Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «Классы и объекты. Инкапсуляция»

Выполнил студент ИВТ-22-2б:

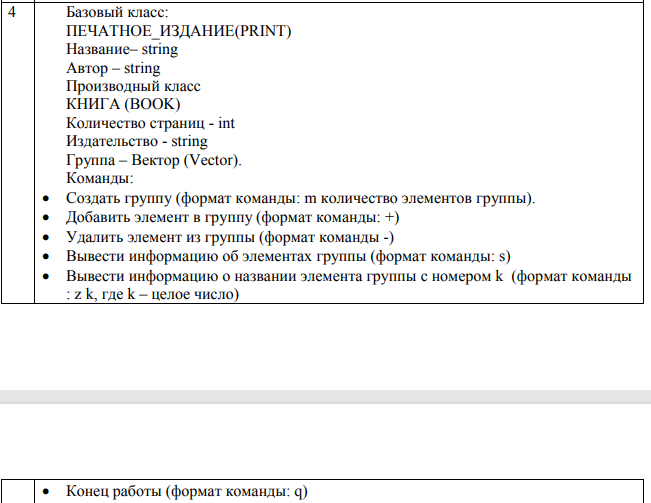
Захаров Дмитрий Сергеевич

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**1 Постановка задачи**

****

**2 Анализ задачи**

В классе Dialog реализованы методы для обработки событий, класс BOOK является производным классом от класса Print.

Для организации обработки событий необходимы следующие методы:

1. Execute - реализация главного цикла обработки событий. Данный метод постоянно получает событие путем вызова GetEvent и обрабатывает их с помощью HandleEvent.

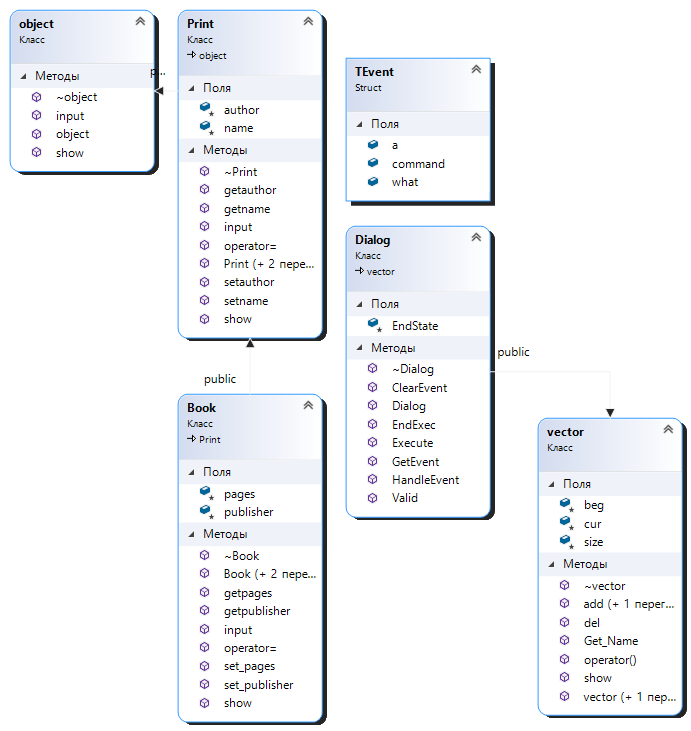
2. GetEvent - метод формирования события

3. HandleEvent - метод обработки событий

4. ClearEvent - метод очистки текущего события

5. EndExec - завершение обработки событий(после вызова этого метода цикл обработки событий заканчивается)

**3 UML-диаграммы**

****

**4 Код программы**

#include "Book.h"

#include "Print.h"

#include "Vector.h"

#include "Event.h"

#include "Dialog.h"

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

system("chcp 1251");

system("cls");

/\*Print\* a = new Print;

a->input();

Book\* b = new Book;

b->input();

vector A(10);

object\* p = a;

cout << "Добавление издания в вектор: " << endl;

A.add(p);

A.show();

cout << "Добавление книги в вектор: " << endl;

p = b;

A.add(p);

A.show();

A.add();

A.show();

cout << "Размер вектора: " << A();

A.del();\*/

Dialog D;

cout << "Команды:\nm создать группу\n+ добавить элемент\n- удалить элемент\ns вывести информацию об элементах\nq конец работы\nz Информация по элементу с номером k" <<endl;

D.Execute();

return 0;

}

**Object.h**

#pragma once

class object {

public:

object(void);

virtual void show() = 0;

virtual void input() = 0;

virtual ~object(void);

};

object::object(void) {

};

object::~object(void) {

};

**Print.h**

#pragma once

#include "Object.h"

#include <string>

#include <iostream>

using namespace std;

class Print :public object {

protected:

string name;

string author;

public:

Print(void);

virtual ~Print(void);

void show();

void input();

Print(string, string);

Print(const Print&);

string getname() { return name; };

string getauthor() { return author; };

void setname(string);

void setauthor(string);

Print& operator=(const Print&);

};

Print::Print(void) {

name = "";

author = "";

};

Print::Print(string name, string author) {

name = name;

author = author;

};

Print::Print(const Print& a) {

this->author = a.author;

this->name = a.name;

};

Print::~Print(void) {

};

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

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

name = a.name;

author = a.author;

return \*this;

};

void Print::setname(string m) {

name = m;

};

void Print::setauthor(string a) {

author = a;

};

void Print::show() {

cout << endl << "Издание: " << endl;

cout << "Название: " << name << endl;

cout << "Автор: " << author << endl;

};

void Print::input() {

cout << "Издание: " << endl << "Введите название книги: ";

getchar();

getline(cin, name);

cout << "Введите имя автора: ";

getline(cin, author);

};

**Vector.h**

#pragma once

#include "Object.h"

#include "Book.h"

#include "Print.h"

#include <string>

#include <iostream>

#include "Event.h"

using namespace std;

class vector {

protected:

object\*\* beg;

int size;

int cur;

public:

vector();

vector(int);

~vector(void);

void add(object\*);

void add();

void del();

void Get\_Name();

void show();

int operator()();

};

void vector::Get\_Name() {

object\*\* p = beg;

int n;

cout << "Введите номер: ";

cin >> n;

for (int i = 1; i <= n; i++) {

if (i == n)

(\*p)->show();

p++;

}

};

vector::vector() {

beg = 0;

size = 0;

cur = 0;

};

vector::~vector(void) {

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

beg = 0;

};

vector::vector(int n) {

beg = new object \* [n];

cur = 0;

size = n;

};

void vector::add(object\* p) {

if (cur < size) {

beg[cur] = p;

cur++;

}

};

void vector::add() {

object\* p;

cout << "1.Издание" << endl;

cout << "2.Книга" << endl;

int y;

do {

cout << "Введите 1 или 2: ";

cin >> y;

} while (y > 2 || y < 1);

if (y == 1) {

Print\* a = new Print;

a->input();

p = a;

if (cur < size) {

beg[cur] = p;

cur++;

}

}

else {

if (y == 2) {

Book\* b = new Book;

b->input();

p = b;

if (cur < size) {

beg[cur] = p;

cur++;

}

}

}

};

void vector::show() {

if (cur == 0) cout << "empty" << endl;

object\*\* p = beg;

cout << "ВЕКТОР: " << endl;

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

(\*p)->show();

p++;

}

};

int vector::operator()() {

return cur;

};

void vector::del() {

if (cur == 0) return;

cur--;

};

**Book.h**

#pragma once

#include "Print.h"

#include <string>

#include<iostream>

using namespace std;

class Book :public Print {

protected:

int pages;

string publisher;

public:

Book(void);

Book(string, string, int, string);

Book(const Book&);

int getpages() { return pages; };

string getpublisher() { return publisher; };

void set\_pages(int);

void set\_publisher(string);

Book& operator=(const Book&);

~Book(void);

void show();

void input();

};

Book::Book(void) :Print() {

pages = 0;

publisher = "";

};

Book::~Book(void) {

};

Book::Book(string name, string author, int pages, string publisher) :Print(name, author)

{

this->pages = pages;

this->publisher = publisher;

};

Book::Book(const Book& a) {

pages = a.pages;

publisher = a.publisher;

name = a.name;

author = a.author;

};

void Book::set\_pages(int pages) {

this->pages = pages;

};

void Book::set\_publisher(string p) {

publisher = p;

};

Book& Book::operator=(const Book& a) {

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

name = a.name;

author = a.author;

return \*this;

};

void Book::show() {

cout << endl << "Книга: " << endl;

cout << "Название: " << name << endl;

cout << "Автор: " << author << endl;

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

cout << "Издательство: " << publisher << endl;

};

void Book::input() {

cout << "Книга: " << endl << "Введите название книги: ";

getchar();

getline(cin, name);

cout << "Введите имя автора: ";

getline(cin, author);

cout << "Введите количество страниц: ";

cin >> pages;

cout << "Введите издательствo: ";

getchar();

getline(cin, publisher);

};

**Dialog.h**

#pragma once

#include <iostream>

#include "Vector.h"

#include "Event.h"

#include "Book.h"

#include <string>

using namespace std;

class Dialog :public vector {

protected:

int EndState;

public:

Dialog();

virtual ~Dialog();

virtual void GetEvent(TEvent&);

virtual int Execute();

virtual void HandleEvent(TEvent&);

virtual void ClearEvent(TEvent&);

bool Valid();

void EndExec();

};

Dialog::Dialog() {

EndState = 0;

};

Dialog::~Dialog() {

};

void Dialog::GetEvent(TEvent& event) {

string opint = "+-szqam";

string s, param;

char code;

cout << '>';

cin >> s;

code = s[0];

if (opint.find(code) >= 0) {

event.what = evMessage;

switch (code)

{

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

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

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

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

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

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

};

int Dialog::Execute() {

TEvent event;

do {

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (Valid());

return EndState;

};

bool Dialog::Valid() {

return EndState == 0;

};

void Dialog::ClearEvent(TEvent& event) {

event.what = evNothing;

};

void Dialog::EndExec() {

EndState = 1;

};

void Dialog::HandleEvent(TEvent& event) {

if (event.what == evMessage) {

switch (event.command) {

case cmMake:

cout << "Введите размер: ";

cin >> size;

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:Get\_Name();

ClearEvent(event);

break;

};

}

};

**Event.h**

#pragma once

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;

struct TEvent {

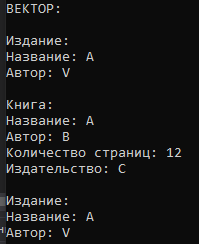
int what;

int command;

int a;

};

**5 Результаты работы программы**

****

**6 Ответы на 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, тоже могут быть группой.

Примеры групп:

- Окно в интерактивной программе, которое владеет такими элементами, как поля ввода и редактирования данных, кнопки, списки выбора, диалоговые окна и т.д. Примерами таких окон являются объекты классов, порождённых от абстрактного класса TGroup (TDeskTop, TWindow, TDialog) в иерархии классов библиотеки Turbo Vision, и объекты классов, порождённых от TWindowObject в иерархии классов библиотеки OWL.

- Агрегат, состоящий из более мелких узлов.

- Огород, состоящий из растений, системы полива и плана выращивания.

- Некая организационная структура (например, ФАКУЛЬТЕТ, КАФЕДРА, СТУДЕНЧЕСКАЯ ГРУППА).

Класс-группа Vector:

class Vector

{

protected:

Object\*\* beg;

int size;

int cur;

public:

Vector(int);

~Vector();

void Add();

void Del();

void Show();

int operator()();

};

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

struct Node

{

Node\* prev = nullptr, \* next = nullptr;

Object\* data;

};

struct Object

{

Object\* beg;

Object();

Object(Object\*);

~Object() {};

virtual void popback() = 0;

virtual void pushback() = 0;

virtual void Show() = 0;

};

class List

{

public:

List(int);//конструктор с параметрами

public:

List();

List(List& t);

~List(void);//деструктор

void popback();//добавление элемента в вектор

void pushback();

void Show();

int operator()();//размер вектора

protected:

Node\* head = nullptr, \* tail = nullptr;

int size;

};

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

List::List()

{

head = 0;

tail = 0;

size = 0;

}

List::List(int a)

{

for(int i = 1; i <= size; i++) pushback();

}

List::List(List& t)

{

Node\* nodeA = t.head;

List newList(t.size);

Node\* newNode = newList.head;

while (nodeA != nullptr)

{

newNode->data = nodeA->data;

nodeA = nodeA->next;

newNode = newNode->next;

}

}

1. Привести пример деструктора для класса-группы Список.

List::~List()

{

Node\* nodeA = head;

while (nodeA != nullptr)

{

head = nodeA->next;

delete nodeA->data;

delete nodeA;

nodeA = head;

}

}

1. 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 = 1; i <= size; i++)

{

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

p = p->next; //передвигаем указатель на следующий объект

}

}

1. Какой вид иерархии даёт группа?

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

Реализовать группу можно разными способами.

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

Абстрактные классы используются в качестве обобщённых концепций, на основе которых можно создавать более конкретные производные классы.

Другими словами, абстрактный класс создаётся для того, чтобы связать классы с общими характеристиками в одну группу.

1. Что такое событие? Для чего используются события?

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 мышью порождают событие, которое передаётся по цепочке объектов, пока не найдётся 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.

1. Какие характеристики должно иметь событие-сообщение?

Сообщение передаваемое от одних объектов другим имеет, как правило, следующие характеристики:

- код класса сообщения, отличающий сообщения объектов одного класса от объектов другого класса

- адрес объекта, которому предназначено сообщение (м. б. не задан, тогда сообщение могут прочитать все объекты)

- информационное поле

1. Привести пример структуры, описывающей событие.

struct TEvent

{

int what;

int command;

};

1. Задана структура события

struct TEvent

{

int what;

union

{

MouseEventType mouse;

KeyDownEvent keyDown;

MessageEvent message;

}

};

Какие значения, и в каких случаях присваиваются полю what?

- evNothing – это пустое событие, которое означает, что ничего делать не надо. Полю what присваивается значение evNothing, когда событие обработано каким-либо объектом.

- evMessage событие-сообщение от объекта.

Для события от объекта (evMessage) задаются два параметра :

- command – код команды, которую необходимо выполнить при появлении данного события;

- передаваемая с событием информация (сообщение).

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?

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

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;

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?

Эти поля необходимы тогда, когда программа получает сообщение с параметром. Поле message здесь будет отвечать за то, какая команда поступила, а поле a будет являться параметром команды.

1. Какие методы необходимы для организации обработки сообщений?

class Dialog :

{

public:

Dialog(void);

~Dialog(void); // деструктор

void GetEvent(TEvent& event); // получение события

void Execute(); // главный цикл обработки

void HandleEvent(TEvent& event); // обработка события

void ClearEvent(TEvent& event); // удаление события

bool Valid(); // провека атрибута завершения обработки

void EndExec(); // обработчик конца работы

protected:

int EndState; // конечное положение

};

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

void Dialog::Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (Valid());

}

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

ClearEvent очищает событие, присваивая полю event.what значение evNothing, когда оно обработано, чтобы оно не обрабатывалось далее.

void Dialog::ClearEvent(TEvent& event)

{

event.what = evNothing;

}

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

HandleEvent – обработчик событий. Обрабатывает каждое событие нужным для него образом. Если объект должен обрабатывать определённое 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).

void Dialog::HandleEvent(TEvent& event)

{

if (event.what == evMessage)

{

switch (event.command)

{

case cmMake:

cout << "Enter size: ";

cin >> size;

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:

Get\_Name();

ClearEvent(event);

break;

}

}

}

1. Какую функцию выполняет метод GetEvent ()?

Метод GetEvent() формирует событие:

void Dialog::GetEvent(TEvent& event)

{

setlocale(LC\_ALL, "ru");

string OpInt = "m+-szq";

string s;

char code;

cout << "Введите команду: ";

cin >> s;

code = s[0];

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

{

event.what = evMessage;

switch (code)

{

case 'm':

event.command = cmMake;

break;

case '+':

event.command = cmAdd;

break;

case '-':

event.command = cmDel;

break;

case 's':

event.command = cmShow;

break;

case 'z':

event.command = cmGet;

break;

case 'q':

event.command = cmQuit;

break;

}

}

else event.what = evNothing;

}

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

Поле EndState содержится в классе-обработчике Dialog. Оно проверяет, окончил ли пользователь свою работу:

void Dialog::EndExec()

{

EndState = 1;

}

1. Для чего используется функция Valid()?

Функция Valid проверяет значение поля EndState:

- Если программа не завершена, она возвращает значение истины

- Если программа завершена, то значение лжи

Функция используется в главном обработчике событий Execute.

class Dialog : public Tree

{

public:

...

bool Valid(); // провека атрибута завершения обработки

...

};

void Dialog::Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (Valid());

}