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
Акбашева Софья Руслановна

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.

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

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 и вывода информации об атрибутах объектов.

2) Для класса необходимо реализовать конструкторы, деструктор, операцию присваивания, селекторы и модификаторы.

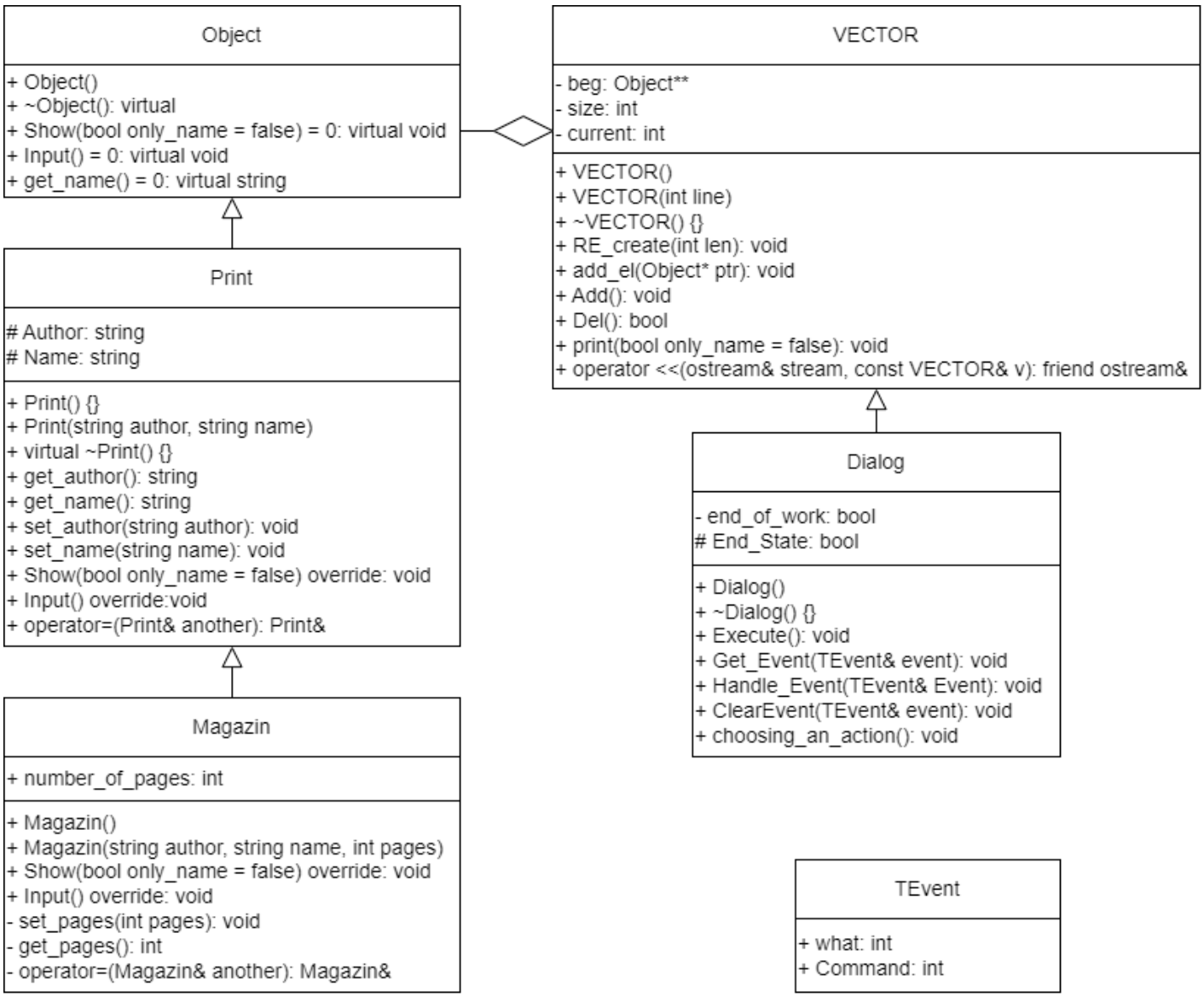
3) Определить класс-группу на основе структуры, Print

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.

UML диаграмма



Код

Файл Object.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>

using namespace std;

class Object {
public:
	Object() {}; //конструктор
	virtual ~Object() {};//деструктор
	virtual void Show(bool only_name = false) = 0;
	virtual void Input() = 0;
	virtual string get_name() = 0;
};
```

Файл Print.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>
```

```
#include "Object.h"
using namespace std;

class Print : public Object {
protected:
	string Author;
	string Name;
public:
	Print() {};//конструктор без параметров
	Print(string author, string name);//конструктор с параметрами
	virtual ~Print() {};//деструктор

	string get_author();//получение имени автора
	string get_name();//получение названия

	void set_author(string author);//установка автора
	void set_name(string name);//установка названия
	void Show(bool only_name = false) override;//вывод данных
	void Input() override;
	Print& operator=(Print& another);
};
```

Файл Print.cpp

```
#include "Print.h"
using namespace std;

Print::Print(string author, string name) {//конструктор с параметрами
		set_author(author);
		set_name(name);
}

string Print::get_author() {//получение имени автора
	return Author;
}
string Print::get_name() {//получение названия
	return Name;
}

void Print::set_author(string author) {//установка автора
	Author = author;
}
void Print::set_name(string name) {//установка названия
	Name = name;
}

void Print::Show(bool only_name){//вывод данных
	if (only_name == false) {
		cout << "Автор: " << this->get_author() << ". ";
	}
	cout << "Название: " << this->get_name();
}
void Print::Input(){
	cin.ignore();
	cout << "Введите имя автора: ";
	getline(cin, Author);
	cout << "Введите название: ";
	getline(cin, Name);
}

Print& Print::operator=(Print& another) {
	Author = another.Author;
```

```
	Name = another.Name;
	return *this;
}
```

Файл Magazin.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>
#include "Print.h"
using namespace std;

class Magazin : public Print {
public:
	int number_of_pages;
	Magazin() : Print() {}//конструктор без параметров
	Magazin(string author, string name, int pages) : Print(author, name) {
		set_pages(pages);
	}

	void Show(bool only_name = false) override;
	void Input() override;
private:
	void set_pages(int pages);
	int get_pages();
	Magazin& operator=(Magazin& another);
};
```

Файл Magazin.cpp

```
#include "Magazin.h"
using namespace std;

void Magazin::Show(bool only_name){
	Print::Show(only_name);
	if (only_name == false) {
		cout << ". Количество страниц: " << number_of_pages;
	}
}

void Magazin::Input(){
	Print::Input();
	cout << "Введите количество страниц: ";
	cin >> number_of_pages;
}

void Magazin::set_pages(int pages) {
	number_of_pages = pages;
}

int Magazin::get_pages() {
	return number_of_pages;
}

Magazin& Magazin::operator=(Magazin& another) {
	Print::operator=(another);
	number_of_pages = another.number_of_pages;
	return *this;
}
```

Файл VECTOR.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include "Object.h"
#include "Print.h"
#include "Magazin.h"
using namespace std;

class VECTOR {
	Object** beg;
	int size;//размер вектора
	int current;
public:
	VECTOR();//конструктор по умолчанию
	VECTOR(int line);//конструктор с длиной
	~VECTOR() {};//деструктор

	void RE_create(int len);//пересоздаю вектор
	void add_el(Object* ptr);//добавление элемента в вектор
	void Add();//добавление элемента
	bool Del();//удаление элемента
	void print(bool only_name = false);//вывод
	friend ostream& operator <<(ostream& stream, const VECTOR& v);//оператор для потока вывода
};
```

Файл VECTOR.cpp

```
#include "VECTOR.h"
using namespace std;

VECTOR::VECTOR() {//конструктор по умолчанию
	beg = 0;
	size = 0;//всего элементов
	current = 0;//элементов сейчас
};

VECTOR::VECTOR(int line) {//конструктор с длиной
	beg = new Object * [line];
	current = 0;
	size = line;
}

void VECTOR::RE_create(int len) {//пересоздаю вектор
	if (beg != 0) {
		for (int i = 0; i < current; i++) {//удаляю текущий вектор
			delete beg[i];
		}
		delete[] beg;
	}
	size = len;
	beg = new Object * [len];
	current = 0;
}

void VECTOR::add_el(Object* ptr) {//добавление элемента в вектор
	if (current < size) {//если вектор не заполнен
		beg[current] = ptr;
		current++;//увеличиваю текущее количество элементов
	}
};

void VECTOR::Add() {
	if (current == size) {
```

```
                cout << "Места нет!" << endl;
                if (beg == NULL) {
                        cout << "Вектор не создан!" << endl;
                }
        }
        else {
                Object* p;
                int choice;
                do {
                        cout << "Какой класс создать?" << endl;
                        cout << "1. Print" << endl;
                        cout << "2. Magazine" << endl;
                        cin >> choice;
                } while (choice < 1 || choice > 2);
                cout << endl;
                if (choice == 1) {
                        Print* a = new Print;
                        a->Input();
                        p = a;
                        add_el(p);
                }
                else {
                        Magazin* b = new Magazin;
                        b->Input();
                        p = b;
                        add_el(p);
                }
        }
}

bool VECTOR::Del() {
        bool flag = false;
        if (beg == NULL) {
                cout << "Вектор не создан!" << endl;
        }
        else {
                if (current == 0) {
                        cout << "Удалять нечего, вектор пустой" << endl;
                }
                else {
                        string NName;
                        cout << "Введите название, по которому надо удалить ";
                        cin.ignore();
                        getline(cin, NName);
                        for (int i = 0; i < current && !flag; i++) {
                                if (beg[i]->get_name() == NName) {
                                        flag = true;
                                        Object* tmp = beg[i];
                                        for (int j = i; j < current - 1; j++) {//перемещаю объекты за место
удаленого
                                                beg[i] = beg[j + 1];
                                        }
                                        delete tmp;
                                        current--;//уменьшаю длину
                                }
                        }
                        if (flag) {
                                cout << "Удаление успешно завершено" << endl;
                        }
                        else {
                                cout << "Объект для удаления не найден" << endl;
                        }
```

```
                }
        }
        return flag;
}

void VECTOR::print(bool only_name) {
        if (current == 0) {
                cout << "Вектор пуст(";
        }
        else {
                for (int i = 0; i < current; i++) {
                        beg[i]->Show(only_name);
                        cout << endl;
                }
        }
}

ostream& operator <<(ostream& stream, const VECTOR& v) {//оператор для потока вывода
        if (v.size == 0) {
                cout << endl << "Вектор пуст(";
        }
        else {
                Object** ptr = v.beg;
                for (int i = 0; i < v.current; i++) {
                        (*ptr)->Show();
                        cout << endl << endl;
                        ptr++;
                }
        }
        return stream;
}
```

Файл Dialog.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>
#include "VECTOR.h"
#include "TEvent.h"
#include "Magazin.h"
using namespace std;

class Dialog : public VECTOR {
private:
        bool end_of_work = false;//надо ли закончить работу
public:
        Dialog();//конструктор без параметров
        ~Dialog() {};//деструктор
        void Execute();
        void Get_Event(TEvent& event);//ставлю событие
        void Handle_Event(TEvent& Event);//выбираю событие
        void ClearEvent(TEvent& event); //очищаю текущую команду
        void choosing_an_action();
protected:
        bool End_State;
};
```

Файл Dialog.cpp

```
#include "Dialog.h"
using namespace std;
```

```
Dialog::Dialog() {
	end_of_work = false;
}

void Dialog:: Execute() {
	TEvent Event;
	while (!end_of_work) {
		Get_Event(Event);
		Handle_Event(Event);
	}
}

void Dialog::Get_Event(TEvent& event) {
	choosing_an_action();

	string str;
	cout << ">  ";
	cin >> str;
	char CM_symbol = str[0];//беру первый символ
	cout << "Ваша команда: " << CM_symbol << endl << endl;

	event.what = EV_Message;
	switch (CM_symbol) {
	case 'm'://m создать группу
		event.Command = CM_create_a_group;
		break;
	case '+'://+ добавить в группу элемента
		event.Command = CM_Add_to_a_group;
		break;
	case '-'://- удалить элемент из группы
		event.Command = CM_Remove_from_the_group;
		break;
	case 's'://s		Вывести информацию об элементах груп
		event.Command = CM_Print;
		break;
	case 'z'://z Вывести информацию о названиях всех элементов группы
		event.Command = CM_inform_all_el_of_the_group;
		break;
	case 'q'://q Конец работы
		event.Command = CM_Quit;
		break;
	default://левое значение
		event.Command = EV_Nothing;
	}
}

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);
        }
}

void Dialog::ClearEvent(TEvent& event) {
        event.what = EV_Nothing;
}

void Dialog::choosing_an_action() {
        cout << "------------------------------------------------------------" << endl;
        cout << "Что вы хотите сделать?" << endl;
        cout << "m:   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:   Вывести информацию о названиях всех элементов группы" << endl;
        cout << "q:   Конец работы" << endl;
}
```

Файл TEvent.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен

using namespace std;

const int CM_create_a_group = 1; //m создать группу
const int CM_Add_to_a_group = 2; //+ добавить в группу элемента
const int CM_Remove_from_the_group = 3; //- удалить элемент из группы
const int CM_Print = 4; //s  Вывести информацию об элементах груп
const int CM_inform_all_el_of_the_group = 5; //z Вывести информацию о названиях всех элементов группы
const int CM_Quit = 6; //q Конец работы

const int EV_Nothing = 0; //левый символ
const int EV_Message = 100; //


class TEvent {
public:
        int what; //тип события
        int Command;
};
```

Файл Class_8.cpp

```
#include "Dialog.h"
using namespace std;

int main(){
   system("chcp 1251 > Null");
   Dialog dialog;
   dialog.Execute();
}
```

Результаты работы

```
------------------------------------------------------------
Что вы хотите сделать?
m:   Создать группу
+:   Добавить элемент в группу
-:   Удалить элемент из группы
s:   Вывести информацию об элементах группы
z:   Вывести информацию о названиях всех элементов группы
q:   Конец работы
>  m
Ваша команда: m

Введите размер: 6
------------------------------------------------------------
Что вы хотите сделать?
m:   Создать группу
+:   Добавить элемент в группу
-:   Удалить элемент из группы
s:   Вывести информацию об элементах группы
z:   Вывести информацию о названиях всех элементов группы
q:   Конец работы
>  +
Ваша команда: +

Какой класс создать?
1. Print
2. Magazine
2

Введите имя автора: Иван
Введите название: мурзилка
Введите количество страниц: 123
------------------------------------------------------------
Что вы хотите сделать?
m:   Создать группу
+:   Добавить элемент в группу
-:   Удалить элемент из группы
s:   Вывести информацию об элементах группы
z:   Вывести информацию о названиях всех элементов группы
q:   Конец работы
>  +
Ваша команда: +

Какой класс создать?
1. Print
2. Magazine
1

Введите имя автора: петров
Введите название: книга
```

```
------------------------------------------------------------
Что вы хотите сделать?
m:   Создать группу
+:   Добавить элемент в группу
-:   Удалить элемент из группы
s:   Вывести информацию об элементах группы
z:   Вывести информацию о названиях всех элементов группы
q:   Конец работы
>  s
Ваша команда: s

Автор: Иван. Название: мурзилка. Количество страниц: 123
Автор: петров. Название: книга

------------------------------------------------------------
Что вы хотите сделать?
m:   Создать группу
+:   Добавить элемент в группу
-:   Удалить элемент из группы
s:   Вывести информацию об элементах группы
z:   Вывести информацию о названиях всех элементов группы
q:   Конец работы
>  -
Ваша команда: -

Введите название, по которому надо удалить книга
Удаление успешно завершено
------------------------------------------------------------
Что вы хотите сделать?
m:   Создать группу
+:   Добавить элемент в группу
-:   Удалить элемент из группы
s:   Вывести информацию об элементах группы
z:   Вывести информацию о названиях всех элементов группы
q:   Конец работы
>  z
Ваша команда: z

Название: мурзилка

------------------------------------------------------------
Что вы хотите сделать?
m:   Создать группу
+:   Добавить элемент в группу
-:   Удалить элемент из группы
s:   Вывести информацию об элементах группы
z:   Вывести информацию о названиях всех элементов группы
q:   Конец работы
>  q
Ваша команда: q
```

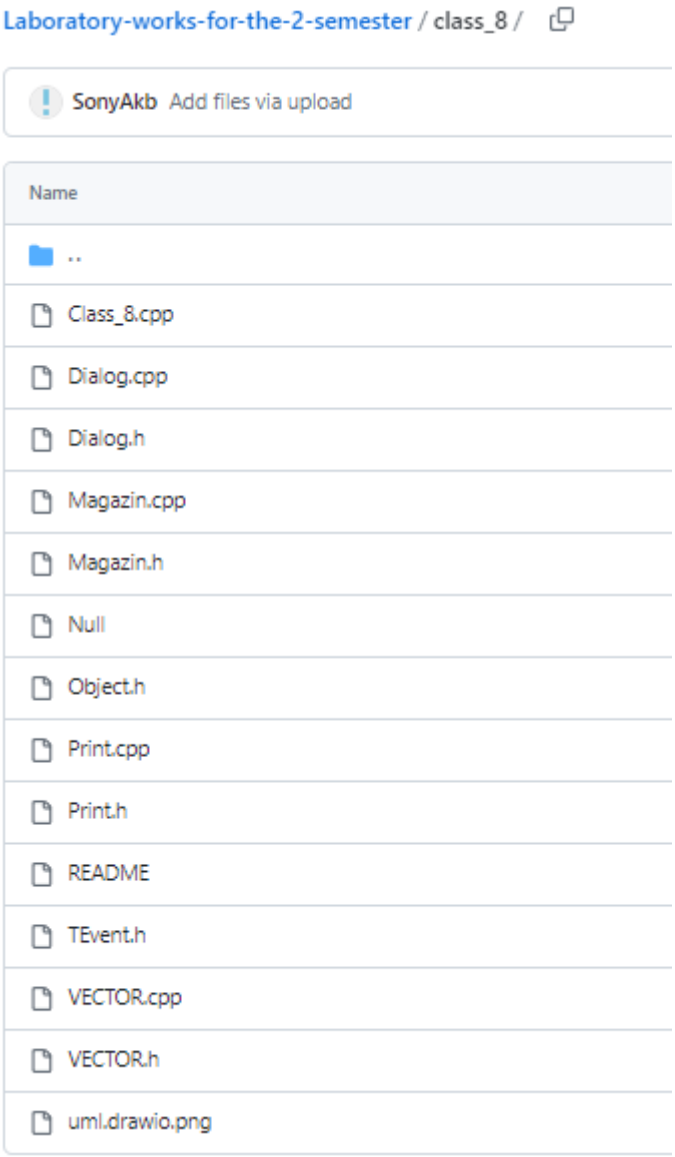
Вывод

В ходе работы я применила знания о программе, управляемой событием. Была определена иерархия пользовательских классов. В программу были добавлены методы для обработки событий группой и объектами пользовательских классов. В ходе работы был определен класс Диалог –

наследник группы, в котором реализованы методы для обработки событий. В основной функции были продемонстрированы методы для обработки событий группой и объектами пользовательских классов.

GitHub

Ссылка: https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/class_8



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.

2. 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;
        }
```

4. 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
        }
```

5. Привести пример метода для просмотра элементов для класса-группы Список.

```
void List::Show() {
    if (size == 0) cout << "Empty" << endl;
    Node* p = head;
    for (int i = 0; i < size; i++) {
        p->data->Show();
        p = p->next;
    }
}
```

6. 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.

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

8. 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.

9. 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.

10. Привести пример структуры, описывающей событие.

```
struct MoveEvent {
    std::string eventName;
    std::string player;
    int startRow;
    int startCol;
    int endRow;
    int endCol;
};
```

11. 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.

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?

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.

13. 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

14. 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).

15. Какой вид имеет главный цикл обработки событий-сообщений?

```
void Dialog:: Execute() {
        TEvent Event;
        while (!end_of_work) {
                Get_Event(Event);
                Handle_Event(Event);
        }
        }
```

16. Какую функцию выполняет метод ClearEvent()? Каким образом?

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

17. 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);
}
}
```

18. 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.

19. 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).

20. 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.