Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП**

**«№8»**

Выполнил:

студент группы РИС-23-1б

Кривошеин Александр Антонович

Проверила:

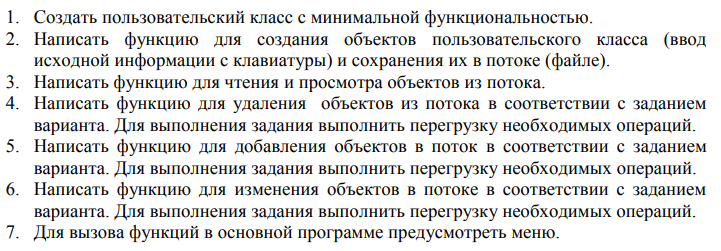
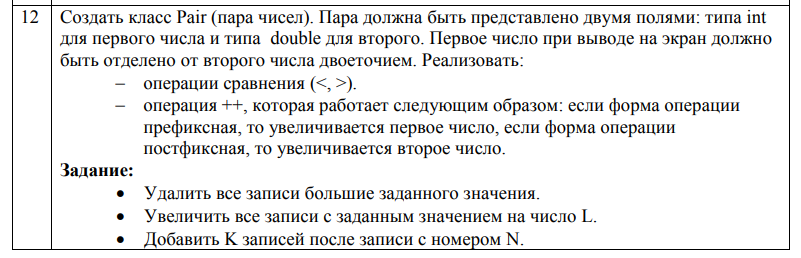
доцент кафедры ИТАС

Полякова Ольга Андреевна

2024 г.

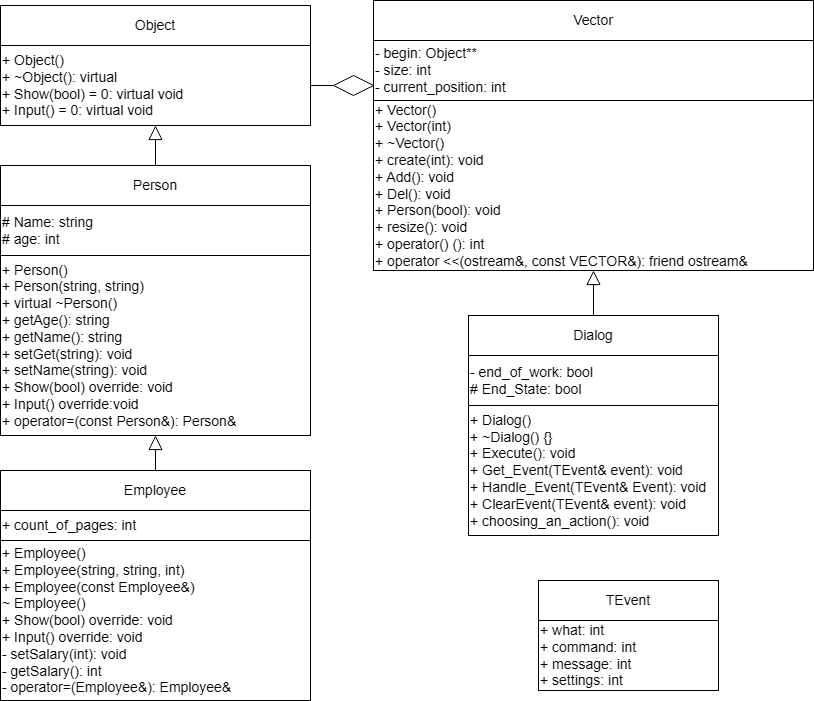
**Разработка алгоритма**

**Постановка задачи:**

****

**Анализ задачи:**

1. Необходимо определить иерархию пользовательских классов, где во главе стоит абстрактный класс Object с чисто виртуальными методами для ввода и вывода информации об атрибутах объектов.
2. Для класса Person, наследуемого от Object, необходимо реализовать конструкторы, деструктор, операцию присваивания, селекторы и модификаторы.
3. Определить класс Employee на основе структуры Person.
4. Для Employee и Person 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 Dialog – наследника группы, в котором реализовать методы для обработки событий.
6. Добавить методы для обработки событий группой и объектами пользовательских классов.



Код на C++:

#include <iostream>

using namespace std;

#include "Includes.cpp"

int main()

{

    //system("chcp 1251>null");

    Dialog dialog;

    dialog.Execute();

    return 0;

}

Classes.h:

#pragma once

#include <iostream>

#include <string>

#include "Event.h"

using namespace std;

class Object

{

public:

    Object() {}

    virtual ~Object() {}

    virtual void Show(bool) = 0;

    virtual void Input() = 0;

};

class Person : public Object

{

protected:

    string Name;

    int age;

public:

    Person();

    Person(string, int);

    Person(const Person &);

    virtual ~Person() {}

    void Show(bool) override;

    void Input() override;

    Person &operator=(const Person &);

    void setAge(int &age) { this->age = age; }

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

    int getAge() { return this->age; }

    string getName() { return this->Name; }

};

class Employee : public Person

{

protected:

    float salary;

    string post;

public:

    Employee();

    Employee(string, int, float, string);

    Employee(const Employee &);

    ~Employee() {}

    void Show(bool) override;

    void Input() override;

    Employee &operator=(const Employee &);

    void setPages(float &salary) { this->salary = salary; }

    float getPages() { return this->salary; }

};

Dialog.h:

#pragma once

#include "Vector.h"

#include "Event.h"

#include "Classes.h"

class Dialog : public Vector

{

protected:

    int EndState;

public:

    Dialog();

    ~Dialog() {}

    void GetEvent(TEvent &);

    bool is\_command(char);

    int Execute();

    void HandleEvent(TEvent &);

    void ClearEvent(TEvent &);

    int Valid();

    void EndExec();

};

Dialog.cpp:

#pragma once

#include "Dialog.h"

Dialog::Dialog() : Vector()

{

    this->EndState = 0;

}

void Dialog::GetEvent(TEvent &event)

{

    cout << "Choose action\n"

         << "m: Create grop (example: m10)\n"

         << "+: Add element in group\n"

         << "-: Delete element from group\n"

         << "s: Output information about group elements\n"

         << "z: Output information about names of group elements\n"

         << "q: End of work\n";

    cout << "-> ";

    string temp\_command, temp\_params;

    cin >> temp\_command;

    char code = temp\_command[0];

    if (is\_command(code))

    {

        event.what = EvMessage;

        switch (code)

        {

        case 'm':

            event.command = CmCreateGroup;

            break;

        case '+':

            event.command = CmAddToGroup;

            break;

        case '-':

            event.command = CmRemoveFromGroup;

            break;

        case 's':

            event.command = CmShow;

            break;

        case 'z':

            event.command = CmShowName;

            break;

        case 'q':

            event.command = CmExit;

            break;

        }

        if (temp\_command.size() > 1)

        {

            temp\_params = temp\_command.substr(1, temp\_command.size() - 1);

            int settings = atoi(temp\_params.c\_str());

            event.settings = settings;

        }

    }

    else

    {

        event.what = EvNothing;

    }

}

bool Dialog::is\_command(char action)

{

    int i = 0;

    string temp\_command = "m+-szq";

    while (i < int(temp\_command.size()))

    {

        if (action == temp\_command[i])

        {

            return true;

        }

        i++;

    }

    return false;

}

int Dialog::Execute()

{

    TEvent event;

    do

    {

        EndState = 0;

        GetEvent(event);

        cout << "\n";

        HandleEvent(event);

    } while (!Valid());

    return EndState;

}

void Dialog::HandleEvent(TEvent &event)

{

    if (event.what == EvNothing)

    {

        cout << "Invalid command.\n";

        return;

    }

    switch (event.command)

    {

    case CmCreateGroup:

        create(event.settings);

        ClearEvent(event);

        break;

    case CmAddToGroup:

        Add();

        ClearEvent(event);

        break;

    case CmRemoveFromGroup:

        Del();

        ClearEvent(event);

        break;

    case CmShow:

        print(false);

        ClearEvent(event);

        break;

    case CmShowName:

        print(true);

        ClearEvent(event);

        break;

    case CmExit:

        EndExec();

        ClearEvent(event);

        break;

    default:

        break;

    }

}

void Dialog::ClearEvent(TEvent &event)

{

    event.what = EvNothing;

}

int Dialog::Valid()

{

    if (EndState == 0)

    {

        return 0;

    }

    return 1;

}

void Dialog::EndExec()

{

    EndState = 1;

}

Event.h:

#pragma once

const int EvNothing = 0;

const int EvMessage = 100;

const int CmCreateGroup = 1;

const int CmAddToGroup = 2;

const int CmRemoveFromGroup = 3;

const int CmShow = 4;

const int CmShowName = 5;

const int CmExit = 6;

struct TEvent

{

    int what;

    union

    {

        int command;

        struct

        {

            int message;

            int settings;

        };

    };

};

Employee.cpp:

#pragma once

#include "Classes.h"

Employee::Employee() : Person()

{

    this->salary = 0;

    this->post = "";

}

Employee::Employee(string name, int age, float salary, string post) : Person(name, age)

{

    this->age = age;

    this->post = post;

}

Employee::Employee(const Employee &other\_employee)

{

    this->age = other\_employee.age;

    this->Name = other\_employee.Name;

    this->salary = other\_employee.salary;

}

void Employee::Show(bool names = false)

{

    if (!names)

    {

        cout << "\nName: " << this->Name << " Age: " << this->age

             << "\nSalary: " << this->salary << " Post: " << this->post << "\n\n";

    }

    else

    {

        cout << "\nName: " << this->Name << "\n\n";

    }

}

void Employee::Input()

{

    cout << "\nEnter data of worker:\n";

    cout << "Name: ";

    cin.seekg(cin.eof());

    getline(cin, this->Name);

    cout << "Age: ";

    cin >> this->age;

    cout << "Salary: ";

    cin >> this->salary;

    cout << "Post: ";

    cin.seekg(cin.eof());

    getline(cin, this->post);

}

Employee &Employee::operator=(const Employee &other\_employee)

{

    if (this == &other\_employee)

    {

        return \*this;

    }

    Person::operator=(other\_employee);

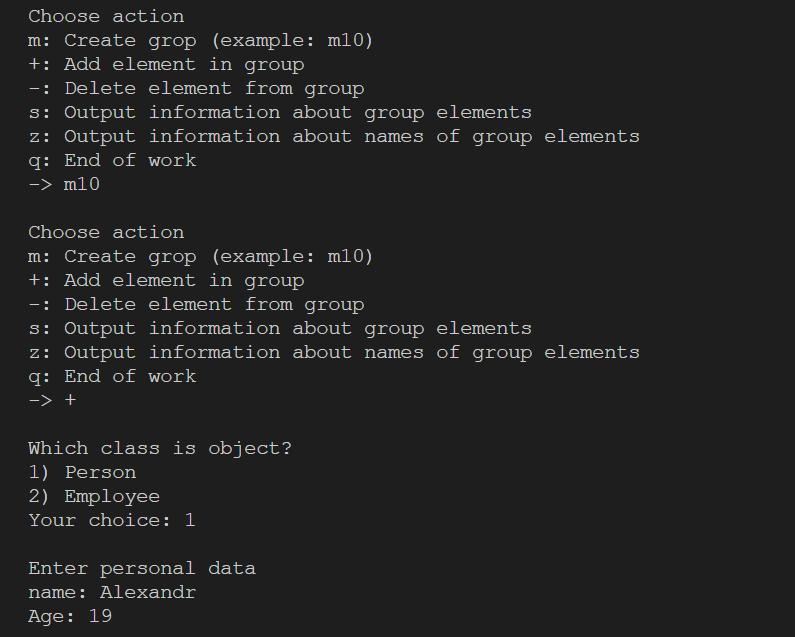
    this->salary = other\_employee.salary;

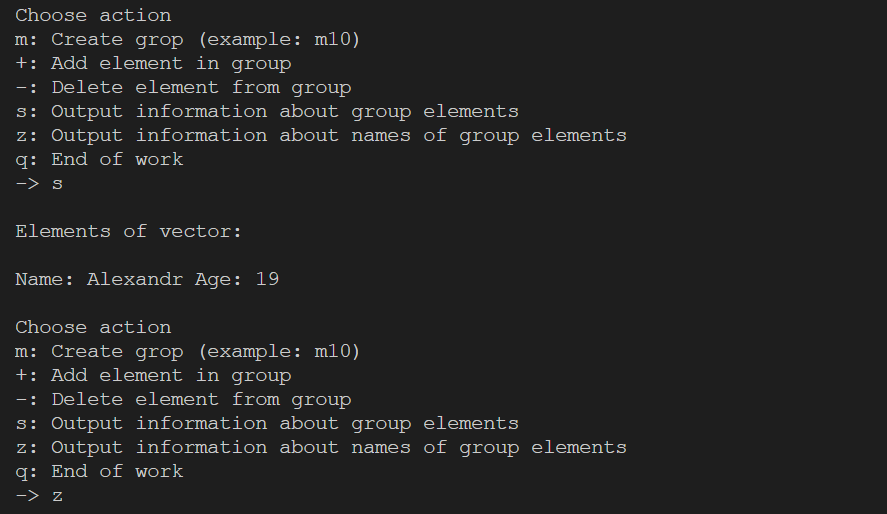
    this->post = other\_employee.post;

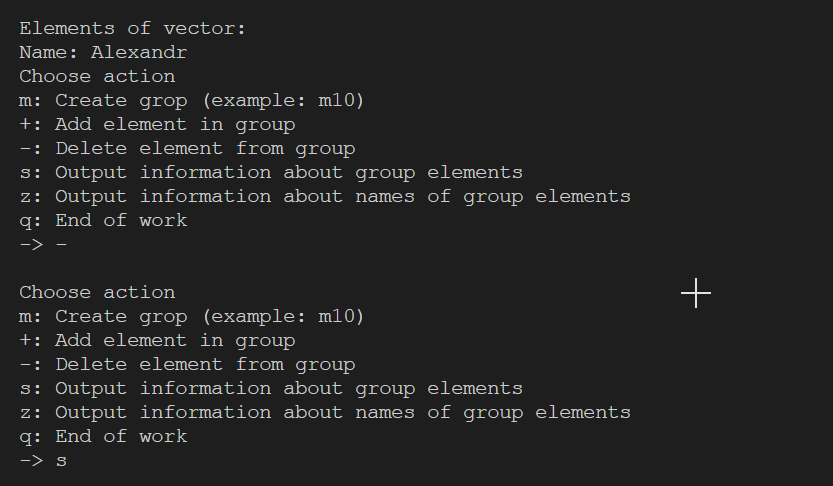
    return \*this;

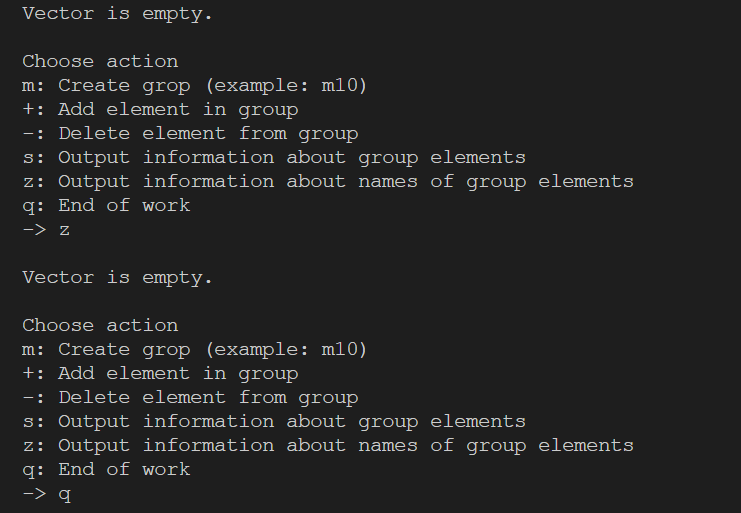
}

**Пример работы:**







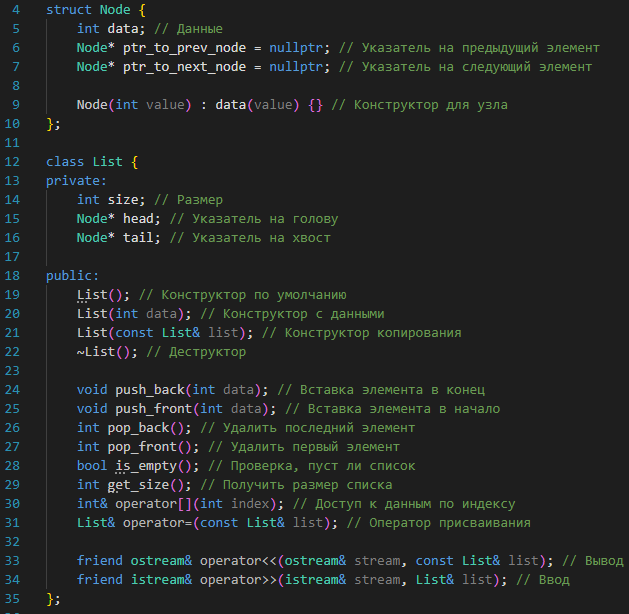


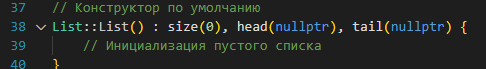
**Контрольные вопросы**

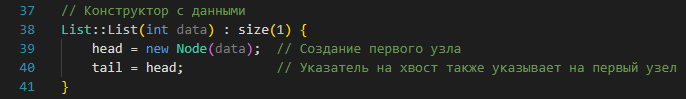
1. **Что такое класс-группа? Привести примеры таких классов.**

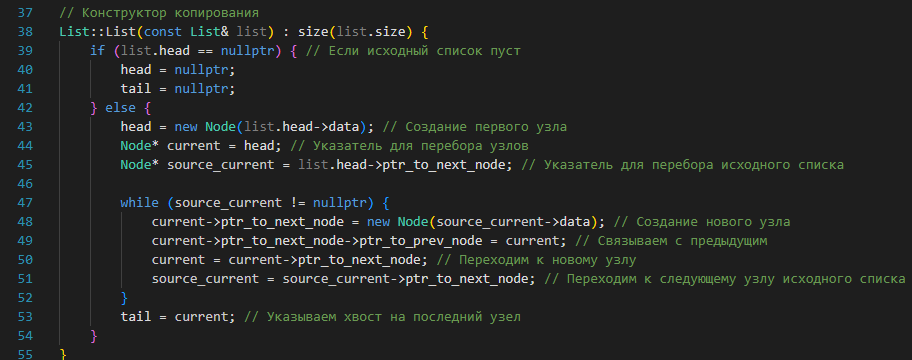
Группа – это объект, в который включены другие объекты. Объекты, входящие в группу, называются элементами группы. Элементы группы, в свою очередь, могут быть группой. Например, можно создать класс-группу "Список студентов", который будет содержать объекты класса "Студент". Класс-группа будет предоставлять методы для добавления/удаления студентов, получения списка студентов и т.д.

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

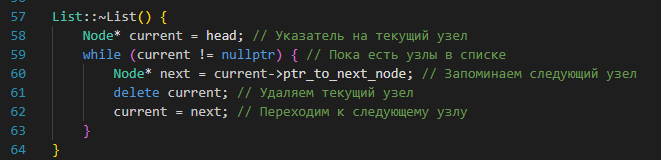
**3. Привести пример конструктора (с параметром, без параметров, копирования) для класса-группы Список.**



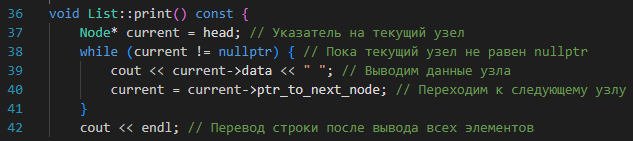




1. **Привести пример деструктора для класса-группы Список.**



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



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?**

Абстрактный класс должен находиться во главе иерархии классов, содержащихся в группе объектов, чтобы обеспечить единообразие интерфейса

* полиморфизм. Это позволяет объектам производных классов использоваться взаимозаменяемо, что упрощает разработку и поддержку кода.

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

Событие — это сигнал о том, что произошло некоторое действие или

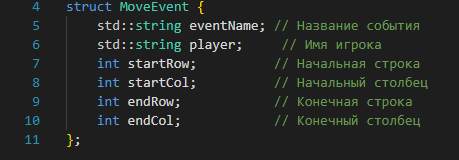
изменение состояния. События используются для реализации интерактивности в приложениях, позволяя компонентам реагировать на действия пользователей или другие внешние факторы.

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

Объект TEvent состоит из двух частей. Первая (what) задает тип события, опреде-ляющий 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. По-лю what присваивается значение evNothing, когда событие обработано каким-либо объектом.
* evMessage - событие-сообщение от объекта.

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



1. **Задана структура события**

**struct TEvent {**

**int what;**

**union {**

**MouseEventType mouse;**

**KeyDownEvent keyDown;**

**MessageEvent message;**

**}**

**};**

**Какие значения, и в каких случаях присваиваются полю what?**

В структуре события TEvent, поле what определяет тип события. В зависимости от значения what, соответствующая часть union будет использована для хранения параметров события. Например, если what равно MouseEventType, то параметры будут храниться в mouse.

1. **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?**

В структуре события TEvent, поле command является кодом команды. В зависимости от значения command, соответствующие параметры команды будут использованы. Поле a и message являются параметрами команды.

1. **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?**

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

1. **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».

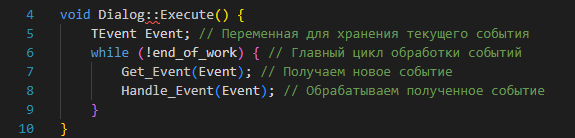
* HandleEvent – обработчик событий. Обрабатывает каждое событие нужным для него образом. Если объект должен обрабатывать определенное событие (сообщение), то его метод HandleEvent должен распознавать это событие и реагировать на него должным образом. Событие может рас-

познаваться, 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).

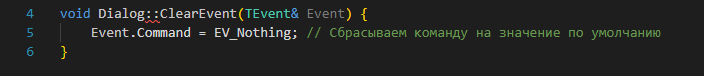
1. **Какой вид имеет главный цикл обработки событий-**

**сообщений?**



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

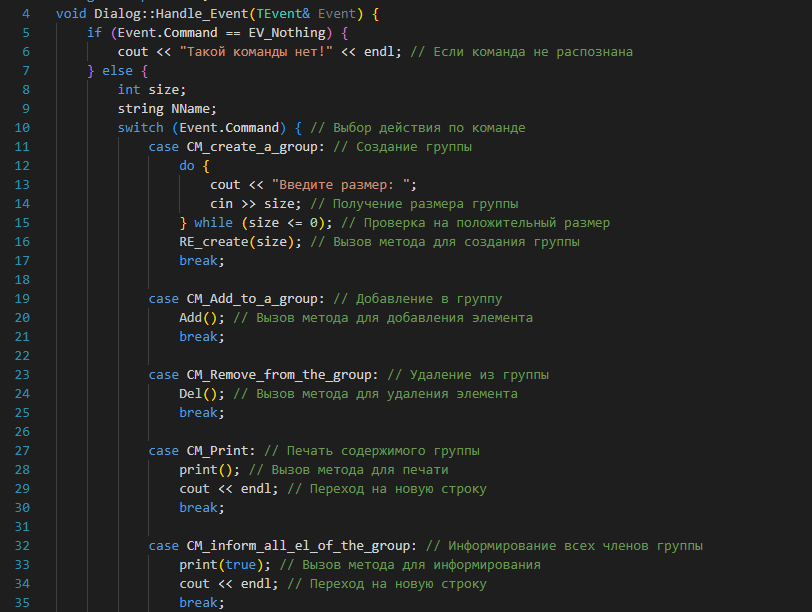
**образом?**

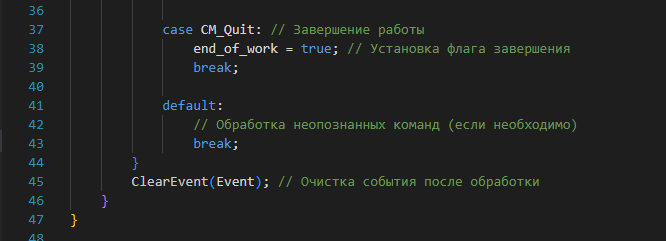


Метод ClearEvent() очищает буфер событий, удаляя все сообщения.

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

**образом?**





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

Метод GetEvent() используется для получения информации о следующем со-бытии 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.

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

**содержит это поле?**

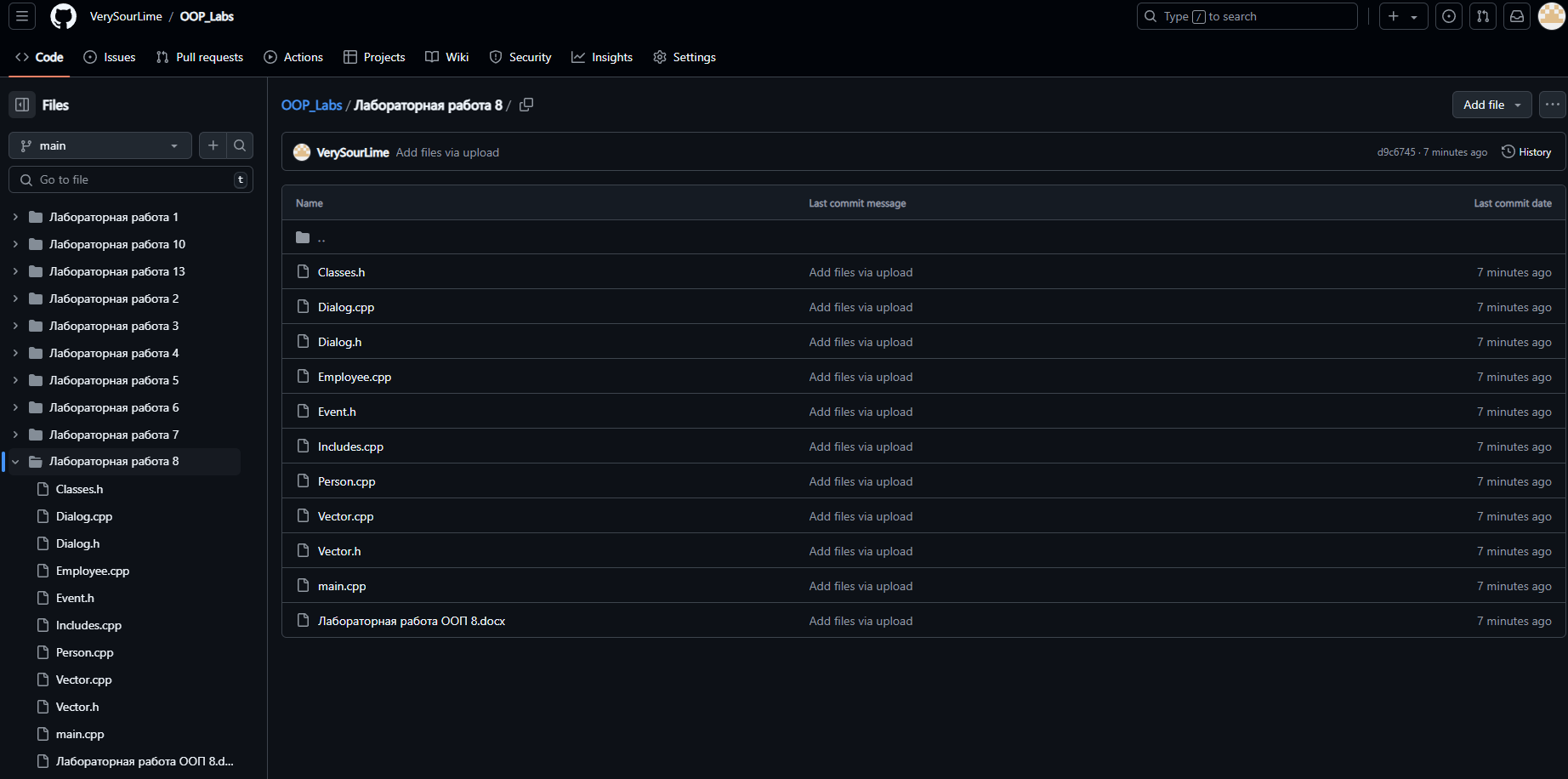
Поле EndState используется для указания конечного состояния системы после выполнения события. Обычно это поле заполняется в обработчике события, который анализирует информацию, полученную из события, и принимает решение о том, как изменить состояние системы.

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

1. **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.

Скриншот из GitHub:



GitHub: <https://github.com/VerySourLime/OOP_Labs>