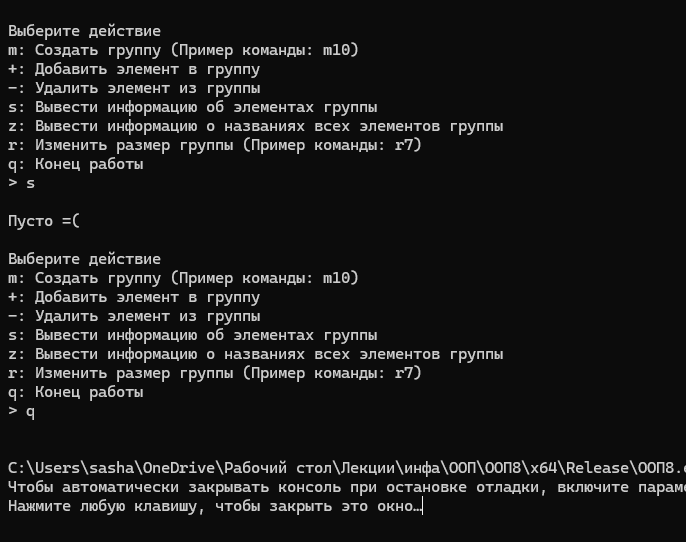
void Dialog::EndExec()

{

EndState = 1;

}

**Результаты работы**

**Github**

<https://github.com/Korovay4ik/Laboratory-works>

**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. Например, можно создать класс-группу "Список студентов", который будет содержать объекты класса "Студент". Класс-группа будет предоставлять методы для добавления/удаления студентов, получения списка студентов и т.д.

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

struct Node {

int data; //данные

Node\* ptr\_to\_prev\_node = nullptr; //указатель на предыдущий элемент Node\* ptr\_to\_next\_node = nullptr; //указатель на следующий элемент

};

class List {

private:

int size; //размер

Node\* head; //указатель на голову

Node\* tail; //указатель на хвост

Iterator begin;//указатель на начальный элемент

Iterator end;//указатель на последний элемент

public:

List(int size); //конструктор с длиной

List(int size, int data);//конструктор с длиной и данными

List(const List& list);//конструктор копирования

~List();//деструктор

void push\_bask(int data);//вставка элемента в конец

void push\_front(int data);//вставка элемента в НАЧАЛО

Iterator first\_el();//последний элемент

Iterator last\_el();//первый элемент

int pop\_bask();//удалить последний элемент

int pop\_front();//удалить первый элемент

bool is\_empty();//список пустой иль нет

int operator () ();//размер списка

int& operator[](int index);//данные по индексу

List& operator = (const List& list);//оператор присваивания

List operator \*(List& list);

friend ostream& operator << (ostream& stream, const List& list);//ВЫВОД friend istream& operator >> (istream& stream, const List& list);//ВВОД

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

List::List(int size) {//конструктор с длиной

this->size = size;

if (size > 0) {//если длина больше 0

Node\* node = new Node;//создаю узел

this->head = node; //указатель на голову

this->tail = node; //указатель на хвост

for (int i = 1; i < size; i++) {//заполняю со второго номера

Node\* New\_Node = new Node;//новый узел

tail->ptr\_to\_next\_node = New\_Node;//хвост указывает на новый узел

New\_Node->ptr\_to\_prev\_node = tail;//новый узел ук-т на хвост как на пред-й tail = New\_Node;//новый узел становится хвостом}

tail->ptr\_to\_next\_node = nullptr;//до хвоста элементов нет

}

else {

this->head = nullptr; //указатель на голову

this->tail = nullptr; //указатель на хвост

}

this->begin.elem = this->head;

this->end.elem = this->tail;

}

List::List(int size, int data) {//конструктор с длиной и данными this->size = size;

if (size > 0) {//если длина больше 0

Node\* node = new Node;//создаю узел

node->data = data; //данные для узла

this->head = node; //указатель на голову

this->tail = node; //указатель на хвост

for (int i = 1; i < size; i++) {//заполняю со второго номера

Node\* New\_Node = new Node;//новый узел

New\_Node->data = data; //данные для нового узла

tail->ptr\_to\_next\_node = New\_Node;//хвост указывает на новый узел

New\_Node->ptr\_to\_prev\_node = tail;//новый узел ук-т на хвост как на пред-й tail = New\_Node;//новый узел становится хвостом

}

tail->ptr\_to\_next\_node = nullptr;//до хвоста элементов нет

}

else {

this->head = nullptr; //указатель на голову

this->tail = nullptr; //указатель на хвост

}

this->begin.elem = this->head;

this->end.elem = this->tail;

}

List::List(const List& list) {//конструктор копирования

this->head = nullptr; //указатель на голову

this->tail = nullptr; //указатель на хвост

this->size = 0;

Node\* current\_node = list.head;//создаю узел

while (current\_node != nullptr) {//пока не пройду весь список push\_bask(current\_node->data);//добавляю в конец эл-т

current\_node = current\_node->ptr\_to\_next\_node;//беру следующий элемент

}

this->begin.elem = this->head;

this->end.elem = this->tail;

}

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

List::~List() {//деструктор

Node\* current\_node = head;//голова

while (current\_node != nullptr) {//пока не пройду весь список

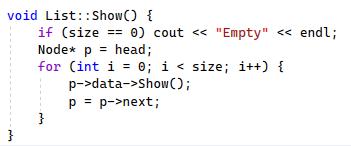
Node\* next\_node = current\_node->ptr\_to\_next\_node;//следующий элемент delete current\_node;//удаляю текущий элемент

current\_node = next\_node;//беру новый элемент

}

head = nullptr; //голова пустая

1. } **Привести пример метода для просмотра элементов для класса-группы Список.**
2. **Какой вид иерархии дает группа?**



Группа дает второй вид иерархии - 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 должен находиться во главе иерархии классов, содержащихся в группе объектов, чтобы обеспечить единообразие интерфейса

* полиморфизм. Это позволяет объектам производных классов использоваться взаимозаменяемо, что упрощает разработку и поддержку кода.

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

Событие — это сигнал о том, что произошло некоторое действие или

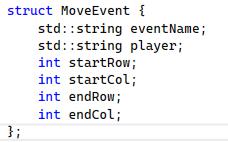
изменение состояния. События используются для реализации интерактивности в приложениях, позволяя компонентам реагировать на действия пользователей или другие внешние факторы.

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

Объект TEvent состоит из двух частей. Первая (what) задает тип события, опреде-ляющий источник данного события. Вторая задает информацию, передаваемую с событием. Для разных типов событий содержание информации различно. Поле what может принимать следующие значения:

* evNothing это пустое событие, которое означает, что ничего делать не надо. По-лю what присваивается значение evNothing, когда событие обработано каким-либо объектом.
* evMessage - событие-сообщение от объекта.

1. **Привести пример структуры, описывающей событие.**
2. **Задана структура события**



**struct TEvent {**

**int what;**

**union {**

**MouseEventType mouse;**

**KeyDownEvent keyDown;**

**MessageEvent message;**

**}**

**};**

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

В структуре события TEvent, поле what определяет тип события. В зависимости от значения what, соответствующая часть union будет использована для хранения параметров события. Например, если what равно MouseEventType, то параметры будут храниться в mouse.

1. **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?**

В структуре события TEvent, поле command является кодом команды. В зависимости от значения command, соответствующие параметры команды будут использованы. Поле a и message являются параметрами команды.

1. **Задана структура события**

**struct TEvent {**

**int what;//тип события**

**union {**

**int command;//код комманды**

**struct//параметры команды**

**{**

**int message;**

**int a;**

**};**

**};**

**};**

**Для чего используются поля a и message?**

Полю command присваиваются коды различных определённых команд. При получении того или иного сообщения, поле command принимает одно из кодов команд

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

* GetEvent – формирование события;
* Execute реализует главный цикл обработки событий. Он постоянно по-лучает событие путем вызова GetEvent и обрабатывает их с помощью HandleEvent. Этот цикл завершается, когда поступит событие

«конец».

* HandleEvent – обработчик событий. Обрабатывает каждое событие нуж-ным 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 рас-

познаваться, например, по коду команды (поле command).

* ClearEvent очищает событие, когда оно обработано, чтобы оно не обра-батывалось далее.
* Valid - проверяет, завершена ли работа.
* EndExec - завершение обработки событий (после вызова этого метода цикл обработки событий заканчивается).

1. **Какой вид имеет главный цикл обработки событий-**

**сообщений?**

void Dialog:: Execute() {

TEvent Event;

while (!end\_of\_work) {

Get\_Event(Event);

Handle\_Event(Event);

}

}

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

**образом?**

Метод ClearEvent() очищает буфер событий, удаляя все сообщения.

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

**образом?**

void Dialog::Handle\_Event(TEvent& Event) {

if (Event.Command == EV\_Nothing) {

cout << "Такой команды нет!" << endl;

}

else {

int size;

string NName;

switch (Event.Command) {//выбор действия

case CM\_create\_a\_group:

do {

cout << "Введите размер: ";

cin >> size;

} while (size <= 0); RE\_create(size); break;

case CM\_Add\_to\_a\_group:

Add();

break;

case CM\_Remove\_from\_the\_group:

Del();

break;

case CM\_Print:

print();

cout << endl;

break;

case CM\_inform\_all\_el\_of\_the\_group:

print(true);

cout << endl;

break;

case CM\_Quit:

end\_of\_work = true;

break;

default:

break;

}

ClearEvent(Event);

}

}

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

Метод GetEvent() используется для получения информации о следующем со-бытии в очереди событий. Если в очереди нет ни одного события, метод бло-кируется до появления нового события. Если в очереди есть события, метод возвращает информацию о первом событии в очереди и удаляет его из очере-ди. Обычно этот метод вызывается в главном цикле обработки событий.

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

**содержит это поле?**

Поле EndState используется для указания конечного состояния системы после выполнения события. Обычно это поле заполняется в обработчике события, который анализирует информацию, полученную из события, и принимает решение о том, как изменить состояние системы.

Поле EndState используется в классе Dialog и отвечает за состояние работы диалога (идёт/завершён).

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

Функция Valid() используется для проверки, является ли текущее состояние объекта допустимым. Она возвращает булево значение true, если текущее со-стояние объекта допустимо, и false в противном случае.