Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Кафедра **«**Информационные технологии и автоматизированные системы**»**

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Семестр 2

Тема: Программа, управляемая событиями

Выполнил работу

Студент группы РИС-22-1Б

Лихачев Д.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Постановка задачи**



**Вариант 9**



**Диаграмма классов**



**Код программы**

**Book.h**

#pragma once

#include "Print.h"

#include <string>

#include <iostream>

using namespace std;

class Book :

public Print

{

public:

Book(void);

public:

~Book(void);

void Show();

void Input();

Book(string, string, int, string);

Book(const Book&);

int Get\_countpages();

string Get\_publishing\_house();

void Set\_countpages(int);

void Set\_publishing\_house(string);

Book& operator=(const Book&);

protected:

int countpages;

string publishing\_house;

};

**Book.cpp**

#include "Book.h"

#include <string>

#include <iostream>

using namespace std;

Book::Book(void) {

name = "";

writer = "";

countpages = 0;

publishing\_house = "";

}

Book::~Book(void) {

}

Book::Book(string name, string writer, int countpages, string publishing\_house) {

this->name = name;

this->writer = writer;

this->countpages = countpages;

this->publishing\_house = publishing\_house;

}

Book::Book(const Book& other) {

name = other.name;

writer = other.writer;

countpages = other.countpages;

publishing\_house = other.publishing\_house;

}

int Book::Get\_countpages() {

return countpages;

}

string Book::Get\_publishing\_house() {

return publishing\_house;

}

void Book::Set\_countpages(int countpages) {

this->countpages = countpages;

}

void Book::Set\_publishing\_house(string publishing\_house) {

this->publishing\_house = publishing\_house;

}

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

name = other.name;

writer = other.writer;

countpages = other.countpages;

publishing\_house = other.publishing\_house;

return \*this;

}

void Book::Show() {

cout << "\nNAME : " << name;

cout << "\nWRITER : " << writer;

cout << "\nCOUNTPAGES : " << countpages;

cout << "\nPUBLISHING HOUSE : " << publishing\_house;

cout << "\n";

}

void Book::Input() {

cout << "\nNAME: ";

getline(cin, name);

cout << "\nWRITER: ";

getline(cin, writer);

cout << "\nCOUNTPAGES: ";

cin >> countpages;

cin.ignore();

cout << "\nPUBLISHING HOUSE: ";

getline(cin, publishing\_house);

}

**Dialog.h**

#pragma once

#include "List.h"

#include "Event.h"

#include "Book.h"

#include <string>

#include <iostream>

using namespace std;

class Dialog :

public List

{

public:

Dialog();

virtual ~Dialog();

virtual void GetEvent(TEvent& event);

virtual int Execute();

virtual void HandleEvent(TEvent& event);

virtual void ClearEvent(TEvent& event);

int Valid();

void EndExec();

protected:

int Endstate;

};

**Dialog.cpp**

#include "List.h"

#include "Event.h"

#include "Book.h"

#include "Dialog.h"

#include <string>

#include <iostream>

using namespace std;

Dialog::Dialog(): List() {

Endstate = 0;

}

Dialog::~Dialog() {

}

//Получение события

void Dialog::GetEvent(TEvent& event) {

string OpInt = "m+-s/zq";

string s;

string param;

char code;

cout << ">";

cin >> s;

code = s[0];

cin.ignore();

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 '/':

event.command = cmGet;

break;

case 'q':

event.command = cmQuit;

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;

}

int Dialog::Valid() {

if (Endstate == 0) {

return 0;

}

else return 1;

}

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:

size = 0;

head = nullptr;

tail = nullptr;

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; //пустое событие

const int evMessage = 100; //непустое событие

const int cmAdd = 1; //добавить обьект в группу

const int cmDel = 2; //удалить обьект из группы

const int cmGet = 3; //вывести атрибут всх обьектов

const int cmShow = 4; //вывести всю группу

const int cmMake = 6; //создать группу

const int cmQuit = 101; //выход

//класс событие

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

**List.h**

#pragma once

#include "Book.h"

#include "Event.h"

#include <string>

#include <iostream>

using namespace std;

struct Node

{

Node\* prev, \* next;

Object\* data;

};

class List

{

public:

List();

~List();

void Add();

void Del();

void Show();

int operator()();

void pushback(Object\* other);

void Get\_name();

//void HandleEvent(const TEvent& e);

protected:

int size;

Node\* head, \* tail;

};

**List.cpp**

#pragma once

#include "List.h"

#include <string>

#include <iostream>

using namespace std;

List::List() {

size = 0;

head = nullptr;

tail = nullptr;

}

List::~List() {

Node\* curr = head;

while (curr != nullptr) {

head = curr->next;

delete curr;

curr = head;

}

}

void List::pushback(Object\* other) {

Node\* newNode = new Node;

newNode->data = other;

if (head == nullptr) {

head = newNode;

tail = newNode;

size++;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

size++;

}

}

void List::Del() {

if (head != nullptr) {

Node\* curr = tail;

tail = curr->prev;

delete curr;

tail->next = nullptr;

size--;

}

}

void List::Add() {

Object\* p;

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

cout << "2.Book" << endl;

int y;

cin >> y;

cin.ignore();

if (y == 1) {

Print\* a = new(Print);

a->Input();

p = a;

pushback(p);

}

else {

if (y == 2) {

Book\* b = new(Book);

b->Input();

p = b;

pushback(p);

}

}

}

void List::Show() {

if (size == 0) {

cout << "Empty" << endl;

}

Node\* node = head;

Object\* p = head->data;

while (node != nullptr) {

p = node->data;

p->Show();

node = node->next;

}

}

int List::operator()() {

return size;

}

void List::Get\_name() {

Node\* node = head;

Object\* p = head->data;

while (node != nullptr) {

p = node->data;

p->Get\_name();

node = node->next;

}

}

**Object.h**

#pragma once

class Object

{

public:

Object(void) {}

virtual void Show() = 0;

virtual void Input() = 0;

virtual string Get\_name() = 0;

~Object(void) {}

};

**Print.h**

#pragma once

#pragma once

#include "Object.h"

#include <string>

using namespace std;

class Print : public Object

{

public:

Print(void);//конструктор без параметров

public:

virtual ~Print(void);//деструктор

//void HandleEvent(const TEvent& e);

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();

string Get\_writer();

//модификаторы

void Set\_name(string);

void Set\_writer(string);

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

protected:

string name;

string writer;

};

**Print.cpp**

#include "Print.h"

#include "Event.h"

#include <string>

#include <iostream>

using namespace std;

Print::Print(void)

{

name = "";

writer = "";

}

//деструктор

Print::~Print(void)

{

}

//констрктор с параметрами

Print::Print(string name, string writer)

{

this->name = name;

this->writer = writer;

}

//конструктор копирования

Print::Print(const Print& Print)

{

name = Print.name;

writer = Print.writer;

}

/\*void Print::HandleEvent(const TEvent& e) {

if (e.what == evMessage) {

switch (e.command) {

case cmGet:

cout << "NAME = " << Get\_name() << endl;

break;

}

}

}\*/

//Селекторы

string Print::Get\_name() {

return name;

}

string Print::Get\_writer() {

return writer;

}

//Модификаторы

void Print::Set\_writer(string writer)

{

this->writer = writer;

}

void Print::Set\_name(string name)

{

this->name = name;

}

//оператор присваивания

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

{

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

name = c.name;

writer = c.writer;

return \*this;

}

//метод для просмотра атрибутов

void Print::Show()

{

cout << "\nNAME : " << name;

cout << "\nWRITER : " << writer;

cout << "\n";

}

//метод для ввода значений атрибутов

void Print::Input()

{

cout << "\nNAME:";

getline(cin, name);

cout << "\nWRITER:";

getline(cin, writer);

}

**Source.cpp**

#include "List.h"

#include "Event.h"

#include "Book.h"

#include "Dialog.h"

#include <string>

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "rus");

cout << "m - создать группу" << endl;

cout << "+ - добавить в группу" << endl;

cout << "- - удалить из группу" << endl;

cout << "s - показать группу" << endl;

cout << "q - выйти" << endl;

Dialog D;

D.Execute();

}

**Результат работы проргаммы**



**Ответы на контрольные вопросы**

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

Класс-группа - это класс, который содержит в себе коллекцию объектов других классов и предоставляет методы для работы с этой коллекцией.

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).

template <typename T>

class List {

public:

// Конструкторы и деструктор

List();

List(const List<T>& other);

~List();

// Операторы

List<T>& operator=(const List<T>& other);

T& operator[](int index);

const T& operator[](int index) const;

// Методы

void insert(int index, const T& value);

void remove(int index);

int size() const;

bool isEmpty() const;

private:

// Внутренний класс узла списка

struct Node {

T data;

Node\* next;

Node(const T& d) : data(d), next(nullptr) {}

};

Node\* head;

int length;

};

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

без параметров:



с параметром:



копирования:



4. Привести пример деструктора для класса-группы Список.



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



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?

В иерархии классов, содержащихся в группе объектов, на вершине должен находиться абстрактный класс, так как он определяет общий интерфейс, который должен быть реализован всеми классами-наследниками. Это позволяет упростить дальнейшую работу с объектами классов-наследников и обеспечить единообразие в их использовании. Кроме того, такая организация позволяет обеспечить полиморфизм и использовать указатели на базовый абстрактный класс для работы с объектами различных классов-наследников через общий интерфейс.

8. Что такое событие? Для чего используются события?

событие (event) представляет собой сигнал, сообщающий программе о каком-то действии, произошедшем в системе или приложении. События используются для организации обработки пользовательских действий, взаимодействия компонентов приложения, реализации асинхронной обработки и многих других задач.

9. Какие характеристики должно иметь событие-сообщение?

Объект TEvent состоит из двух частей. Первая (what) задает тип события, определяющий 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. Полю what присваивается значение evNothing, когда событие обработано каким-либо объектом.  
• evMessage - событие-сообщение от объекта.

10. Привести пример структуры, описывающей событие.



11. Задана структура события





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).

12. Задана структура события



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



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 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).

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



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



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

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

Метод HandleEvent() выполняет обработку события-сообщения, которое было передано в качестве аргумента метода. Обычно он содержит логику для обработки конкретного типа событий.

  
18. Какую функцию выполняет метод GetEvent ()?

Метод GetEvent() используется для получения информации о следующем событии в очереди событий. Если в очереди нет ни одного события, метод блокируется до появления нового события. Если в очереди есть события, метод возвращает информацию о первом событии в очереди и удаляет его из очереди. 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 состояние объекта допустимо, и false в противном случае.