Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа  
“Классы”. №8**

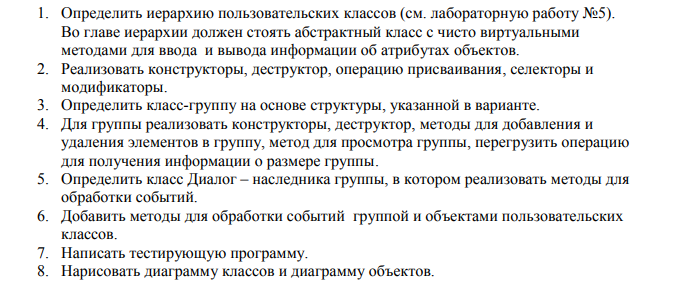
Выполнил:   
студент группы РИС-23-1б   
Фаезов Александр

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

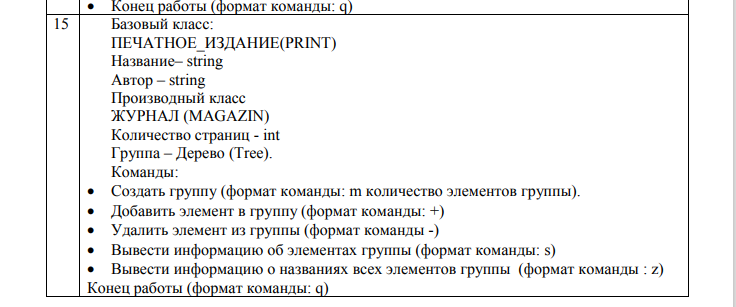
Пермь, 2024 г.

**«Классы и объекты. Программа, управляемая событиями»**

**Анализ задачи:**

****

***Вариант 15:***

****

**Код на языке C++:**

#include <iostream>

#include <string>

using namespace std;

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;

union

{

int command;

struct

{

int message;

int a;

};

};

};

class Object

{

public:

Object() {};

~Object() {};

virtual void show() = 0;

virtual void GetName() = 0;

virtual void input() = 0;

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

};

class Print : public Object

{

protected:

string name;

string author;

public:

void GetName() {

cout << name << endl;

}

Print() {

name = "no";

author = "no";

}

Print(string name, string author) {

this->name = name; this->author = author;

}

void show() {

cout << "Издание: " << name << "\nАвтор: " << author << endl;

}

void input() {

cout << "Введите издание: ";

cin >> this->name;

cout << "Введите имя автора: ";

cin >> author;

}

void setName(string name) {

this->name = name;

}

void setAuthor(string author) {

this->author = author;

}

string getName() {

return name;

}

string getAuthor() {

return author;

}

void HandlEvent(const TEvent& e) {

if (e.what==evMessage)

switch (e.command)

{

case cmGet:

cout << "Название: " << getName() << endl;

break;

default:

break;

}

}

};

class Magazin : public Print

{

protected:

int pages;

public:

void GetName() {

cout << name << endl;

}

Magazin() :Print::Print() {

pages = 0;

}

Magazin(string n, string a, int p) : Print::Print(n, a) {

pages = p;

}

void show() {

cout << "Издание: " << name

<< "\nАвтор: " << author << "\nКоличество страниц: " << pages << endl;

}

void input() {

cout << "Введите издание: ";

cin >> this->name;

cout << "Введите имя автора: ";

cin >> author;

cout << "Введите количество страниц: ";

cin >> pages;

}

int getPages() {

return pages;

}

void setPages(int p) {

pages = p;

}

Magazin& operator=(const Magazin& a) {

if (&a == this)

return \*this;

name = a.name;

author = a.author;

pages = a.pages;

return \*this;

}

};

class Vector

{

protected:

Object\*\* beg;

int size;

int cur;

public:

Vector() {

beg = 0;

size = 0;

cur = 0;

}

void Add()

{

Object\* p;

cout << "1. Издательство" << endl << "2. Журнал" << endl;

int x;

cin >> x;

if (x == 1)

{

Print\* a = new Print;

a->input();

p = a;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else if (x == 2)

{

Magazin\* b = new Magazin;

b->input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

}

~Vector() {

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

beg = 0;

}

Vector(int n) {

beg = new Object \* [n];

cur = 0;

size = n;

}

void Get\_Name() {

Object\*\* p = beg;

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

(\*p)->GetName();

p++;

}

}

void Show() {

if (cur == 0) {

cout << "Пусто." << endl;

}

Object\*\* p = beg;

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

(\*p)->show();

p++;

}

}

void Del() {

if (cur == 0)

return;

cur--;

}

void HandlEvent(const TEvent& e) {

if (e.what == evMessage) {

Object\*\* p = beg;

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

(\*p)->HandlEvent(e);

p++;

}

}

}

};

class Dialog : public Vector {

protected:

int EndState;

public:

Dialog() {

EndState = 0;

}

~Dialog() {

}

void ClearEvent(TEvent& event) {

event.what = evNothing;

}

void EndExec() {

EndState = 1;

}

void Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (Valid());

}

bool Valid()

{

return EndState == 0;

}

void GetEvent(TEvent& event) {

string OpInt = "m+-szq";

string s;

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('z'): event.command = cmGet; break;

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

}

}

else {

event.what = evNothing;

}

}

void HandleEvent(TEvent& event) {

if (event.what == evMessage) {

switch (event.command)

{

case cmMake:

cout << "Введите размер группы: ";

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;

}

}

}

};

int main() {

system("chcp 1251 > null");

cout << "m: Создать группу" << endl;

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

cout << "-: Убрать элемент" << endl;

cout << "s: информация о членах в группе" << endl;

cout << "z: информация о названиях элементов группы" << endl;

cout << "q: Конец работы" << endl;

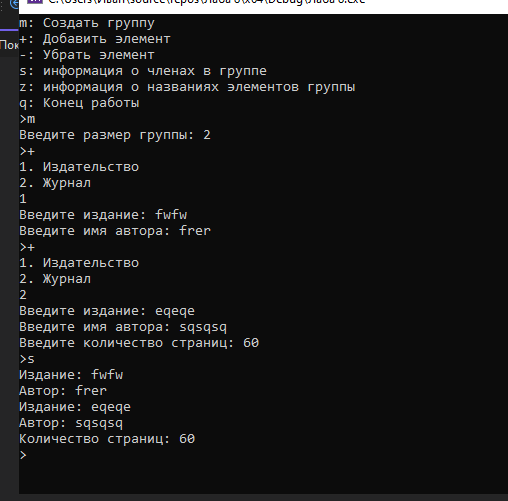
Dialog D;

D.Execute();

return 1;

}

**Работа программы:**



**Ответ на вопросы:**

1. Класс-группа - это класс, который объединяет в себе набор объектов одного типа или связанных объектов для логической организации и управления ими. Примеры таких классов могут быть классы "Студент" со списком студентов университета, "Заказ" с товарами в корзине интернет-магазина и т.д.

2. Пример описания класса-группы Список (List):

class List {

private:

int\* items;

int maxSize;

int currentSize;

public:

List();

List(int size);

List(const List& other);

~List();

void ViewItems();

// Другие методы класса

};

3. Пример конструкторов для класса-группы Список:

List(); // Конструктор без параметров

List(int size); // Конструктор с параметром

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

4. Пример деструктора для класса-группы Список:

~List() {

delete[] items;

}

5. Пример метода для просмотра элементов для класса-группы Список:

void ViewItems() {

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

cout << items[i] << " ";

}

cout << endl;

}

6. Группа дает иерархию, в которой объекты могут быть организованы и управляться в логически связанных структурах. Это позволяет упорядочить объекты по их свойствам и взаимосвязям для удобного доступа к ним и их обработки.

7. Абстрактный класс обычно находится во главе иерархии, потому что он позволяет объединить общие атрибуты и методы для всех классов-наследников под ним. Это содействует упрощению кода и позволяет структурировать логику программы.

8. Событие - это сигнал или уведомление о каком-либо произошедшем событии (например, действии пользователя), которое требуется обработать в программе. События используются для реагирования на внешние воздействия, взаимодействия программы с пользователем или другими компонентами системы.

9. Характеристики события-сообщения:

- Тип события.

- Дополнительные данные или параметры события.

- Признак того, что событие требует обработки.

10. Пример структуры, описывающей событие:

struct TEvent {

int what;

union {

int mouse;

int keyDown;

};

int message;

};

11. Полю `what` присваиваются различные значения в зависимости от типа события. Например, для MouseEventType это может быть значение, обозначающее событие от мыши, а для KeyDownEvent - значение, обозначающее нажатую клавишу.

12. Полю `command` присваиваются различные коды команд в зависимости от типа события или действия, которое требуется выполнить в программе.

13. Поля `message` и `a` используются для хранения дополнительных параметров или данных команды, которые могут потребоваться при обработке события.

14. Для организации обработки сообщений необходимы методы для получения, обработки и реагирования на события. Например, обработчики событий, функции для анализа и вызова соответствующих действий.

15. Главный цикл обработки событий-сообщений может иметь вид бесконечного цикла, который ожидает и обрабатывает поступающие события в программе.

16. Метод `ClearEvent()` выполняет очистку события, устанавливая поле `what` объекта события в значение, обозначающее отсутствие события.

17. Метод `HandleEvent()` используется для обработки событий, определения типа события и выполнения соответствующих действий в зависимости от кода команды.

18. Метод `GetEvent()` используется для получения событий, взаимодействия с внешними источниками событий или пользователем, и передачи их на дальнейшую обработку.

19. Поле `EndState` используется для представления состояния завершения диалога или программы. Класс `Dialog` содержит это поле для управления окончанием выполнения программы.

20. Функция `Valid()` используется для проверки состояния выполнения программы (например, завершена ли работа) и возвращает булевое значение, истину если состояние работоспособности программы соблюдается и ложь в противном случае.**Вывод:** Задача была выполнена. Всё получилось.

