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

ПНИПУ

**Лабораторная работа №8**

**«Программа, управляемая событиями»**

**14 вариант**

Выполнил:

Студент группы РИС-23-1б

Шароглазов Егор Алексеевич

Проверила:

Доцент кафедры ИТАС

О.А. Полякова

1. г

**Постановка задачи:**

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.

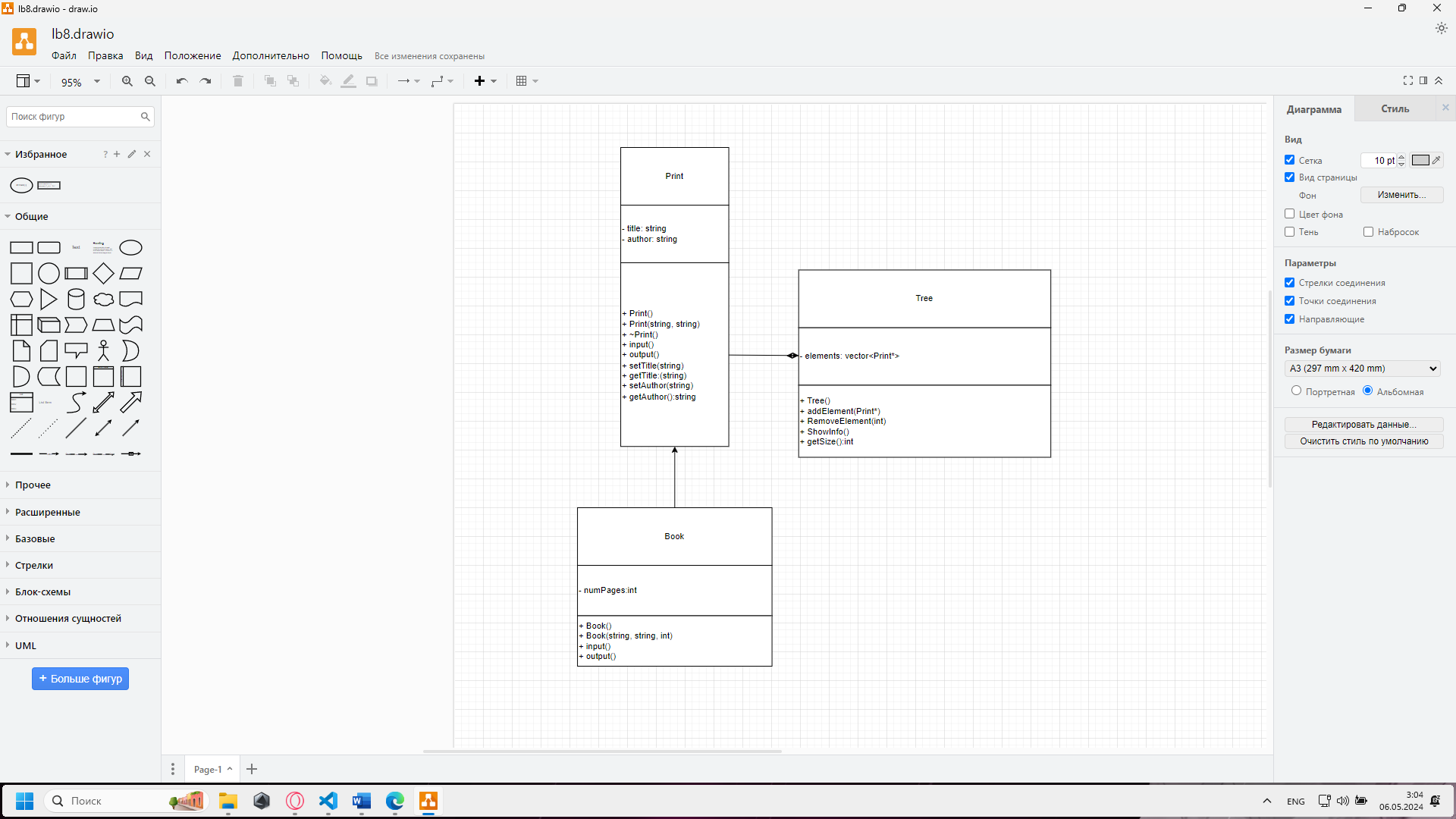
7. Написать тестирующую программу.

8. Нарисовать диаграмму классов и диаграмму объектов.

**Анализ задачи:**

 Класс Print:  
Абстрактный базовый класс для печатных изданий.  
Имеет защищенные атрибуты title (название) и author (автор) типа string.  
Определены конструкторы (по умолчанию и с параметрами), виртуальный деструктор, чисто виртуальные методы input() и output() для ввода и вывода информации.  
Имеет методы setTitle, getTitle, setAuthor, getAuthor для установки и получения значений атрибутов.  
3. Класс Book:  
Дочерний класс Print, представляющий книгу.  
Имеет приватный атрибут numPages (количество страниц) типа int.  
Определены конструкторы (по умолчанию и с параметрами).  
Переопределены методы input() и output() для ввода и вывода информации о книге(включая количество страниц).  
4. Класс Tree:  
Класс для хранения группы элементов Print\*.  
Имеет приватный атрибут elements типа vector<Print\*>, представляющий собой динамический массив указателей на объекты Print.  
Определены конструктор, методы addElement (добавление элемента в группу), removeElement (удаление элемента по индексу), showInfo (вывод информации о всех элементах группы) и getSize (возвращает размер группы).  
Объявлен как friend в main для доступа к приватным членам.  
5. Функция main():  
Создается объект group типа Tree.  
Пользователь вводит количество элементов, и создаются объекты Book, которые добавляются в group.  
Цикл с меню позволяет пользователю добавлять, удалять элементы, просматривать информацию о группе или выходить из программы.  
В зависимости от выбора пользователя вызываются соответствующие методы класса Tree.

**UML – диаграмма**



**Код**

#include <iostream>

#include <string>

#include <vector>

#include <windows.h>

#include <stdio.h>

using namespace std;

class Print {

protected:

    string title;

    string author;

public:

    Print() : title(""), author("") {}

    Print(string t, string a) : title(t), author(a) {}

    virtual ~Print() {}

    virtual void input() = 0;

    virtual void output() = 0;

    void setTitle(string t) {

        title = t;

    }

    string getTitle() const {

        return title;

    }

    void setAuthor(string a) {

        author = a;

    }

    string getAuthor() const {

        return author;

    }

};

class Book : public Print {

private:

    int numPages;

public:

    Book() : Print(), numPages(0) {}

    Book(string t, string a, int np) : Print(t, a), numPages(np) {}

    void input() override {

        cout << "Введите название: ";

        cin >> title;

        cout << "Укажите автора: ";

        cin >> author;

        cout << "Введите количество страниц: ";

        cin >> numPages;

    }

    void output() override {

        cout << "Название: " << title << endl;

        cout << "Автор: " << author << endl;

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

    }

};

class Tree {

private:

    vector<Print\*> elements;

public:

    Tree() {}

    void addElement(Print\* p) {

        elements.push\_back(p);

    }

    void removeElement(int index) {

        if (index >= 0 && index < elements.size()) {

            delete elements[index];

            elements.erase(elements.begin() + index);

        }

    }

    void showInfo() {

        for (Print\* p : elements) {

            p->output();

        }

    }

    int getSize() const {

        return elements.size();

    }

    friend int main();

};

int main() {

    Tree group;

    char choice;

    int numElements;

    cout << "Введите количество элементов в группе: ";

    cin >> numElements;

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

        Book\* mag = new Book();

        mag->input();

        group.addElement(mag);

    }

    do {

        cout << "МЕНЮ:" << endl;

        cout << "1. Добавить элемент в группе (+)" << endl;

        cout << "2. Удалить элемент из группы (-)" << endl;

        cout << "3. Показать информацию об элементах группы" << endl;

        cout << "4. Показать названия всех элементах группы" << endl;

        cout << "5. Выход" << endl;

        cout << "Выберите действие: ";

        cin >> choice;

        switch (choice) {

        case '1': {

            Book\* mag = new Book();

            mag->input();

            group.addElement(mag);

            break;

        }

        case '2': {

            int index;

            cout << "Введите индекс элемента для удаления: ";

            cin >> index;

            group.removeElement(index);

            break;

        }

        case '3':

            group.showInfo();

            break;

        case '4':

            for (Print\* p : group.elements) {

                cout << p->getTitle() << endl;

            }

            break;

        case '5':

            break;

        default:

            cout << "Неверный выбор. Попробуйте снова" << endl;

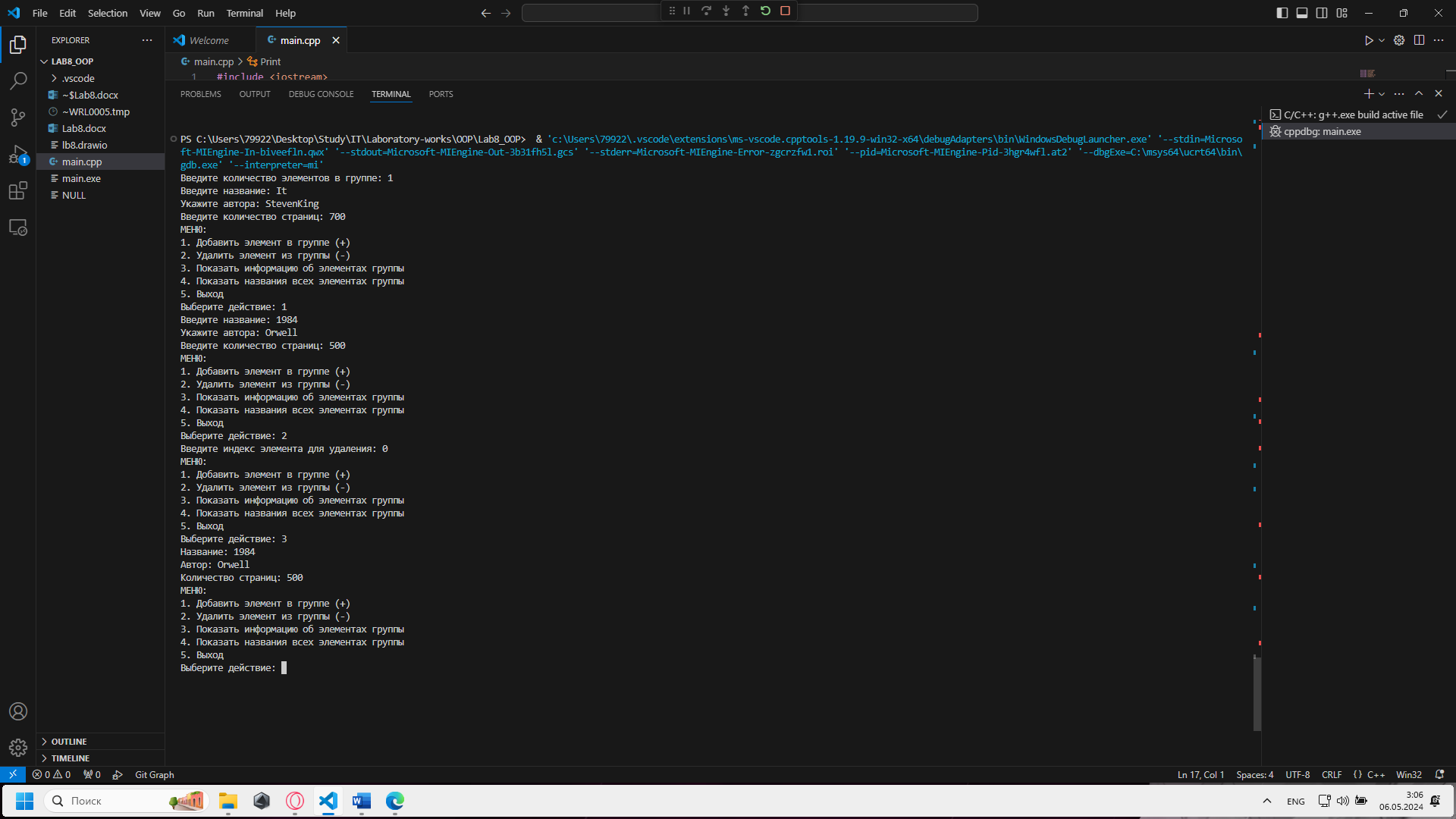
        }

    } while (choice != '5');

    return 0;

}

**Вывод**



**Вопросы**

1. Класс-группа - это абстрактный класс, который объединяет другие классы в логическую или функциональную группу. Он предоставляет общий интерфейс для работы с этой группой классов. Примеры классов-групп могут включать классы для работы с коллекциями (например, список, стек, очередь), классы для работы с графическими элементами (например, элементы управления), классы для работы с событиями и др.
2. Пример описания класса-группы "Список" может выглядеть следующим образом:

cpp

class List {

public:

virtual void add() = 0;

virtual void remove() = 0;

virtual void show() = 0;

virtual ~List() {}

};

1. Примеры конструкторов для класса-группы "Список":
   * Конструктор без параметров:

cpp

List() {}

* Конструктор с параметром:

cpp

List(int size) {}

* Конструктор копирования:

cpp

1. List(const List& other) {}
2. Пример деструктора для класса-группы "Список":

cpp

virtual ~List() {}

1. Пример метода для просмотра элементов для класса-группы "Список":

cpp

virtual void show() = 0;

1. Группа классов образует иерархию, которая позволяет структурировать и организовать код программы. Она обеспечивает удобное разделение функциональности на логически связанные компоненты.
2. Во главе иерархии классов, содержащихся в группе объектов, должен находиться абстрактный класс, так как он определяет общий интерфейс для всех классов этой группы. Это позволяет использовать полиморфизм для работы с объектами разных конкретных классов через общий интерфейс.
3. Событие представляет собой сообщение о возникновении какого-либо события или действия в программе. События используются для управления потоком выполнения программы, обработки пользовательских действий и реакции на внешние воздействия.
4. Характеристики события-сообщения:
   * Тип события (например, нажатие клавиши, перемещение мыши, команда от пользователя).
   * Параметры события (дополнительные данные, например, координаты мыши, код клавиши, дополнительные параметры команды).
   * Код события (определяет конкретное действие или команду).
5. Пример структуры, описывающей событие:

cpp

struct Event {

int type; // Тип события

union {

MouseEvent mouse; // Параметры мыши

KeyDownEvent keyDown; // Параметры нажатия клавиши

MessageEvent message; // Параметры сообщения

};

};

1. В структуре события TEvent полю what присваиваются значения типа события. Например:

* what = 0 может означать общее сообщение или отсутствие события.
* what = 1 может означать сообщение о нажатии клавиши.
* what = 2 может означать сообщение о перемещении мыши.
* what = 3 может означать сообщение пользовательской команды.

1. В структуре события TEvent полю command присваиваются значения кодов команд. Например:

* command = 1 может означать добавление элемента.
* command = 2 может означать удаление элемента.
* command = 3 может означать выполнение определенной команды.

1. Поля a и message в структуре события TEvent используются для передачи дополнительных параметров команды. Например, a может содержать дополнительный аргумент или индекс, а message может содержать текстовое сообщение или дополнительные данные.
2. Для организации обработки сообщений необходимы методы для обработки каждого типа событий, например, методы для обработки нажатия клавиши, перемещения мыши, пользовательских команд и т. д.
3. Главный цикл обработки событий-сообщений часто является циклом, в котором происходит ожидание событий и их обработка. Он отвечает за получение событий из очереди событий, их анализ и вызов соответствующих обработчиков событий.
4. Метод ClearEvent() выполняет очистку поля события, устанавливая его тип в значение, обозначающее отсутствие события (например, evnothing).
5. Метод HandleEvent() выполняет обработку полученного события, выполняя соответствующие действия в зависимости от типа события и его параметров.
6. Метод GetEvent() используется для получения нового события из источника событий (например, пользовательского ввода, системных сообщений).
7. Поле EndState используется для определения состояния выполнения программы (например, завершение работы программы). Оно содержится в классе Dialog.
8. Функция Valid() используется для проверки текущего состояния программы или события. Например, она может возвращать true, если программа находится в активном состоянии, и false, если программа завершилась или находится в состоянии ожидания.