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»

# Лабораторная работа №8

# «Программа, управляемая событиями»

# Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Зырянов Ростислав

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**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

**Вариант 15**

Базовый класс: ПЕЧАТНОЕ\_ИЗДАНИЕ(PRINT)

Название– string   
Автор – string

Производный класс ЖУРНАЛ (MAGAZIN)

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)

**Ответы на контрольные вопросы**

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

Класс-группа - это набор классов, имеющих сходный интерфейс и общую функциональность. Примеры классов-групп: списки, множества, стеки, очереди и т.д.

**2. Привести пример описания класса-группы Список (List).**

Класс-группа Список может быть описана следующим образом:

Пример кода на C++:

template <class T>

class List {

public:

List(); // конструктор по умолчанию

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

~List(); // деструктор

void push\_front(const T& value); // добавление элемента в начало списка

void push\_back(const T& value); // добавление элемента в конец списка

void pop\_front(); // удаление элемента из начала списка

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

T& front(); // доступ к первому элементу списка

T& back(); // доступ к последнему элементу списка

bool empty() const; // проверка, пустой ли список

size\_t size() const; // получение размера списка

private:

struct Node {

T data;

Node\* next;

Node\* prev;

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

};

Node\* head;

Node\* tail;

size\_t count;

};

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

Пример кода на C++:

// Конструктор по умолчанию

template <class T>

List<T>::List() : head(nullptr), tail(nullptr), count(0) {}

// Конструктор копирования

template <class T>

List<T>::List(const List<T>& other) : head(nullptr), tail(nullptr), count(0) {

Node\* current = other.head;

while (current != nullptr) {

push\_back(current->data);

current = current->next;

}

}

// Конструктор с параметром

template <class T>

List<T>::List(const T& value) : head(nullptr), tail(nullptr), count(0) {

push\_back(value);

}

**4. Привести пример деструктора для класса-группы Список.**

template <class T>

List<T>::~List() {

while (head != nullptr) {

Node\* current = head;

head = head->next;

delete current;

}

count = 0;

}

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

Пример кода на C++:

void Print() const {

Node\* curr = head;

while (curr != nullptr) {

std::cout << curr->data << " ";

curr = curr->next;

}

std::cout << std::endl;

}

**6. Какой вид иерархии дает группа?**

Группа объектов обычно дает иерархию наследования, где родительские классы предоставляют общие свойства и методы для дочерних классов, которые могут дополнительно расширять функциональность и добавлять уникальные свойства.

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

Абстрактный класс служит основой для дочерних классов и предоставляет общие методы и свойства, необходимые для всех классов в группе. Кроме того, абстрактный класс может определять чисто виртуальные методы, которые должны быть реализованы в каждом дочернем классе, обеспечивая единый интерфейс для работы с объектами разных классов.

Пример абстрактного класса на С++:

class Shape {

public:

virtual double getArea() const = 0; // чисто виртуальный метод

virtual double getPerimeter() const = 0; // чисто виртуальный метод

};

**8. Что такое событие? Для чего используются события?**

Событие - это действие, которое происходит в программе и может быть обработано соответствующим обработчиком. События используются для реализации обработки ввода пользователя, асинхронной обработки данных и обновления интерфейса приложения в реальном времени.

**9. Какие характеристики должно иметь событие-сообщение?**

Событие-сообщение должно содержать информацию о том, какое событие произошло, где оно произошло и когда. Кроме того, событие-сообщение может содержать дополнительные параметры, такие как данные, связанные с событием, и идентификатор объекта, который инициировал событие.

**10. Привести пример структуры, описывающей событие.**

Пример кода на C++:

struct MouseEvent

{

enum Type { Press, Release, Move } type;

int x;

int y;

int button;

};

Эта структура описывает событие мыши и содержит информацию о типе события (нажатие, отпускание или перемещение мыши), координатах x и y указателя мыши и коде кнопки мыши (если событие связано с нажатием или отпусканием кнопки мыши).

**11. 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, которое произошло. Значения этого поля могут быть определены разработчиком программы в зависимости от конкретной задачи и типов событий, которые необходимо обрабатывать.   
В данном случае будут присваиваться значения полю what «MouseEventType mouse; KeyDownEvent keyDown; MessageEvent message;»при различных действиях предусмотренных программой.

**12. 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?**

Поле command используется для хранения кода команды, которая вызвала событие. Значения поля command также могут быть определены разработчиком программы в зависимости от конкретной задачи. Например, для меню команды могут иметь следующие значения: открыть файл, сохранить файл, печать документа и т.д. Значение поля command присваивается в момент создания события и используется для определения команды при ее обработке.

Пример присваивания значения полю command:

TEvent event;

event.command = IDM\_OPEN\_FILE; // присваивание значения команде "открыть файл"

**1З. задана структура события struct TEvent { int what;//тип события union { int command;//код комманды struct//параметры команды { int message; int a; }; }; }; Для чего используются поля a и message?**

Поля a и message используются для передачи параметров команды, которая вызвала событие. Значения этих полей также могут быть определены разработчиком программы в зависимости от конкретной задачи и параметров команд, которые необходимо передать. Например, для команды "открыть файл" параметрами могут быть путь к файлу и тип файла. Значения полей a и message присваиваются в момент создания объекта события и содержат дополнительную информацию о событии, которую можно использовать при его обработке.

**14. Какие методы необходимы для организации обработки сообщений?**

Для организации обработки сообщений необходимы методы, которые позволяют получить информацию о типе события и его параметрах, а также методы для обработки каждого типа событий. Например, для структуры TEvent можно определить методы:

Пример кода на C++:

class EventHandler {

public:

virtual void handleMouseEvent(MouseEventType event) = 0;

virtual void handleKeyDownEvent(KeyDownEvent event) = 0;

virtual void handleMessageEvent(MessageEvent event) = 0;

};

class EventProcessor {

public:

void processEvent(TEvent event) {

switch (event.what) {

case MOUSE\_EVENT:

eventHandler->handleMouseEvent(event.mouse);

break;

case KEY\_DOWN\_EVENT:

eventHandler->handleKeyDownEvent(event.keyDown);

break;

case MESSAGE\_EVENT:

eventHandler->handleMessageEvent(event.message);

break;

default:

// Неизвестный тип события

break;

}

}

private:

EventHandler\* eventHandler;

};

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

Главный цикл обработки событий-сообщений имеет следующую структуру:

Пример кода на C++:

EventProcessor eventProcessor;

TEvent event;

while (true) {

// Ожидание события

waitEvent(&event);

// Обработка события

eventProcessor.processEvent(event);

}

В этом цикле программа постоянно ожидает поступления новых событий и передает их на обработку объекту класса EventProcessor. Функция waitEvent может использоваться для ожидания поступления новых событий.

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

Метод ClearEvent() очищает содержимое объекта TEvent, устанавливая все его поля в нулевые значения. Он выполняется путем присваивания значения 0 полю what и всем полям внутри объединения.

Пример кода на C++:

void TEvent::ClearEvent() {

what = 0;

memset(&mouse, 0, sizeof(mouse));

memset(&keyDown, 0, sizeof(keyDown));

memset(&message, 0, sizeof(message));

}

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

Метод HandleEvent() обрабатывает событие, переданное в качестве аргумента. Он проверяет тип события и вызывает соответствующие обработчики, зарегистрированные для этого типа событий.

Пример кода на C++:

void TApplication::HandleEvent(TEvent& event) {

switch (event.what) {

case evMouseDown:

MouseDown(event.mouse);

break;

case evKeyDown:

KeyDown(event.keyDown);

break;

case evMessage:

Message(event.message.a, event.message.message);

break;

default:

break;

}

}

**18. Какую функцию выполняет метод GetEvent()?**

Метод GetEvent() получает следующее событие из очереди событий и возвращает его. Если очередь пуста, метод блокируется до тех пор, пока не будет получено новое событие.

Пример кода на C++:

void TApplication::GetEvent(TEvent& event) {

while (queue.empty()) {

WaitForEvent();

}

event = queue.front();

queue.pop();

}

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

Поле EndState используется для хранения кода завершения приложения, который будет возвращен операционной системе при завершении работы приложения. Класс TProgram содержит это поле.

Пример кода на C++:

class TProgram {

public:

int EndState;

void Execute() {

Init();

Run();

Done();

}

};

**20. Для чего используется функция Valid()?**

Функция Valid() используется для проверки корректности состояния объекта. Она возвращает true, если объект находится в допустимом состоянии, и false в противном случае.

Пример кода на C++:

bool TRect::Valid() const {

return left <= right && top <= bottom;

}

**Диаграмма класса**

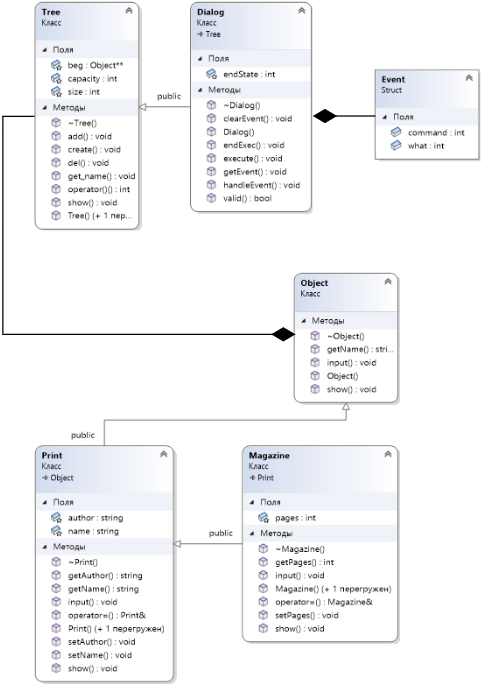


Рисунок 1 – диаграмма классов Object, Print, Magazine, Tree, Dialog и Event

**Описание классов**

**Класс object**

#pragma once

#include <iostream>

#include <string>

#include <conio.h>

using namespace std;

class Object

{

public:

Object() {};

virtual ~Object() {};

virtual void show() = 0;

virtual void input() = 0;

virtual string getName() = 0;

};

**Класс Print**

#pragma once

#include "Object.h"

class Print : public Object

{

protected:

string name;

string author;

public:

Print();

Print(const Print& object);

~Print();

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

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

string getName() override { return name; };

string getAuthor() { return author; };

Print& operator=(const Print& object);

void show() override;

void input() override;

};

**Класс Magazine**

#pragma once

#include "Print.h"

class Magazine : public Print

{

protected:

int pages;

public:

Magazine();

Magazine(const Magazine& object);

~Magazine();

void setPages(int pages) { this->pages = pages; }

int getPages() { return pages; };

Magazine& operator=(const Magazine& object);

void show() override;

void input() override;

};

**Класс Event**

#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 Event

{

int what;

int command;

};

**Класс Tree**

#pragma once

#include "Object.h"

class Tree

{

protected:

Object\*\* beg;

int capacity;

int size;

public:

Tree();

Tree(int a);

~Tree();

void create();

void add();

void del();

void show();

void get\_name();

int operator()();

};

**Класс Dialog**

#pragma once

#include "Tree.h"

#include "Event.h"

class Dialog : public Tree

{

protected:

int endState;

public:

Dialog();

~Dialog();

void execute();

void getEvent(Event& event);

void handleEvent(Event& event);

void clearEvent(Event& event);

bool valid();

void endExec();

};

**Определение методов**

**Класс Print**

#include "Print.h"

Print::Print()

{

name = "NULL";

author = "NULL";

}

Print::Print(const Print& object)

{

this->name = object.name;

this->author = object.author;

}

Print::~Print()

{

}

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

{

if (this == &object)

return \*this;

this->name = object.name;

this->author = object.author;

return \*this;

}

void Print::show()

{

cout << "Name: " << name;

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

cout << endl;

}

void Print::input()

{

cout << "Name(str): "; cin >> name;

cout << "Author(str): "; cin >> author;

}

**Класс Magazine**

#include "Magazin.h"

Magazine::Magazine() : Print()

{

pages = 0;

}

Magazine::Magazine(const Magazine& object) : Print(object)

{

this->pages = object.pages;

}

Magazine::~Magazine()

{

}

Magazine& Magazine::operator=(const Magazine& object)

{

if (this == &object)

return \*this;

this->pages = object.pages;

this->name = object.name;

this->author = object.author;

return \*this;

}

void Magazine::show()

{

cout << "Name: " << name;

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

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

cout << endl;

}

void Magazine::input()

{

cout << "Name(str): "; cin >> name;

cout << "Author(str): "; cin >> author;

cout << "Pages(int): "; cin >> pages;

}

**Класс Tree**

#include "Tree.h"

#include "Print.h"

#include "Magazin.h"

Tree::Tree()

{

beg = nullptr;

capacity = 0;

size = 0;

}

Tree::Tree(int a)

{

beg = new Object \*[a];

size = 0;

capacity = a;

}

Tree::~Tree()

{

if (beg != nullptr)

delete[] beg;

beg = nullptr;

}

void Tree::create()

{

if (beg != nullptr)

del();

cout << "\nEnter size: "; cin >> capacity;

beg = new Object \* [capacity];

size = 0;

}

void Tree::add()

{

if (size < capacity && capacity != 0)

{

unsigned int answer;

cout << "\n----- enter -----\n1 - Print\n2 - Magazine\n--> "; cin >> answer;

cout << endl;

if (answer == 1)

{

Print\* a = new Print;

a->input();

beg[size] = a;

++size;

cout << "\ndata added successfully!\n";

}

else if (answer == 2)

{

Magazine\* b = new Magazine;

b->input();

beg[size] = b;

++size;

cout << "\ndata added successfully!\n";

}

else

cout << "\n[Try again]\n";

}

else

cout << "\n[Out of range or the group was not created]\n";

}

void Tree::del()

{

if (beg != nullptr)

{

for (size\_t i = 0; i < size; i++)

delete beg[i];

delete beg;

beg = nullptr;

size = 0;

capacity = 0;

cout << "\nThe group was successfully deleted!\n";

}

else

cout << "\n[group is empty]\n";

size = 0;

}

void Tree::show()

{

if (size == 0)

cout << "\n[group is empty]\n";

else

for (size\_t i = 0; i < size; i++)

{

cout << "(" << i + 1 << ")\n";

beg[i]->show();

}

}

void Tree::get\_name()

{

if (size == 0)

cout << "\n[group is empty]\n";

else

for (size\_t i = 0; i < size; i++)

cout << "(" << i + 1 << ")\n" << beg[i]->getName() << endl;

}

int Tree::operator()()

{

return size;

}

**Класс Dialog**

#include "Dialog.h"

void pause()

{

cout << "\npress any key to continue . . . ";

while (true)

if (\_kbhit())

{

\_getch();

break;

}

system("cls");

}

Dialog::Dialog()

{

endState = 0;

}

Dialog::~Dialog()

{

if (beg != nullptr)

{

for (size\_t i = 0; i < size; i++)

delete beg[i];

delete beg;

}

}

void Dialog::execute()

{

Event event;

do

{

system("cls");

cout << "m: Create a group\n";

cout << "+: Add element \n";

cout << "-: Delete group\n";

cout << "s: Group data\n";

cout << "z: Name\n";

cout << "q: End of work\n";

endState = 0;

getEvent(event);

handleEvent(event);

pause();

} while (valid());

}

void Dialog::getEvent(Event& event)

{

string operation = "m+-szq";

char choice;

cout << "--> "; cin >> choice;

if (operation.find(choice) != -1)

{

event.what = evMessage;

switch (choice)

{

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

cout << "\n[Input operation is invalid]\n";

}

void Dialog::handleEvent(Event& event)

{

if (event.what == evMessage)

{

switch (event.command)

{

case cmMake:

create();

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;

}

}

}

void Dialog::clearEvent(Event& event)

{

event.what = evNothing;

}

bool Dialog::valid()

{

return !endState;

}

void Dialog::endExec()

{

endState = 1;

}

**Результаты работы программы**

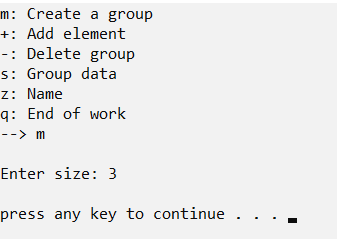


Рисунок 2 – создание группы на 3 элемента

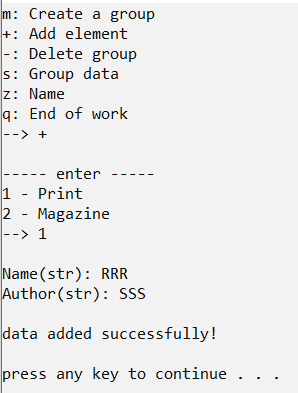


Рисунок 3 – добавление группы Print

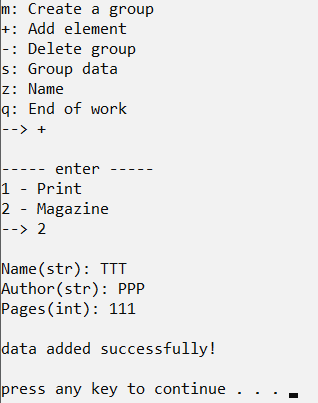


Рисунок 4 – добавление группы Magazine

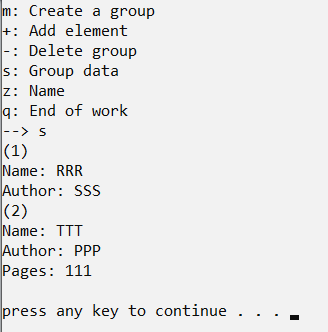


Рисунок 5 – вывод данных о группах

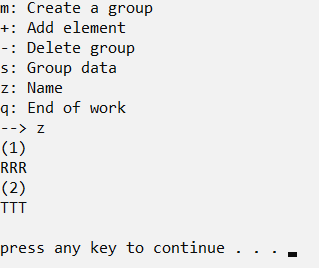


Рисунок 6 – вывод имен групп

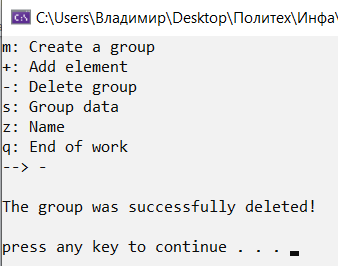


Рисунок 7 – удаление групп

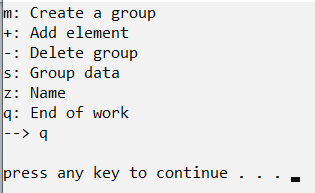


Рисунок 8 – конец работы программы