Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №8

Дисциплина: Основы алгоритмизации и программирования

Тема: Программа управляемая событиями

|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Деревнин И.В. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

**Анализ предметной области**

**Постановка задачи**

1. Определить иерархию пользовательских классов. Во главе иерархии должен стоять абстрактный класс с чисто виртуальными функциями.
2. Реализовать конструкторы, деструкторы, 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 группа 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.

Вариант 15:

Базовый класс: ПЕЧАТНОЕ\_ИЗДАНИЕ(PRINT)

Название – string, автор – string

Производный класс: ЖУРНАЛ (MAGAZINE)

Количество страниц int.

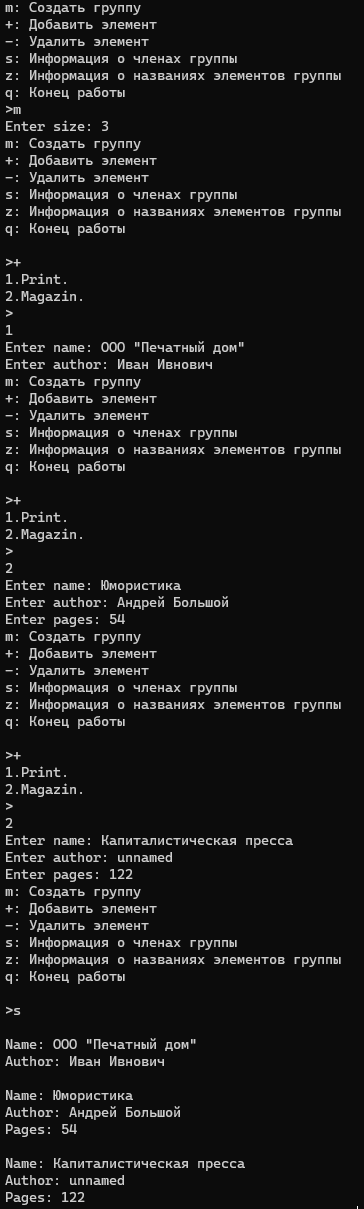
Группа - дерево (Tree)

Команды:

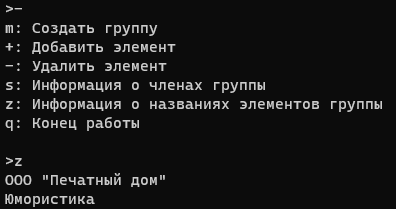
* Создать группу (формат команды: m количество элементов группы)
* Добавить элемент в группу (формат команды: +)
* Удалить элемент из группы (формат команды: -)
* Вывести информацию об элементах группы (формат команды: s)
* Вывести информацию о названиях всех элементов группы (формат команды: z)
* Конец работы (формат команды: q)

**Анализ задачи**

Необходимо реализовать иерархию пользовательских классов, для этого необходимо создать абстрактный класс object с виртуальными методами: input, show и GetName. Далее от него необходимо унаследовать два класса magazine и print, в них переопределить виртуальные методы. Далее необходимо реализовать класс Tree, который будет обладать методами: добавление и удаление элементов, вывод элементов в консоль и вывод имен в консоль. От данного класса необходимо унаследовать класс Dialog, который будет использовать Tree для управления данными. Для метода Dialog также необходимо создать структуру, отвечающую за сигнал и набор констант, для более наглядного кода. Набор констант должен содержать в себе команды, содержащиеся в варианте.

****Тестирование программы**

*Рис. 1 – Создание группы, добавление элементов, вывод группы в консоль.*

*Рис. 2 – Удаление элемента и вывод имен группы в консоль.*

**Заключение**

# Была разработана программа, которая представляет из себя обработчик событий по добавлению элементов в структуру данных.

# 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.

В отличие от контейнера мы понимаем группу как класс, который не только хранит объекты других классов, но и обладает собственными свойствами, не вытекающими из свойств его элементов.

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

Пример:

Класс-группа Vector. Группа содержит указатель на динамический массив указателей типа Object (Object \*\*beg), количество элементов, под которые выделена память (int size), номер последнего элемента, добавленного в группу (int cur).

class Vector {

public:

Vector(int);//конструктор с параметрами

public:

~Vector(void);//деструктор

void Add();//добавление элемента в вектор

void Del();

void Show();

int operator()();//размер вектора

protected:

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

int size;//размер

int cur;//текущая позиция };

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

struct Node {

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

Object\* data;

};

class List {

public:

List();

List(int);

List(List&);

~List(void);

void puchback();

void popback();

void Show();

protected:

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

int size;

};

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

Без параметров:

List::List() {

size = 0;

head = nullptr;

tail = nullptr;

}

С параметрами:

List::List(int s) {

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

pushback();

}

Копирования:

List::List(const List& a) {

Node\* nodeA = a.head;

List newList(a.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\* curr = head;

while (curr != nullptr) {

head = curr->next;

delete curr->data;

delete curr;

curr = 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 = 0;i < size;i++) {

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

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

}

}

1. 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?

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах.

Таким образом, можно создать функцию, параметром которой является указатель на абстрактный класс. На место этого параметра при выполнении программы может передаваться указатель на объект любого производного класса. Это позволяет создавать полиморфные функции, работающие с объектом любого типа в пределах одной иерархии.

Именно поэтому во главе иерархии классов должен находится абстрактный класс.

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 передается по цепочке объектов, пока не найдется объект, знающий, как обрабатывать это событие. События сами по себе не производят никаких действий в программе, но в ответ на событие могут создаваться новые объекты, модифицироваться или уничтожаться существующие, что и приводит к изменению состояния программы. Иными словами все действия по обработке данных реализуются объектами, а события лишь управляют их работой.

События предназначены для того, чтобы иметь возможность предусмотреть реакцию программы на то или иное действие.

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.

enum Event

{

evNothing = 0, //пустое событие

evMessage = 100,//непустое событие

cmAdd = 1, // +

cmDel = 2, // -

cmGet = 3, // z

cmShow = 4, // s

cmMake = 5, // m

cmQuit = 101 // q

};

struct TEvent {

int what; //наличие события

int command; //тип комманды

};

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 задает тип события, определяющий источник данного события. Оно может принимать следующие значения:

* 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 (непустое событие).

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?

Полю command присваиваются коды различных определённых команд. При получении того или иного сообщения, поле command принимает одно из кодов команд (см. пример из вопроса 10).

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*?

Поля *a* и *message* необходимы в случае, если программа получает сообщение с параметром. Тогда поле *message* будет отвечать за то, какая команда поступила, а поле *a* будет являться параметром команды.

Пример:

Пользователь вводит следующее сообщение:

*+10*

В этом случае, программа сначала посмотрит, какую команду мы получили (+ - добавить в группу) и на основе этого запишет в поле *message* заданную команду, а затем посмотрит, есть ли после символа + целочисленное значение. Далее, программа запишет введенное значение в поле *a*, и после этого обработчик события будет знать, что необходимо добавить 5 элементов в группу.

1. Какие методы необходимы для организации обработки сообщений?

Следующие методы необходимы для организации обработки событий (названия произвольны).

* GetEvent – формирование события;
* Execute реализует главный цикл обработки событий. Он постоянно получает событие путем вызова GetEvent и обрабатывает их с помощью HandleEvent. Этот цикл завершается, когда поступит событие «конец».
* HandleEvent – обработчик событий. Обрабатывает каждое событие нужным 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 распознаваться, например, по коду команды (поле command).
* ClearEvent очищает событие, когда оно обработано, чтобы оно не обрабатывалось 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?

Главный цикл обработки событий реализуется в методе Execute главной группы-объекта “прикладная программа” по следующей схеме:

int TMyApp::Execute() {

do {

endState=0;

GetEvent(event); //получить событие

HandleEvent(event); //обработать событие

//событие осталось не обработано

if(event.what!=evNothing)

EventError(event);

} while(!Valid());

return endState;

}

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

ClearEvent очищает событие, присваивая полю event.what значение evNothing.

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

Получив событие (структуру типа TEvent), обработчик событий для класса TDerivedClass обрабатывает его по следующей схеме:

void TDerivedClass::HandleEvent(TEvent& event) {

//Вызов обработчика событий базового класса

TBaseClass::HandleEvent( event );

if (event.what == evCommand) { // Если обработчик базового

// событие не обработал

switch( event.message.command ) {

case cmCommand1:

// Обработка команды cmCommand1;

СlearEvent( event );

break;

case cmCommand2:

// Обработка команды cmCommand2;

СlearEvent( event );

break; …

case cmCommandN:

// Обработка команды cmCommandN

СlearEvent( event );

break;

default: // событие не обработано

break;

} };

}

1. Какую функцию выполняет метод GetEvent()?

Метод GetEvent() выполняет функцию формирования события на основе полученного сообщения.

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

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

class Dialog :

public Vector

{

public:

Dialog(void);

~Dialog(void);//деструктор

void GetEvent(TEvent& event);//получить событие

void Execute();//главный цикл обработки событий

void HandleEvent (TEvent& event); //обработчик

void ClearEvent (TEvent & event);//очистить событие

bool Valid();//проверка атрибута EndState

void EndExec();//обработка события «конец работы»

protected:

int EndState;

};

1. Для чего используется функция Valid()?

Функция Valid проверяет состояние поля EndState. Она возвращает значение True, если программа не завершена, False - если завершена. Используется в главном обработчике событий Execute.

bool Dialog::Valid() {

return EndState == 0;

}

void Dialog::Execute() {

TEvent event;

do {

EndState = 0;

GetEvent(event); //получить событие

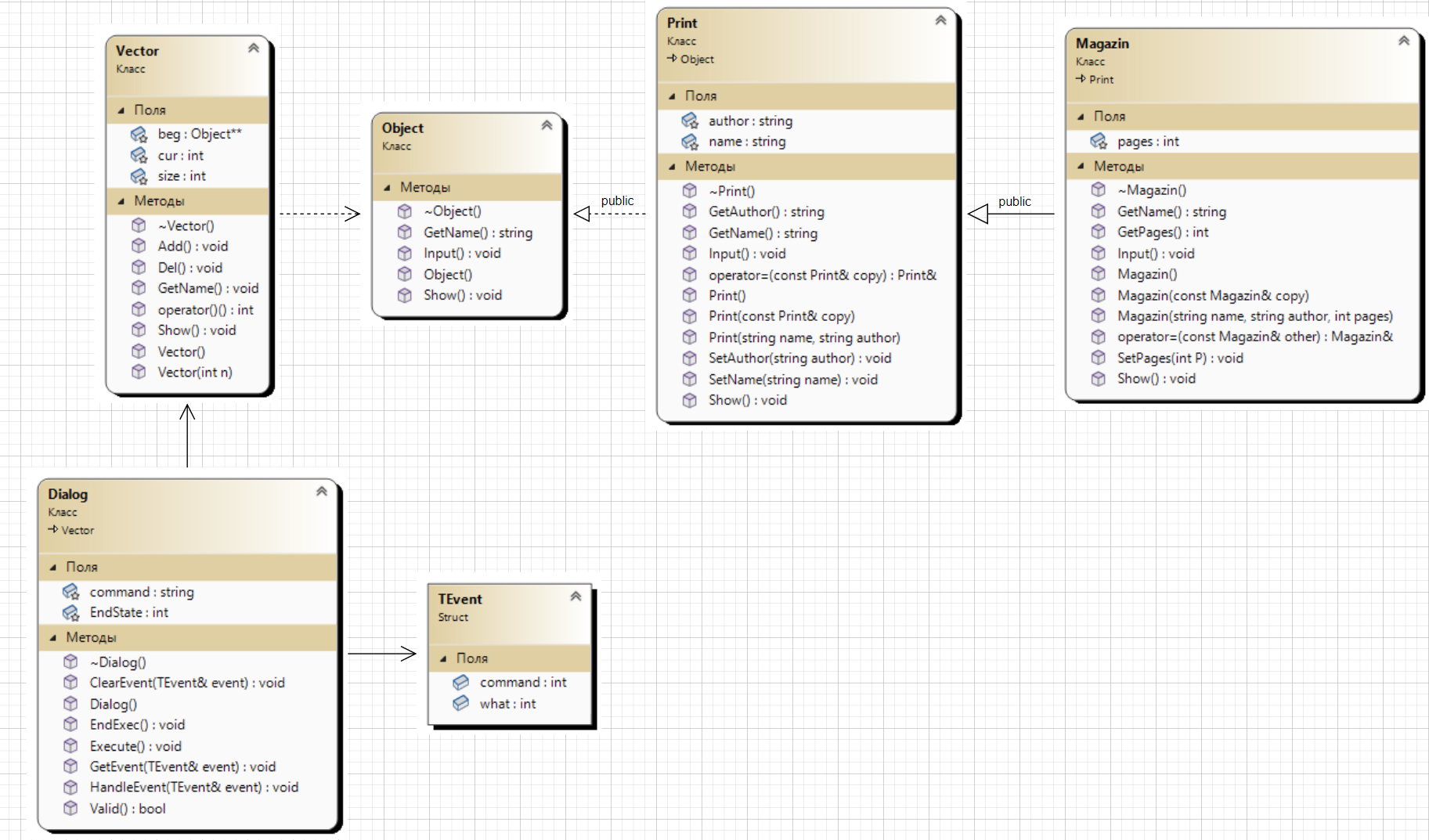
HandleEvent(event); //обработать событие

} while (Valid());

}

**Приложения**

UML-диаграмма:



Приложение Б – код программы

Main.cpp:

#include "Tree.h"

#include "Dialog.h"

#include "Event.h"

#include "Magazin.h"

#include "Object.h"

#include "Print.h"

#include <Windows.h>

int main()

{

system("chcp 1251 >> null");

cout << "m: Создать группу\n";

cout << "+: Добавить элемент\n";

cout << "-: Удалить элемент\n";

cout << "s: Информация о членах группы\n";

cout << "z: Информация о названиях элементов группы\n";

cout << "q: Конец работы\n";

Dialog D;

D.Execute();

return 0;

}list.h:

#pragma once

#include <iostream>

class List

{

private:

void removeAt(int index);

//минимальная ячейка памяти

class Node //вложенный класс - ячейка памяти

{

public:

Node(int data = int(), Node\* pNext = nullptr) //используется параметр по умолчанию, то есть если мы не передаем указатель, то он автоматически присваивается nullptr

{ //для поля data также используется параметр по умолчанию, выглядит он как вызов конструктора по умолчанию

this->data = data;

this->pNext = pNext;

}

Node\* pNext; //хранит указатель на следующий элемент

int data; //хранит наши данные

};

Node\* head; //указатель на первый элемент в спискеint

int size; //количество элементов в списке.

public:

class iterator //вложнный итератор

{

private:

friend class List;

Node\* head; //указатель на node

Node\* current;

public:

iterator(); //конструкторы итератора / деструкторы

~iterator();

iterator(List& lst);

void reset();

void operator +(int index); //операторы итератора

void operator ++(int);

void operator ++();

int& operator \*();

bool operator ==(iterator it);

bool operator !=(iterator it);

};

List(); //Конструктор

List(List& copy);//!!!!!!

~List(); //Деструктор

List& operator=(const List& other);//!!!!!!!!!!

Node\* push\_back(int data); //добавляет элемент в конец списка

int GetSize() { return size; } //так как инкапсуляция не дает нам доступ к количеству элементов в списке, то нам необходимо происать геттер

int& operator[](const int index); //перегрузка оператора [] для того, чтобы мы могли взаимодействовать со списком как с массивом

void pop\_front(); //удаляет элемент из начала списка

void clear(); //удаляет все элементы списка (логика как у pop\_front, деструктор основан на этой штуке)

void push\_front(int data); //добавляет элемент в начало списка

void removeAt(List::iterator& iter); //удаление элемента по указаному индексу

void pop\_back(); //удаление из конца списка

iterator begin();

iterator end();

void removeFewAt(int index\_start, int index\_finish); //удаление нескольких элементов

friend class iterator;

void insert(int value, List::iterator& it);

List operator\*(const List& other);

};

tree.h:

#include "Tree.h"

#include "Dialog.h"

#include "Event.h"

#include "Magazin.h"

#include "Object.h"

#include "Print.h"

#include <Windows.h>

int main()

{

system("chcp 1251 >> null");

cout << "m: Создать группу\n";

cout << "+: Добавить элемент\n";

cout << "-: Удалить элемент\n";

cout << "s: Информация о членах группы\n";

cout << "z: Информация о названиях элементов группы\n";

cout << "q: Конец работы\n";

Dialog D;

D.Execute();

return 0;

}

tree.cpp:

#include "Tree.h"

Tree::Tree()

{

beg = nullptr;

size = 0;

cur = 0;

}

Tree::~Tree()

{

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

beg = nullptr;

}

Tree::Tree(int n)

{

beg = new Object\* [n];

cur = 0;

size = n;

}

void Tree::GetName()

{

Object\*\* p = beg;

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

{

cout << (\*p)->GetName() << endl;

p++;

}

}

void Tree::Add()

{

Object\* p;

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

cout << "2.Magazin.\n>" << endl;

int f;

cin >> f;

if (f == 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 if (f == 2) //добавление объекта класса Magazin

{

Magazin\* b = new Magazin;

b->Input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else return;

}

void Tree::Show()

{

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

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

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

{

(\*p)->Show(); //вызов метода Show()

p++; //передвигаем указатель на следующий объект

}

}

int Tree::operator ()()

{

return cur;

}

void Tree::Del()

{

if (cur == 0) return; //Если пустой

cur--;

}

Object.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Object //чисто абстрактный класс

{

public:

Object() {}

~Object() {}

virtual void Show() = 0; //чисто виртуальные методы класса

virtual void Input() = 0;

virtual string GetName() = 0;

};

Print.h:

#pragma once

#include "Object.h"

class Print :

public Object

{

public:

Print(); //конструктор/деструкторы

Print(string name, string author);

Print(const Print& copy);

virtual ~Print();

void Show(); //переопределенные методы ввода/вывода

void Input();

string GetName() { return name; } //сеттеры и геттеры

string GetAuthor() { return author; }

void SetName(string name) { this->name = name; }

void SetAuthor(string author) { this->author = author; }

Print& operator=(const Print& copy); //оператор присваивания

protected:

string name;

string author;

};

Print.cpp:

#include "Print.h"

Print::~Print() {}

Print::Print()

{

name = "";

author = "";

}

Print::Print(string naem, string author)

{

this->name = name;

this->author = author;

}

Print::Print(const Print& copy)

{

this->name = copy.name;

this->author = copy.author;

}

void Print::Show()

{

cout << "\nName: " << name;

cout << "\nAuthor: " << author << endl;

}

void Print::Input()

{

cin.get();

cout << "Enter name: "; getline(cin, name);

cout << "Enter author: "; getline(cin, author);

}

Print& Print::operator=(const Print& copy)

{

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

name = copy.name;

author = copy.author;

return \*this;

}

Magazine.h:

#pragma once

#include "Print.h"

class Magazin :

public Print

{

public:

Magazin(); //конструкторы и деструктор

Magazin(string name, string author, int pages);

Magazin(const Magazin& copy);

~Magazin();

int GetPages() { return pages; } //сеттеры/геттеры

void SetPages(int P) { pages = P; }

string GetName() { return name; } //переопределенные методы

void Show();

void Input();

Magazin& operator=(const Magazin& other); //оператор присваивания

protected:

int pages;

};

Magazine.cpp:

#include "Magazin.h"

Magazin::~Magazin() {}

Magazin::Magazin(){

pages = 0;

}

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

{

this->pages = pages;

}

Magazin::Magazin(const Magazin& copy)

{

name = copy.name;

author = copy.author;

pages = copy.pages;

}

void Magazin::Show()

{

cout << "\nName: " << name;

cout << "\nAuthor: " << author;

cout << "\nPages: " << pages << endl;

}

void Magazin::Input()

{

cin.get();

cout << "Enter name: "; getline(cin, name);

cout << "Enter author: "; getline(cin, author);

cout << "Enter pages: "; cin >> pages;

}

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

{

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

name = other.name;

author = other.author;

pages = other.pages;

return \*this;

}

Event.h:

#pragma once

enum Event

{

evNothing = 0, //пустое событие

evMessage = 100,//непустое событие

cmAdd = 1, // +

cmDel = 2, // -

cmGet = 3, // z

cmShow = 4, // s

cmMake = 5, // m

cmQuit = 101 // q

};

struct TEvent {

int what; //наличие события

int command; //тип комманды

};

Dialog.h:

#pragma once

#include "Event.h"

#include "Tree.h"

class Dialog : public Tree

{

public:

Dialog(); //конструктор/деструктор

~Dialog();

void HandleEvent (TEvent& event); //обработчик

void GetEvent(TEvent& event); //получить событие

void Execute(); //главный цикл обработки событий

void ClearEvent (TEvent& event); //очистить событие

bool Valid(); //проверка атрибута EndState

void EndExec(); //обработка события «конец работы»

protected:

string command;

int EndState;

};

Dialog.cpp:

#include "Dialog.h"

Dialog::Dialog() :Tree()

{

command = "m: Создать группу\n+: Добавить элемент\n-: Удалить элемент\ns: Информация о членах группы\nz: Информация о названиях элементов группы\nq: Конец работы\n\n";

EndState = 0;

}

Dialog::~Dialog() {}

void Dialog::GetEvent(TEvent& event)

{

string OpInt = "m+-szq"; //строка содержит коды операций

string s;

char code;

cout << '>';

cin >> s; code = s[0];

if (OpInt.find(code) >= 0) //является ли символ кодом операции

{

event.what = evMessage;

switch (code)

{

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

case '+': event.command = Event::cmAdd; break; //добавить объект в группу

case '-': event.command = Event::cmDel; break; //удалить объект из группы

case 's': event.command = Event::cmShow; break; //просмотр группы

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

case 'q': event.command = Event::cmQuit; break; //конец работы

}

}

else event.what = evNothing;//пустое событие

}

void Dialog::Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event); //получить событие

HandleEvent(event); //обработать событие

} while (Valid());

}

bool Dialog::Valid()

{

return EndState == 0;

}

void Dialog::ClearEvent(TEvent & event)

{

event.what = evNothing;//пустое событие

cout << "\033[2J\033[1;1H";

cout << command;

}

void Dialog::EndExec()

{

EndState = 1;

}

void Dialog::HandleEvent(TEvent& event)

{

if (event.what == evMessage)

{

switch (event.command)

{

case Event::cmMake: //создание группы

cout << "Enter size: ";

cin >> size;

beg = new Object \* [size]; //выделяем память под массив указателей

cur = 0; //текущая позиция

ClearEvent(event); //очищаем событие

break;

case Event::cmAdd: //добавление

Add();

ClearEvent(event);

break;

case Event::cmDel:

Del(); //удаление

ClearEvent(event);

break;

case Event::cmShow:

Show(); //просмотр

system("pause");

ClearEvent(event);

break;

case Event::cmQuit:

EndExec(); //выход

ClearEvent(event);

break;

case Event::cmGet:

GetName();

system("pause");

ClearEvent(event);

break;

};

};

}