Программа, управляемая событиями

Событие в ООП

- Событие в ООП – это то, что может случится с каким-либо объектом при определенных условиях (например, с кнопкой при клике на неё мышкой). При взаимодействии других объектов с этим событием (клик мыши), данный объект будет знать, что клик мыши произошел.
- При выполнении события вызывается обработчик подписанного на это событие объекта.
- Событие с точки зрения языка C++ – это объект, отдельные поля которого характеризуют те или иные свойства передаваемой информации.

Объектно-ориентированная программа

- Объектно-ориентированная программа – это программа, управляемая событиями.
- 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.

Объединения

Объединения — это структура данных, члены которой расположены по одному и тому же адресу. Поэтому размер объединения равен размеру его наибольшего члена. В любой момент времени объединение хранит значение только одного из членов.


```
union MyUnion
{
    char a; // 1 байт
    short int b; // 2 байта
    int c; // 4 байта
};
......
    MyUnion X;
    X.a = 'b';
    X.b = 3;
    X.c = 40000;
......
```

Анонимные объединения

Анонимные объединения — это те же объединения, но не имеющие ни названия, ни порожденных от себя объектов.

```
struct MyStruct
{
    union
    {
        int x;
        double y;
    };
};
```

Пример структуры события

Событие с точки зрения языка C++ – это объект, отдельные поля которого характеризуют те или иные свойства передаваемой информации.

- В данном примере, объект Event состоит из двух частей. Первая (what) задает тип события, определяющий источник данного события.
- Вторая задает информацию, передаваемую событием. Для разных типов событий содержание информации различно.

```
struct Event {
    int what; //тип события
    union
    {
        MouseEventType mouse;
        KeyDownEvent keyDown;
        MessageEvent message;
    };
};
```

Пример структуры события от мыши

- evMouse – событие от мыши (константа, которая присваивается полю what и обозначает тип события)
- buttons указывает нажатую клавишу;
- doubleClick указывает был ли двойной щелчок;
- where указывает координаты мыши, имеет поля x и y;

```
struct MouseEventType
{
    int buttons;
    int doubleClick;
    Tpoint where;
};
```

Пример структуры события от клавиатуры

evKeyDown – событие от клавиатуры (константа, которая присваивается полю what и обозначает тип события)

```
struct KeyDownEvent
{
    int keyCode;
    char charCode;
};
```

Пример структуры события - сообщения от объекта

Для события сообщения от объекта (evMessage) задаются два параметра:

command – код команды, которую необходимо выполнить при появлении данного события;

info – передаваемая с событием (сообщением) информация.

```
template <typename T>
struct MessageEvent
{
   int command;
   T info;
};
```

Методы обработки событий

Для организации обработки событий необходимы следующие методы:

1. Execute - реализация главного цикла обработки событий. Данный метод постоянно получает событие путем вызова GetEvent и обрабатывает их с помощью HandleEvent.
2. GetEvent - метод формирования события
3. HandleEvent - метод обработки событий
4. ClearEvent - метод очистки текущего события
5. EndExec - завершение обработки событий(после вызова этого метода цикл обработки событий заканчивается)

Лабораторная работа 8. Вариант 15

1. Определить иерархию пользовательских классов. 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.

Иерархия пользовательских классов

```
class Object
{
public:
    Object();
    virtual void Show() = 0;
    virtual void Input() = 0;
    virtual string Get_name() = 0;
    ~Object();
};
Object::Object() {}
Object::~Object() {}
```

```
class Print : public Object
{
public:
    Print();
public:
    void Show();
    void Input();
    Print(string, string);
    Print(const Print&);

    string Get_name() { return name; }
    string Get_author() { return author; }

    void Set_name(string);
    void Set_author(string);
    Print& operator=(const Print&);

    ~Print();
protected:
    string name;
    string author;
};
```

```
void Print::Show()
{
    cout << "\nName: " << name;
    cout << "\nAuthor: " << author << endl;
}

void Print::Input()
{
    cout << "\nEnter name: "; cin >> name;
    cout << "\nEnter author: "; cin >> author;
    cout << endl;
}
```

Иерархия пользовательских классов

```
class Object
{
public:
    Object();
    virtual void Show() = 0;
    virtual void Input() = 0;
    virtual string Get_name() = 0;
    ~Object();
};
Object::Object() {}
Object::~Object() {}
```

```
class Magazin : public Print
{
public:
    Magazin();
public:
    Magazin(string, string, int);
    Magazin(const Magazin&);

    void Show();
    void Input();
    string Get_name() {return name;}

    int Get_pages() { return pages; }
    void Set_pages(int);

    Magazin& operator=(const Magazin&);
    ~Magazin();
protected:
    int pages;
};
```

```
void Magazin::Show()
{
    cout << "\nName: " << name;
    cout << "\nAuthor: " << author;
    cout << "\nPages: " << pages << endl;
}
void Magazin::Input()
{
    cout << "\nEnter name: "; cin >> name;
    cout << "\nEnter author: "; cin >> author;
    cout << "\nEnter pages: "; cin >> pages;
    cout << endl;
}
```

Класс-группа

```
class Vector
{
public:
    Vector();
    Vector(int);

    void Add();
    void Del();
    void Show();
    void Get_Name();
    int operator()();

    ~Vector();
protected:
    Object** beg;
    int size;
    int cur;
};
```

```
void Vector::Add()
{
    Object* p;
    cout << "1. Print" << endl << "2. Magazin" << endl;
    int x;
    cin >> x;
    if (x == 1)
    {
        Print* a = new Print;
        a->Input();
        p = a;
        if (cur < size)
        {
            beg[cur] = p;
            cur++;
        }
    }
    else
        if (x == 2)
        {
            Magazin* b = new Magazin;
            b->Input();
            p = b;
            if (cur < size)
            {
                beg[cur] = p;
                cur++;
            }
        }
}
```

```
void Vector::Get_Name()
{
    Object** p = beg;
    for (int i = 0; i < cur; i++)
    {
        (*p)->Get_name();
        p++;
    }
}
```

```
void Vector::Show()
{
    if (cur == 0) cout << "Empty" << endl;
    Object** p = beg;
    for (int i = 0; i < cur; i++)
    {
        (*p)->Show();
        p++;
    }
}
int Vector::operator()() { return cur; }
```

Класс Диалог – наследник группы, в нем реализованы методы для обработки событий

```
class Dialog : public Vector
{
public:
    Dialog();
    void GetEvent(TEvent& event);
    void Execute();
    void HandleEvent(TEvent& event);
    void ClearEvent(TEvent& event);
    bool Valid();
    void EndExec();
    ~Dialog();
protected:
    int EndState;
};
```

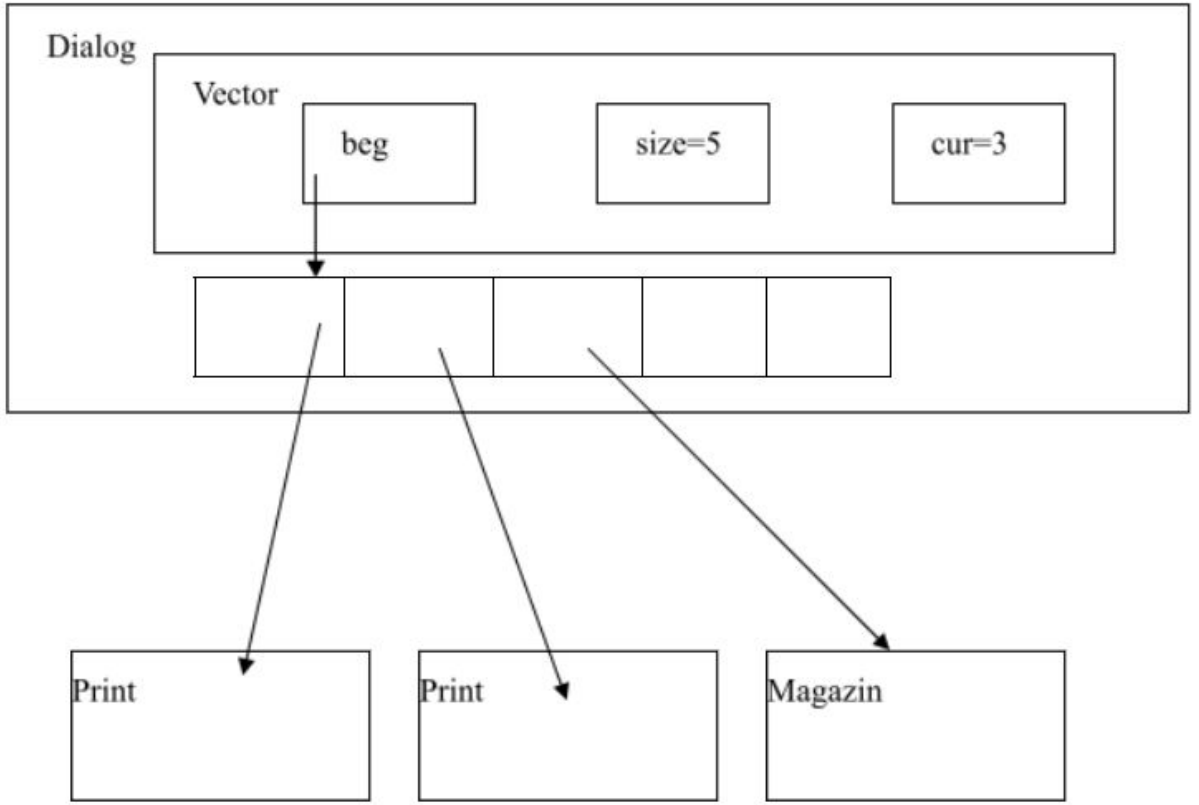
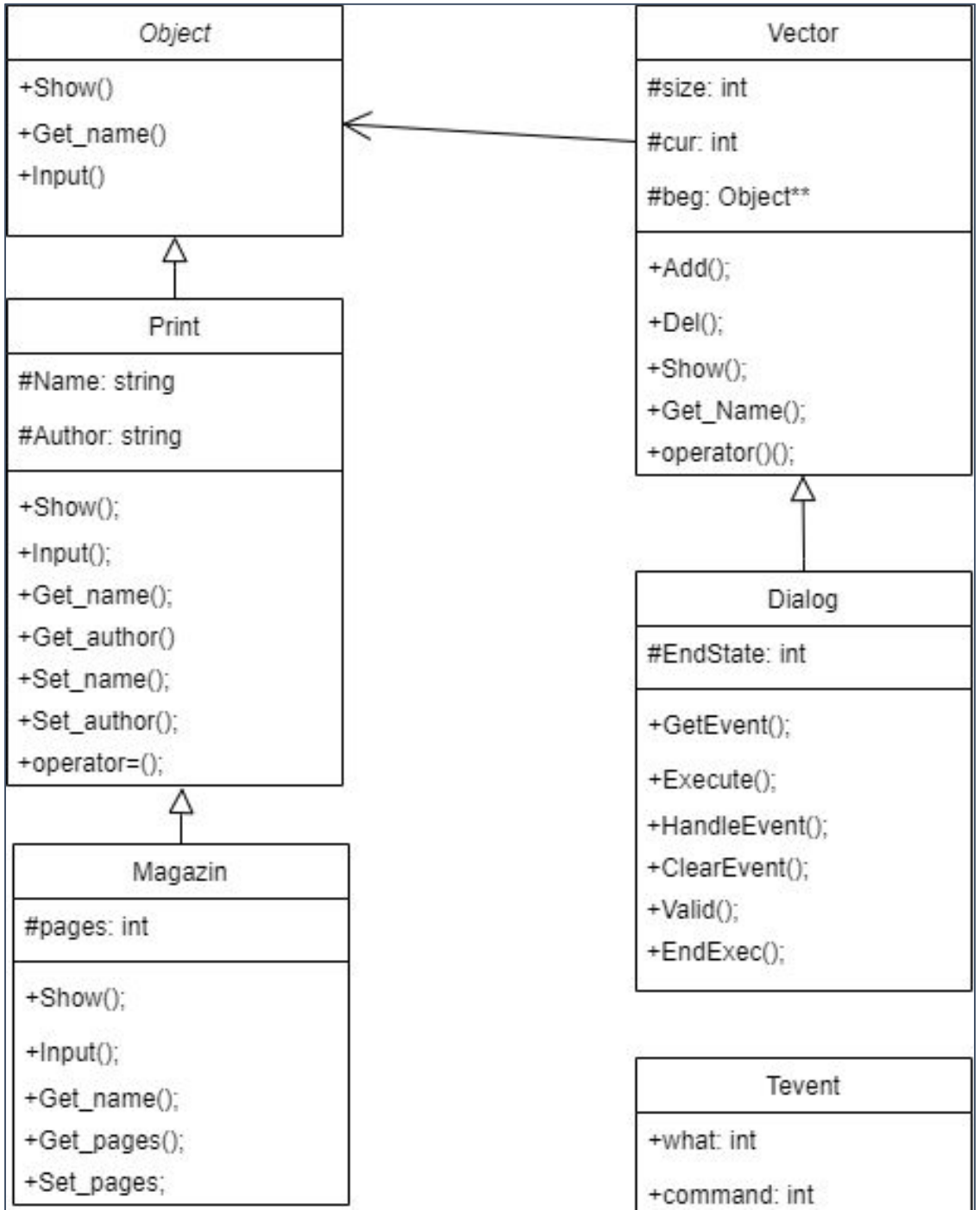


Диаграмма классов



```
void Dialog::Execute()
{
    TEvent event;
    do {
        EndState = 0;
        GetEvent(event);
        HandleEvent(event);
    } while (Valid());
}
bool Dialog::Valid()
{return EndState == 0;}
```

```
const int evNothung = 0;
const int evMessage = 100;
const int cmAdd = 1; // +
const int cmDel = 2; // -
const int cmGet = 3; // z
const int cmShow = 4; // s
const int cmMake = 6; // m
const int cmQuit = 101; // q
struct TEvent
{
    int what; // наличие события
    int command; // тип команды
};
```

```
void Dialog::HandleEvent(TEvent& event)
{
    if (event.what == evMessage)
    {
        switch (event.command)
        {
        case cmMake:
            cout << "Enter size: ";
            cin >> size;
            beg = new Object * [size];
            cur = 0;
            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::GetEvent(TEvent& event)
{
    string OpInt = "m+-szq";
    string s;
    char code;
    cout << '>';
    cin >> s; code = s[0];
    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 'z': event.command = cmGet; break;
        case 'q': event.command = cmQuit; break;
        }
    }
    else event.what = evNothung;
}
```

Методы класса Dialog

```
void Dialog::ClearEvent(TEvent& event)
{
    event.what = evNothung;
}
```

```
bool Dialog::Valid()
{
    return EndState == 0;
}
```

```
void Dialog::EndExec()
{
    EndState = 1;
}
```

Работа программы

```
int main()
{
    setlocale(LC_ALL, "rus");
    cout << "m: Создать группу\n+: Добавить элемент\n";
    cout << "-: Удалить элемент\ns: Информация о членах группы\n";
    cout << "z: Информация о названиях элементов группы\nq: Конец работы\n";
    Dialog D;
    D.Execute();
    return 0;
}
```

```
m: Создать группу
+: Добавить элемент
-: Удалить элемент
s: Информация о членах группы
z: Информация о названиях элементов группы
q: Конец работы
>m
Enter size: 3
>+
1. Print
2. Magazin
1

Enter name: Forbes

Enter author: Andrew Semenov

>+
1. Print
2. Magazin
2

Enter name: Myrzilka

Enter author: Mark Ibragimov

Enter pages: 76

>z
Forbes
Myrzilka
>-
>s

Name: Forbes
Author: Andrew Semenov
>q

Для продолжения нажмите любую клавишу . . .
```