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: 09.03.04 - «Программная инженерия»

Лабораторная работа №8

По теме **«Программа, управляемая событиями»**

Вариант №10

Выполнял:

студент группы РИС-24-1б

Морозова Н.С.

Проверял:

Доцент кафедры ИТАС

Полякова О.А.

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 – Список (List).

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. Описать базовый класс Object и реализовать от него производные классы Print и Magazin, через которые будут вводиться и выводиться данные.
2. Описать класс-контейнер Vector, где будут храниться элементы группы.
3. Через классы TEvent и Dialog реализовать команды-события, через которые можно управлять данными.

Решение

Код

Файл Lab\_8.cpp

#include "dialog.h"

#include <iostream>

#include <Windows.h>

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "m<кол-во элементов> : создать группу" << endl;

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

cout << "- : удалить последний элемент" << endl;

cout << "s : показать все объекты " << endl;

cout << "z: вывести все названия" << endl;

cout << "q : закончить диалог" << endl;

Dialog dlg;

dlg.Execute(); // Запуск диалогового режима

return 0;

}

Файл Object.h

#pragma once

#include <iostream>

using namespace std;

class Object

{

public:

Object(void) { }

virtual void Show() = 0;

virtual void Input() = 0;

virtual ~Object(void) { }

};

Файл Vector.h

#pragma once

#include <iostream>

#include "Object.h"

class Vector

{

protected:

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

int size;//размер

int cur;//текущая позиция

public:

Vector(int);//конструктор с параметрами

~Vector();//деструктор

void Add();

void Add(Object\*);//добавление элемента в вектор

void Del();

void Show();

int operator()();//размер вектора

};

Файл Print.h

#pragma once

#include "Object.h"

#include <iostream>

#include <string>

using namespace std;

class Print : public Object

{

protected:

string name;

string author;

public:

Print();//конструктор без параметров

virtual ~Print(void) { } //деструктор

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

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

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

Print(const Print&);//конструктор копирования

//селекторы

string Get\_name() { return name; }

string Get\_author() { return author; }

//модификаторы

void Set\_name(string N) { name = N; }

void Set\_author(string A) { author = A; }

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

};

Файл magazin.h

#pragma once

#include "Print.h"

#include <iostream>

#include <string>

using namespace std;

class Magazin : public Print

{

protected:

int num;

int n;

string\* list;

public:

Magazin();

Magazin(string, string, int, int, string\* );

Magazin(const Magazin&);

~Magazin() { if (list != nullptr) delete[] list; }

void Show();

void Input();

int Get\_num();

int Get\_n();

string Get\_list();

void Set\_num(int);

void Set\_n(int);

void Set\_list(string\*);

Magazin& operator=(const Magazin&);

};

Файл dialog.h

#pragma once

#include "Event.h"

#include "Vector.h"

class Dialog : public Vector

{

protected:

int EndState;

public:

Dialog() : Vector(0), EndState(0) {} //контруктор по умолчанию

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»

};

Файл Event.h

#pragma once

const int evNothing = 0; //пустое 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 evMessage = 100; //непустое событие

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

};

};

};

Файл Print.cpp

#include "Print.h"

#include "Event.h"

#include "Object.h"

#include <iostream>

using namespace std;

Print::Print(void){

name = ' ';

author = ' ';

}

Print::Print(string N, string A){//констрктор с параметрами

name = N;

author = A;

}

Print::Print(const Print& pr){//конструктор копирования

name = pr.name;

author = pr.author;

}

Print& Print::operator=(const Print& pr){//оператор присваивания

if (&pr == this)

return \*this;

name = pr.name;

author = pr.author;

return \*this;

}

void Print::Show(){//метод для просмотра атрибутов

cout << "\nНазвание: " << name;

cout << "\nАвтор: " << author;

cout << endl;

}

void Print::Input(){//метод для ввода значений атрибутов

cout << "\nНазвание: ";

cin >> name;

cout << "Автор: ";

cin >> author;

}

Файл magazin.h

#include "Print.h"

#include "magazin.h"

#include <iostream>

using namespace std;

Magazin::Magazin() : Print(), num(0), n(0), list(nullptr) {}

Magazin::Magazin(string N, string A, int Num, int pag, string\* L) : Print(N, A), num(Num), n(pag){

if (pag > 0) {

list = new string[pag];

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

list[i] = L[i];

}

else

list = nullptr;

}

Magazin::Magazin(const Magazin& M):Print(M.name, M.author), num(M.num) {

n = M.n;

num = M.num;

if (num > 0) {

list = new string[num];

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

list[i] = M.list[i];

}

else list = nullptr;

}

int Magazin::Get\_num() { return num; }

int Magazin::Get\_n() { return n; }

string Magazin::Get\_list() { return (num > 0) ? list[0] : ""; }

void Magazin::Set\_num(int n) { num = n; }

void Magazin::Set\_n(int N) { n = N; }

void Magazin::Set\_list(string\* L) {

if (list != nullptr)

delete[] list;

if (num > 0) {

list = new string[num];

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

list[i] = L[i];

}

else list = nullptr;

}

Magazin& Magazin::operator=(const Magazin& M){

if (&M == this)

return \*this;

name = M.name;

author = M.author;

if (list != nullptr)

delete[] list;

n = M.num;

num = M.num;

if (num > 0) {

list = new string[num];

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

list[i] = M.list[i];

}

else

list = nullptr;

return \*this;

}

void Magazin::Show(){

Print::Show(); // Вызываем метод базового класса

cout << "Количество страниц: " << num << endl;

if (n > 0) {

cout << "Список статей/разделов: " << endl;

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

cout << " " << (i + 1) << ") " << list[i] << endl;

}

}

void Magazin::Input(){

Print::Input(); // Вводим данные базового класса

cout << "Введите количество страниц: ";

cin >> num;

if (list != nullptr)

delete[] list;

cout << "Введите количество статей/разделов: ";

cin >> n;

if (n > 0) {

list = new string[n];

cout << "Вводите названия статей/разделов:" << endl;

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

cout << " Статья " << (i + 1) << ": ";

cin >> list[i];

}

}

else list = nullptr;

}

Файл Vctor.cpp

#include "Vector.h"

#include "Print.h"

#include "magazin.h"

#include <iostream>

using namespace std;

Vector::~Vector(void){//деструктор

if (beg != 0)

delete[] beg;

beg = 0;

}

Vector::Vector(int n){//конструктор с параметрами

beg = new Object \* [n];

cur = 0;

size = n;

}

void Vector::Add(){//добавление объекта, на который указывает указатель

Object\* p; //выбор из объектов двух возможных классов

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

cout<<"2.Magazin"<<endl;

int y;

cout << "> ";

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 return;

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

Magazin \*b = new Magazin;

b->Input();

p = b;

if (cur < size) {

beg[cur] = p;

cur++;

}

} else return;

}

void Vector::Add(Object\* p) {

if (cur < size) {

beg[cur] = p; // Добавляем переданный объект

cur++;

}

}

void Vector::Show(){//просмотр вектора

if (cur == 0)

cout << "Empty" << endl;

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

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

(\*p)->Show();

p++;

}

}

int Vector::operator()() { return cur; } //возвращает размер вектора

void Vector::Del(){//удаление элемента из вектора, память не освобождается

if (cur == 0)

return;//пустой

cur--;

}

Файл dialog.cpp

#include "dialog.h"

#include "Print.h"

#include <iostream>

#include <string>

using namespace std;

void Dialog::GetEvent(TEvent& event){//получение события

string input;

cout << "> ";

cin >> input;

if (input.empty()) {

event.what = evNothing;

return;

}

char code = input[0];

event.what = evMessage;

switch (code) {

case 'm': event.command = cmMake; break; // Создать группу

case '+': event.command = cmAdd; break; // Добавить объект

case '-': event.command = cmDel; break; // Удалить объект

case 's': event.command = cmShow; break; // Показать всю группу

case 'z': event.command = cmGet; break; // Показать названия

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

default: event.what = evNothing; return; // Неизвестная команда

}

if (input.length() > 1){// Обработка параметров ('m10' — размер группы)

string param = input.substr(1);

event.a = stoi(param); // Преобразуем в число

}

else event.a = 0; // Параметр по умолчанию

}

int Dialog::Execute(){

TEvent event;

do {

GetEvent(event); //получить событие

HandleEvent(event); //обработать событие

} while (!Valid());

return EndState;

}

void Dialog::HandleEvent(TEvent& event){//обработчик событий

if (event.what != evMessage) return;

switch (event.command) {

case cmMake: // Создать группу (m N)

if (event.a > 0) {

size = event.a;

beg = new Object \* [size];

cur = 0;

}break;

case cmAdd: // Добавить объект (+)

Add();

break;

case cmDel: // Удалить объект (-)

Del();

break;

case cmShow: // Показать всю группу (s)

Show();

break;

case cmGet: // Показать названия (z)

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

Print\* p = dynamic\_cast<Print\*>(beg[i]);

if (p) cout << p->Get\_name() << endl;

}break;

case cmQuit: // Выход (q)

EndExec();

break;

}

ClearEvent(event);

}

int Dialog::Valid() { return EndState; }

void Dialog::ClearEvent(TEvent& event) { //пустое событие

event.what = evNothing;

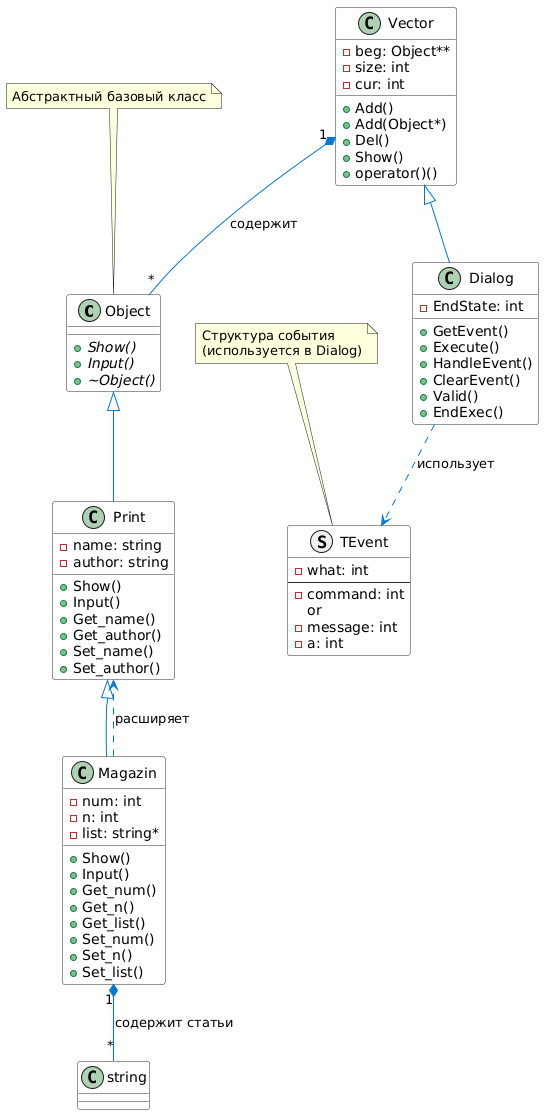
event.command = 0;

event.a = 0;

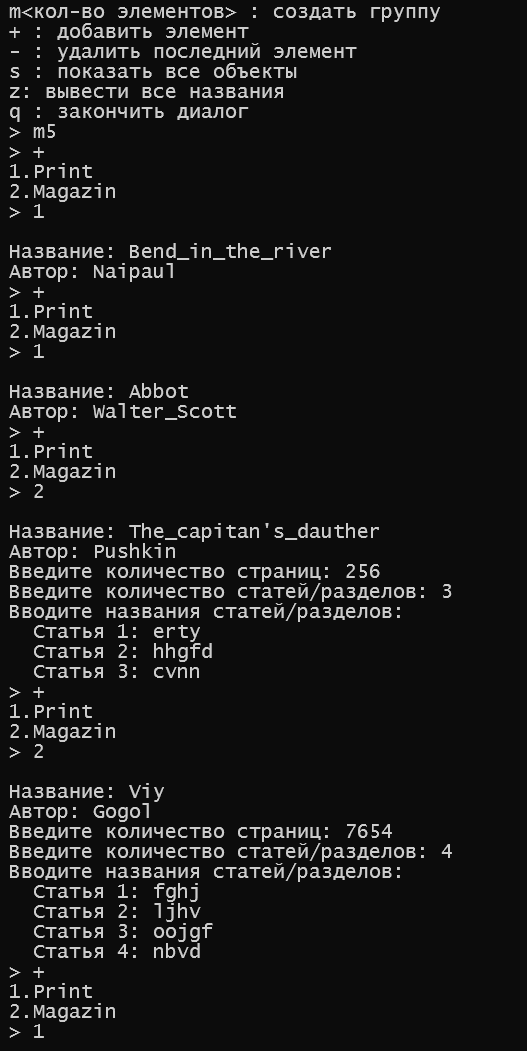
}

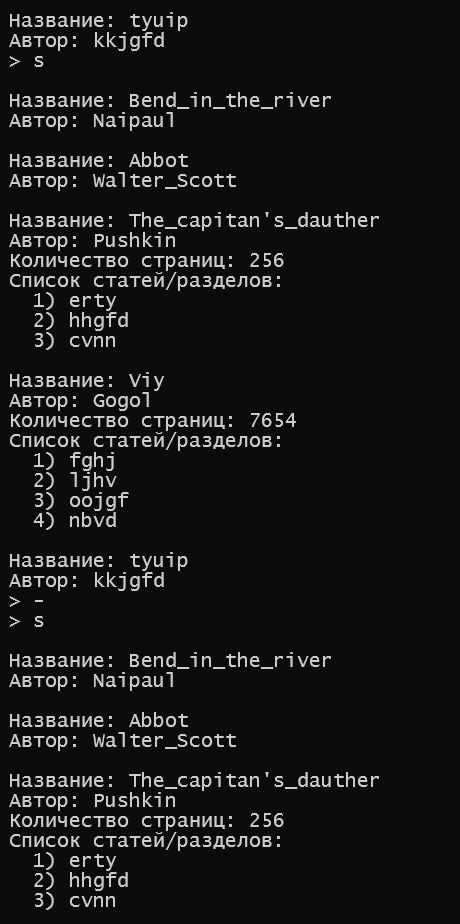
void Dialog::EndExec() { EndState = 1; }

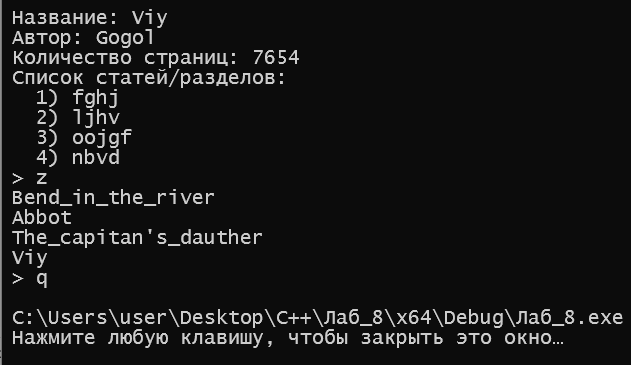
UML-диаграмма



Результат работы программы:







Контрольные вопросы:

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

Класс-группа – это класс, который управляет коллекцией (группой) объектов других классов, обеспечивая их хранение, добавление, удаление и обработку.

Примеры:

List (список) – хранит элементы в виде последовательности.

Stack (стек) – работает по принципу LIFO (последний вошел, первый вышел).

Queue (очередь) – работает по принципу FIFO (первый вошел, первый вышел).

Tree (дерево) – иерархическая структура данных.

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

class List {

private:

struct Node {

int data;

Node\* next;

Node(const int value):

data(value), next(nullptr) {}

};

Node\* head;

public:

List(); // Конструктор по умолчанию

List(const List& other); // Конструктор копирования

~List(); // Деструктор

void push\_back(const int value); // Добавление элемента

void print() const; // Просмотр элементов

};

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

// Конструктор без параметров (по умолчанию)

List::List() : head(nullptr) {}

// Конструктор копирования

List::List(const List& other) : head(nullptr) {

Node\* current = other.head;

while (current) {

push\_back(current->data);

current = current->next;

}

}

// Конструктор с параметром

List::List(const int value): head(new Node(value)) {}

1. Привести пример деструктора для класса-группы Список.

List::~List() {

Node\* current = head;

while (current) {

Node\* next = current->next;

delete current;

current = next;

}

head = nullptr;

}

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

void List::print() const {

Node\* current = head;

while (current) {

cout << current->data << ‘ ‘;

current = current->next;

}

cout << endl;

}

1. Какой вид иерархии дает группа?

Группа реализует агрегацию (часть-целое), где класс-группа содержит объекты других классов, но они могут существовать независимо.

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

Абстрактный класс (с чистыми виртуальными методами) позволяет:

1. Унифицировать интерфейс обработки объектов в группе.
2. Реализовать полиморфизм (разные подклассы могут обрабатываться единообразно).
3. Запретить создание экземпляров базового класса.
4. Что такое событие? Для чего используются события?

Событие – это уведомление о каком-либо действии (например, клик мыши, нажатие клавиши).

Использование:

1. Обработка пользовательского ввода.
2. Реакция на изменения в системе.
3. Реализация паттерна Наблюдатель (Observer).
4. Какие характеристики должно иметь событие-сообщение?

Тип события (например, MouseClick, KeyPress).

Данные события (координаты мыши, код клавиши).

Источник события (объект, который его вызвал).

Временная метка (момент возникновения).

1. Привести пример структуры, описывающей событие.

struct Event {

enum class Type { MouseClick, KeyPress, Timer };

Type type;

union {

struct { int x, y; } mouseData;

struct { int keyCode; } keyData;

};

void\* source; // Указатель на объект-источник

time\_t timestamp;

};

1. Задана структура события

struct TEvent{

int what;

union

{

MouseEventType mouse;

KeyDownEvent keyDown;

MessageEvent message;

}

};

Какие значения, и в каких случаях присваиваются полю what?

Поле what определяет тип события. В зависимости от его значения, используется соответствующее поле в union:

what = 1 → событие мыши (mouse).

what = 2 → нажатие клавиши (keyDown).

what = 3 → сообщение (message).

1. Задана структура события

struct TEvent{

int what; //тип события

union{

int command; //код команды

struct{ //параметры команды

int message;

int a;

};

};

};

Какие значения, и в каких случаях присваиваются полю command?

Поле command содержит код команды, если what указывает на командное событие.

what = 100 → событие-команда, тогда command содержит её код (например, 1 = "Сохранить", 2 = "Открыть").

Если what не связан с командой, command не используется.

1. 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 – идентификатор сообщения,

a – дополнительный параметр (например, координата, флаг).

1. Какие методы необходимы для организации обработки сообщений?

GetEvent() – получение события из очереди.

HandleEvent() – обработка события.

ClearEvent() – очистка события.

DispatchEvent() – перенаправление события нужному обработчику.

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

while (!shouldExit) {

TEvent event;

if (GetEvent(event)) { // Получаем событие

HandleEvent(event); // Обрабатываем

ClearEvent(event); // Очищаем

}

}

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

Сбрасывает состояние события, чтобы избежать повторной обработки. Например:

void ClearEvent(TEvent& e) {

e.what = 0; // Обнуляем тип

e.command = 0; // Очищаем команду

}

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

Определяет, как обрабатывать событие на основе what и данных в union. Например:

void HandleEvent(TEvent& e) {

switch (e.what) {

case 1: // Мышь

cout << "Mouse at: " << e.mouse.x << ‘,’ << e.mouse.y << endl;

break;

case 2: // Клавиша

cout << "Key pressed: " << e.keyDown.code << endl;

break;

}

}

1. Какую функцию выполняет метод GetEvent()? Каким образом?

Извлекает событие из системной очереди (или генерирует его). Например:

bool GetEvent(TEvent& e) {

if (есть\_событие\_в\_очереди) {

e = очередь\_событий.front();

очередь\_событий.pop();

return true;

}

return false;

}

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

Флаг EndState указывает, что программа должна завершиться (например, после команды "Выход").

Используется обычно в классе приложения (TApplication):

class TApplication {

bool EndState;

public:

void Run() {

while (!EndState) { // Обработка событий }

}

};

1. Для чего используется функция Valid()?

Проверяет, корректно ли событие или состояние объекта.

bool TEvent::Valid() const {

return (what >= 0 && what < MAX\_EVENT\_TYPES);

}

if (event.Valid()) {

HandleEvent(event);

}