

# Объектно-ориентированное программирование. Языки C++ и C#

§ 46. Что такое ООП?

§ 47. Объекты и классы

§ 48. Создание объектов в программе

§ 49. Скрытие внутреннего устройства

§ 50. Иерархия классов

§ 51. Программы с графическим интерфейсом

§ 52. Программирование в § 52.

Программирование в RAD-§ 52.

Программирование в RAD-средах

§ 53. Использование компонентов

§ 54. Совершенствование компонентов

§ 55. Модель и представление

# Объектно-ориентированное программирование. Языки C++ и C#

## § 46. Что такое ООП?

# Зачем нужно что-то новое?

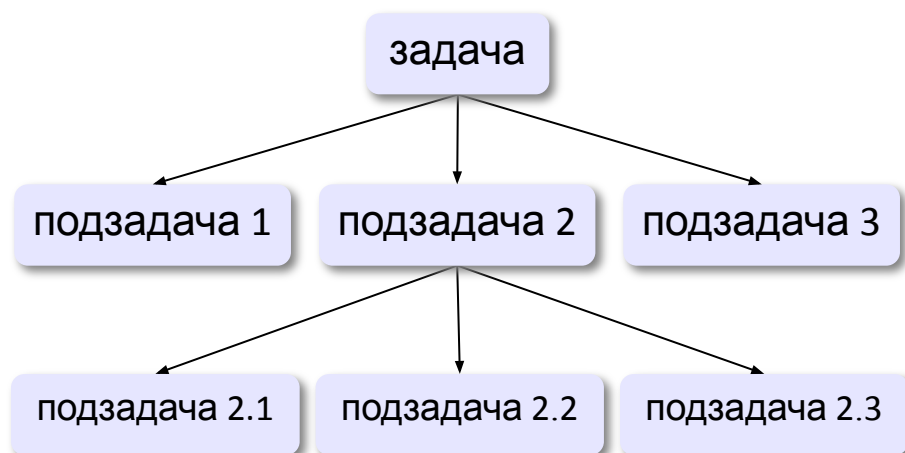


Главная проблема – **сложность!**

- программы из миллионов строк
- тысячи переменных и массивов

Э. Дейкстра: «Человечество еще в древности придумало способ управления сложными системами: **«разделяй и властвуй»**».

**Структурное программирование:**

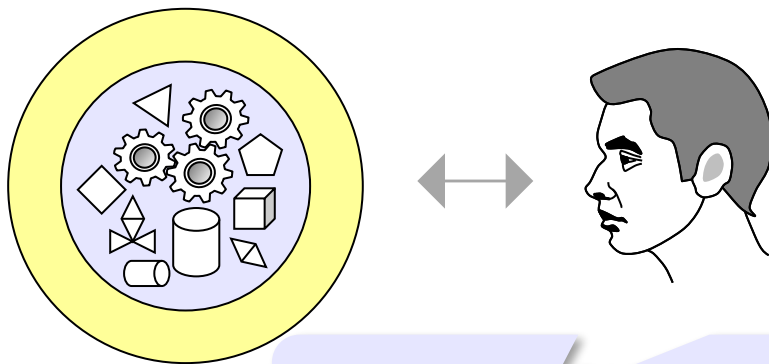


**декомпозиция по задачам**



человек мыслит иначе, объектами

# Как мы воспринимаем объекты?



существенные  
свойства

**Абстракция** – это выделение существенных свойств объекта, отличающих его от других объектов.



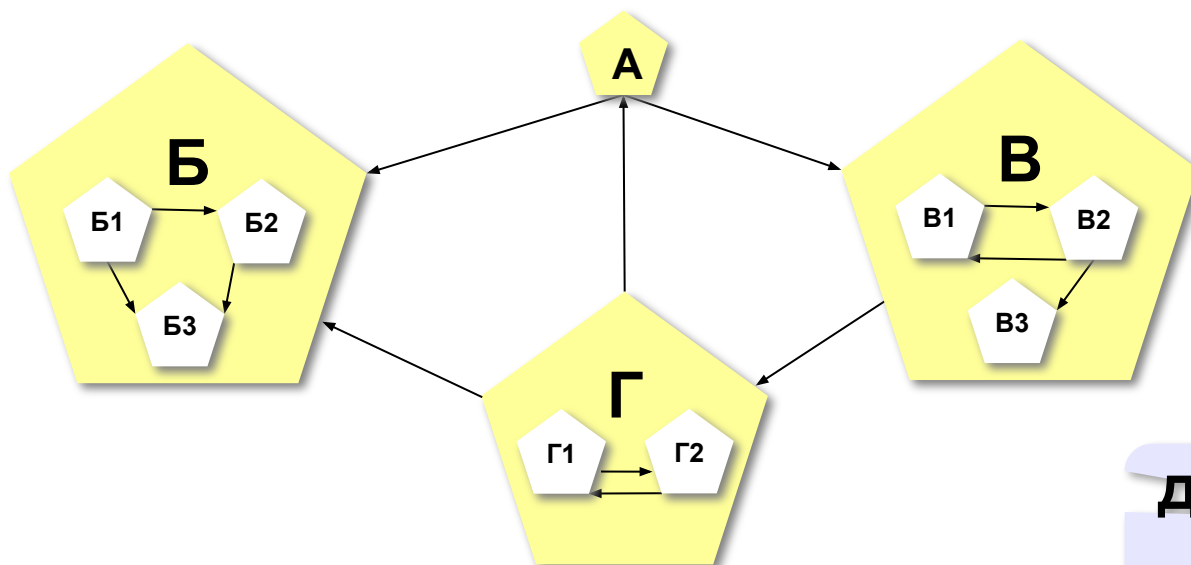
Разные цели –  
разные модели!

# Использование объектов

**Программа** – множество объектов (моделей), каждый из которых обладает своими свойствами и поведением, но его внутреннее устройство скрыто от других объектов.



Нужно «разделить» задачу на объекты!



**декомпозиция по  
объектам**

# Объектно-ориентированное программирование. Языки C++ и C#

## **§ 47. Объекты и классы**

# С чего начать?

## Объектно-ориентированный анализ (ООА):

- выделить **объекты**
- определить их существенные **свойства**
- описать **поведение** (команды, которые они могут выполнять)



Что такое объект?

**Объектом** можно назвать то, что имеет чёткие границы и обладает *состоянием* и *поведением*.

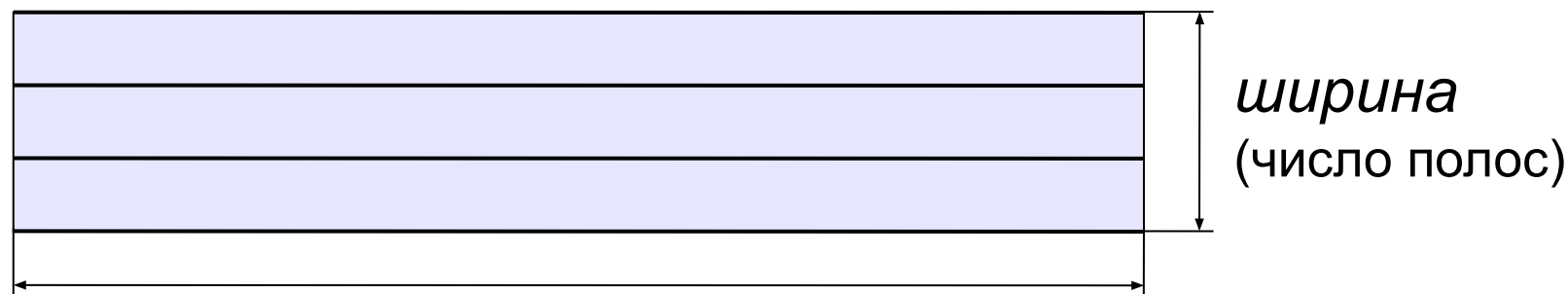
## Состояние определяет поведение:

- лежащий человек не прыгнет
- незаряженное ружье не выстрелит

**Класс** — это множество объектов, имеющих общую структуру и общее поведение.

# Модель дороги с автомобилями

## Объект «Дорога»:



название  
класса

**Дорога**

**свойства**  
(состояние)

длина  
ширина

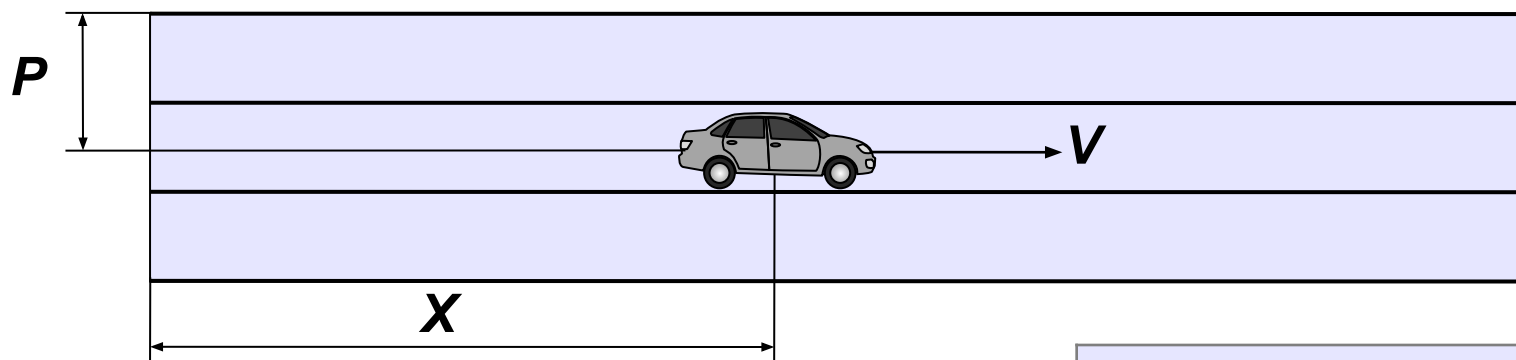
**методы**  
(поведение)



# Модель дороги с автомобилями

## Объект «Машина»:

свойства: координаты и скорость



- все машины одинаковы
- скорость постоянна
- на каждой полосе – одна машина
- если машина выходит за правую границу дороги, вместо нее слева появляется новая машина

<i><b>Машина</b></i>
$X$ (координата) $P$ (полоса) $V$ (скорость)
двигаться

**Метод** – это процедура или функция, принадлежащая классу объектов.

# Модель дороги с автомобилями

## Взаимодействие объектов:

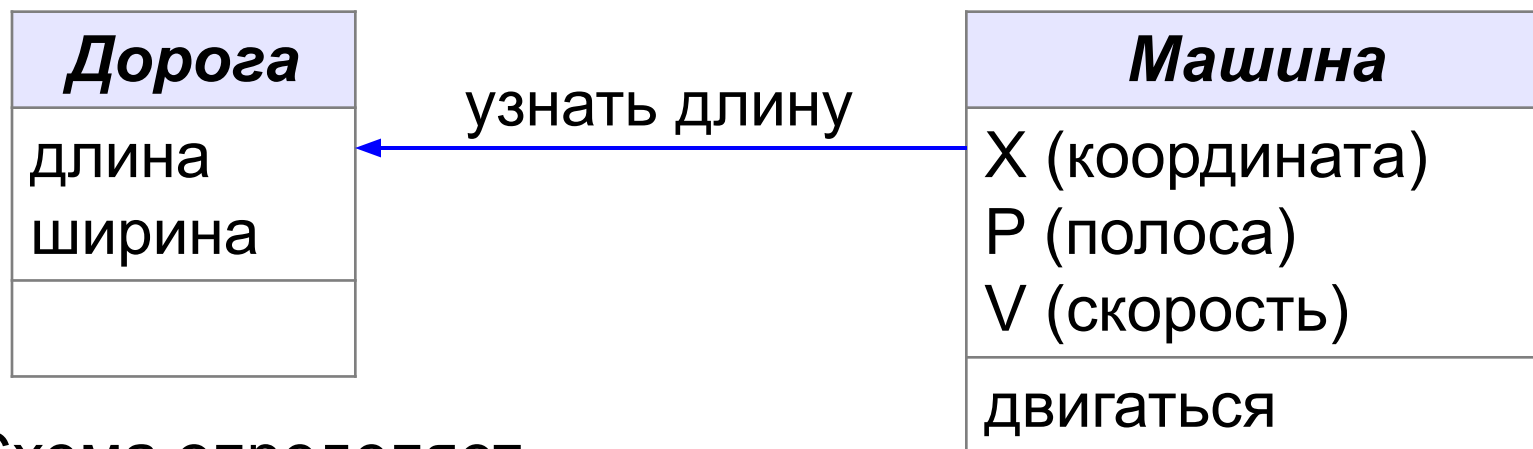


Схема определяет

- **свойства** объектов
- **методы**: операции, которые они могут выполнять
- **связи** (обмен данными) между объектами



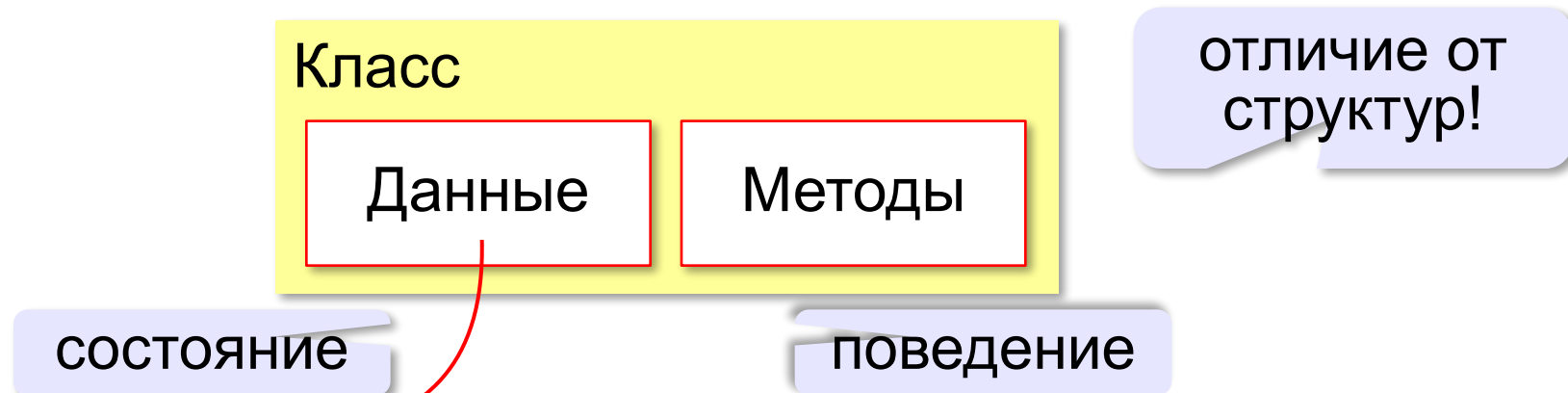
Ни слова о внутреннем устройстве объектов!

# Объектно-ориентированное программирование. Языки C++ и C#

## **§ 48. Создание объектов в программе**

# Классы

- программа – множество взаимодействующих **объектов**
- любой объект – экземпляр какого-то **класса**
- **класс** – описание группы объектов с общей структурой и поведением



**Поле** – это переменная, принадлежащая объекту.

# Класс «Дорога»

## Объявление класса:

```
class TRoad  
{  
    float Length;  
    int Width;  
};
```



Память не выделяется!

## Объявление переменной (создание объекта):

```
TRoad road;
```

## Попытка изменить данные:

```
road.Length = 60;  
road.Width = 3;
```

ошибка

private



По умолчанию все члены класса закрытые!

# Класс «Дорога»

## Объявление класса:

```
class TRoad
{
    public:
        float Length;
        int Width;
};
```



Общедоступные данные!

## Основная программа:

```
main ()
{
    TRoad road;
    road.Length = 60; // работает!
    road.Width = 3;   // работает!
}
```

# Класс «Дорога»

```
TRoad road;
```

вызов конструктора

**Конструктор** – это метод класса, который вызывается для создания объекта этого класса.

**!** Конструктор по умолчанию строится автоматически!

**?** Что записано в полях?

```
road.Length = ???  
road.Width = ???
```

«мусор»

# Новый конструктор

## Класс:

```
class TRoad
{
    public:
        float Length;
        int Width;
        TRoad(); // объявление конструктора
};
```



Имя конструктора совпадает с именем класса!

## Конструктор:

```
TRoad::TRoad()
{
    Length = 0;
    Width = 0;
}
```

МЕТОД **aaa** класса **TRoad**

```
TRoad::aaa()
{
    ...
}
```

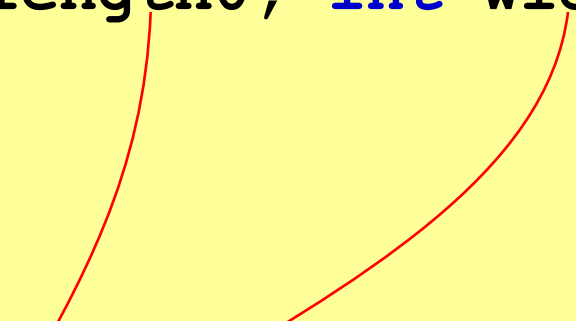


# Конструктор с параметрами

```
class TRoad
{
    public:
        ...
        TRoad ( float length0, int width0 );
};
```

## Конструктор:

```
TRoad::TRoad ( float length0, int width0 )
{
    Length = length0;
    Width = width0;
}
```



## Вызов:

```
TRoad road ( 60, 3 );
```

# Защита от неверных данных

```
TRoad::TRoad ( float length0, int width0 )  
{  
    if ( length0 > 0 )  
        Length = length0;  
    else Length = 1;  
    if ( width0 > 0 )  
        Width = width0;  
    else Width = 1;  
}
```

# Значения параметров по умолчанию

```
class TRoad
{
public:
    ...
    TRoad ( float length0, int width0 = 3 );
};
```

значение по умолчанию

последние в списке параметров

**Вызов:**

```
TRoad road ( 60 );
```

width = 3

# Класс «Машина»

```
class TCar
{
    public:
        float X, V;
        int P;
        TRoad *Road;
        void move();
        TCar(); // конструктор без параметров
        TCar ( TRoad *road0, int p0, float v0 );
};
```

координата,  
скорость

полоса

дорога, по  
которой едет

# Конструкторы класса «Машина»

```
TCar::TCar ()  
{  
    Road = NULL;  
    P = 0; V = 0; X = 0;  
}
```

защита от ошибок –  
самостоятельно

```
TCar::TCar ( TRoad *road0, int p0,  
            float v0 )  
{  
    Road = road0;  
    P = p0; V = v0; X = 0;  
}
```

# Класс «Машина»: метод move

## Равномерное движение:

$$X = X_0 + V \cdot \Delta t$$

$\Delta t = 1$  интервал дискретизации

перемещение за одну единицу времени

```
void TCar::move()  
{  
    X = X + V;  
    if ( X > Road->Length ) X = 0;  
}
```

выезжает с другой стороны

обращение через указатель

# Основная программа

```
const int N=3;
TCar cars[N];
int i;
for ( i = 0; i < N; i++ )
{
    cars[i].Road = &road;
    cars[i].P = i + 1;
    cars[i].V = 2.0 * (i + 1);
}
do {
    for ( i = 0; i < N; i++ )
        cars[i].move();
}
while ( !kbhit() );
```

`#include <conio.h>`

пока не нажата  
(любая) клавиша

# Использование указателей

```
const int N = 3;
```

```
TCar *cars[N];
```

массив указателей

```
for ( i = 0; i < N; i++ )
```

```
    cars[i] = new TCar ( &road, i+1,  
                        2.0*(i+1) );
```

создание объектов

```
for ( i = 0; i < N; i++ )
```

```
    cars[i] -> move();
```



# Что в этом хорошего и плохого?

**ООП** – это метод разработки **больших** программ!



- основная программа – простая и понятная
- классы могут разрабатывать разные программисты независимо друг от друга (+интерфейс!)
- повторное использование классов



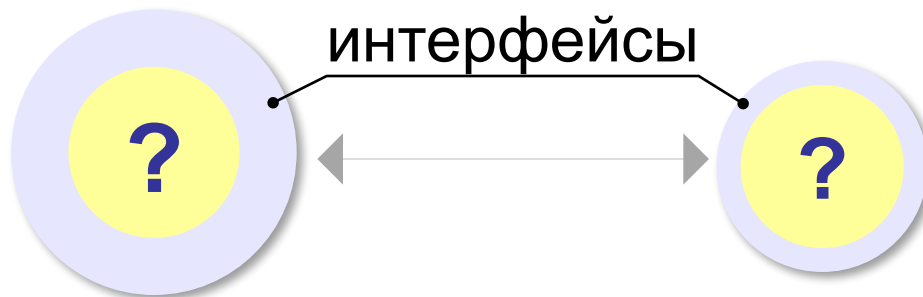
- неэффективно для небольших задач

# Объектно-ориентированное программирование. Языки C++ и C#

## **§ 49. Скрытие внутреннего устройства**

# Зачем скрывать внутреннее устройство?

## Объектная модель задачи:



- ✚ защита внутренних данных
- проверка входных данных на корректность
- изменение устройства с сохранением интерфейса

**Инкапсуляция** («помещение в капсулу») – скрывание внутреннего устройства объектов.



Также объединение данных и методов в одном объекте!

## Пример: класс «перо»

```
class TPen
```

```
{
```

*Field* – поле

R

G

B

```
    private:
```

```
        string FColor; // цвет, "FF00FF"
```

```
};
```



По умолчанию все члены класса закрытые – **private**!

```
class TPen
```

```
{
```

```
    private:
```

```
        string FColor;
```

```
    public:
```

```
        string getColor ();
```

```
        void setColor ( string newColor );
```

```
};
```



Как к ним обращаться?

методы  
доступа

## Пример: класс «перо»

Получить значение:

```
string TPen::getColor ()  
{  
    return FColor;  
}
```

Записать значение:

```
void TPen::setColor ( string newColor )  
{  
    if ( newColor.length() != 6 )  
        FColor = "000000";  
    else FColor = newColor;  
}
```

если ошибка,  
чёрный цвет



Защита от неверных данных!

# Пример: класс «перо»

## Использование:

установить  
цвет

```
TPen pen;  
pen.setColor ( "FFFF00" );  
cout << "цвет пера: " << pen.getColor();
```



Не очень удобно!

прочитать  
цвет

```
pen.color = "FFFF00";  
cout << pen.color;
```

нельзя в  
C++

# Изменение внутреннего устройства

Удобнее хранить цвет в виде числа:

```
class TPen
```

```
{
```

```
private:
```

```
    int FColor;
```

```
public:
```

```
    string getColor();
```

```
    void setColor( string newColor );
```

```
};
```



Найди отличие!

изменилось внутреннее устройство



Интерфейс не изменился!

# Преобразования `int` → `hex`

Использование потока (байтов):

16711935 →

→ "FF00FF"

записываем в поток число в шестнадцатеричной системе

читаем строку

```
#include <sstream>
```

подключить строковые потоки

```
stringstream s;  
s << hex << FColor;
```

? Что плохо?

255 → "FF"

"0000FF"

правильно так!



# Преобразования `hex` ↔ `int`

```
#include <sstream>
#include <iomanip>
```

подключить  
манипуляторы

```
string TPen::getColor()
{
    stringstream s;
    s << setfill('0')
      << setw(6)
      << hex << FColor;
    return s.str();
}
```

заполнять не  
пробелами, а нулями

вывести 6 знаков

прочитать строку  
`string` из потока

255 → "0000FF"

# Преобразования **hex** → **int**

Использование потока:

**"FF00FF"** →

записываем в  
поток строку

⊙ → 16711935

читаем число в  
шестнадцатеричной системе

```
void TPen::setColor ( string newColor )
{
    stringstream s;
    if ( newColor.length() != 6 )
        FColor = 0; // чёрный цвет
    else {
        s << newColor;
        s >> hex >> FColor;
    }
}
```

записываем строку

читаем число

# Свойства в C#

## Доступ с помощью методов:

```
TPen pen;  
pen.setColor ( "FFFF00" );  
cout << "цвет пера: " << pen.getColor();
```

## Доступ с помощью свойства color:

```
TPen pen;  
pen.color = "FFFF00";  
cout << "цвет пера: " << pen.color;
```

ВЫЗОВ `pen.setColor`

ВЫЗОВ `pen.getColor`

**Свойство** – это способ доступа к внутреннему состоянию объекта, имитирующий обращение к его внутренней переменной.

# СВОЙСТВА В C#

```
class TPen
```

закрытое поле

```
{
```

```
    private string FColor;
```

```
    public string color
```

открытое  
свойство

```
{
```

```
    get { return FColor; }
```

метод чтения

```
    set { FColor = value; }
```

метод записи

```
}
```

```
}
```

## Использование:

```
TPen pen;
```

```
pen.color = "FFFF00";
```

```
string s = pen.color;
```

# Свойства в C#

## Защита от неверного ввода данных:

```
public string color
{
    get { return FColor; }
    set
    {
        if ( value.Length != 6 )
            FColor = "000000";
        else FColor = value;
    }
}
```

переданное  
значение

# Свойства в C#

## Изменение внутреннего устройства:

```
class TPen
{
    private int FColor;
    public string color
    {
        get {
            return FColor.ToString ( "x6" );
        }
        set {
            FColor = Convert.ToInt32 (value, 16);
        }
    }
}
```

в строку

шестнадцатеричный формат, 6 знаков

переданное значение

в целое

из шестнадцатеричной записи



Данные – целое число, свойство – строка!

## СВОЙСТВО «ТОЛЬКО ДЛЯ ЧТЕНИЯ»

---

**Скорость машины** МОЖНО ТОЛЬКО ЧИТАТЬ:

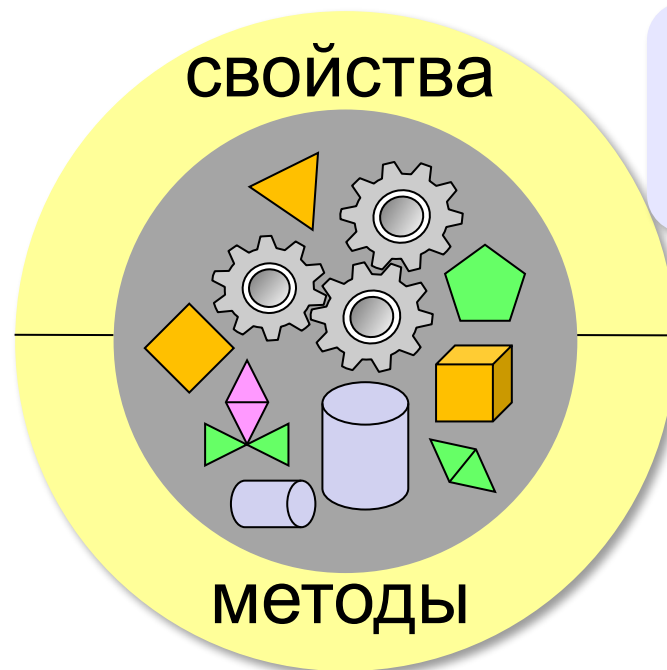
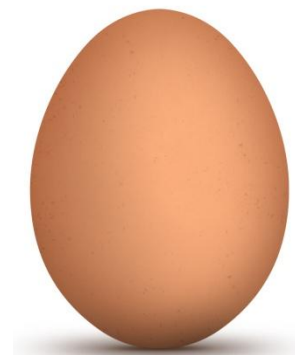
```
class TCar {  
    private: нет метода записи  
        double Fv;  
    public:  
        double getV() { return Fv; }  
};
```

**СВОЙСТВО на C#:**

```
class TCar {  
    private double Fv;  
    public double V {  
        get { return Fv; }  
    }  
};
```

# Скрытие внутреннего устройства

**Инкапсуляция** («помещение в капсулу»)



внутреннее  
устройство  
(**private**)

интерфейс  
(**public**)



# Объектно-ориентированное программирование. Языки C++ и C#

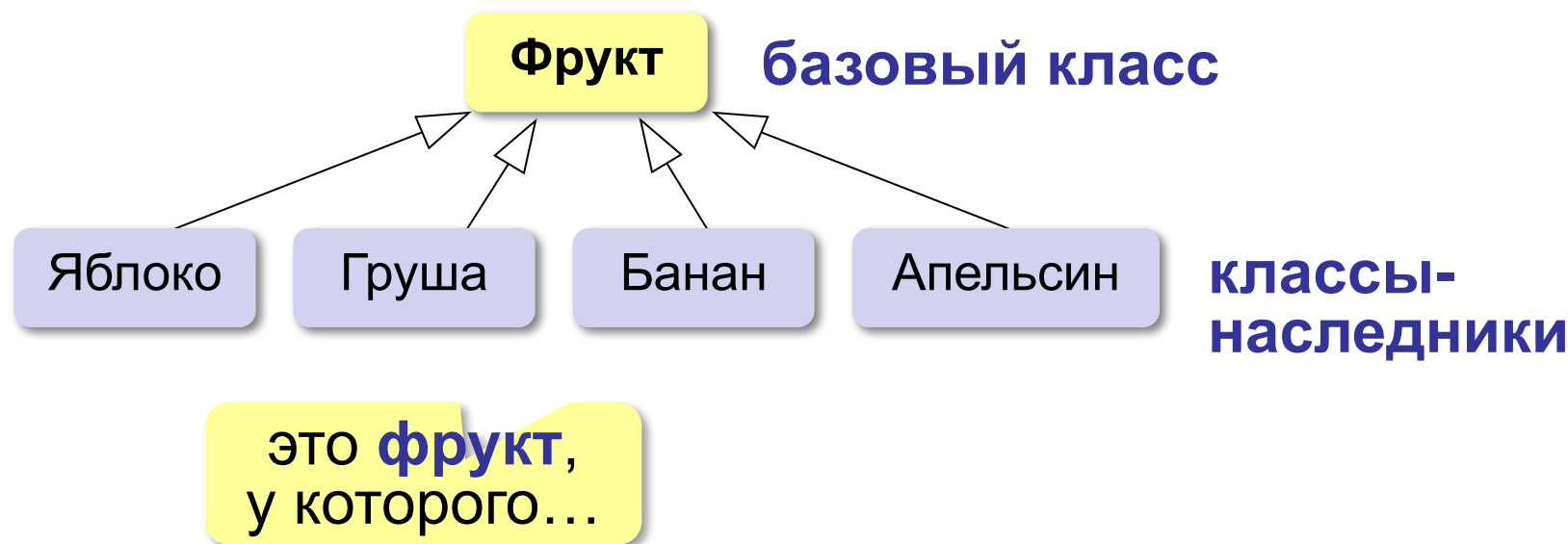
## **§ 50. Иерархия классов**

# Классификации

**?** Что такое классификация?

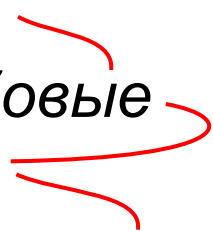
**Классификация** – разделение изучаемых объектов на группы (классы), объединенные общими признаками.

**?** Зачем это нужно?



# Что такое наследование?

класс *Двудольные*  
семейство *Бобовые*  
род *Клевер*  
**горный клевер**



наследует свойства  
(имеет все свойства)

Класс Б является **наследником** класса А, если можно сказать, что Б – **это разновидность** А.

✓ яблоко – фрукт

яблоко – **это** фрукт

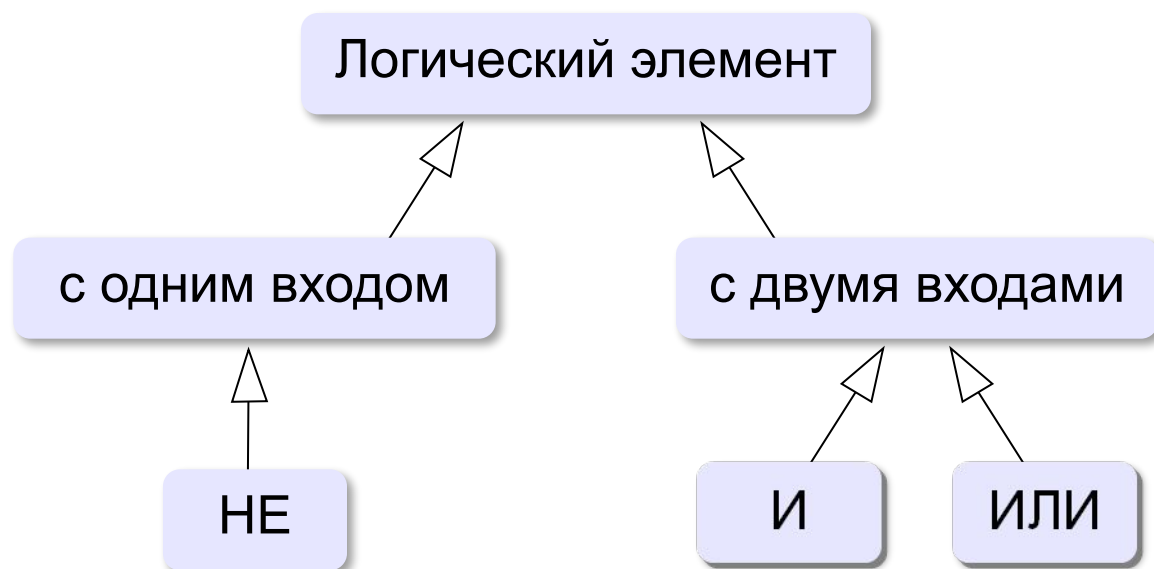
✓ горный клевер – клевер

горный клевер – **это**  
растение рода *Клевер*

✗ машина – двигатель

машина **содержит**  
двигатель (часть – целое)

# Иерархия логических элементов



**Объектно-ориентированное программирование** – это такой подход к программированию, при котором программа представляет собой множество взаимодействующих **объектов**, каждый из которых является экземпляром определенного **класса**, а классы образуют иерархию **наследования**.

# Базовый класс

## ЛогЭлемент

In1 (вход 1)

In2 (вход 2)

Res (результат)

calc

```
class TLogElement
{
    public:
        bool In1, In2, Res;
        void calc();
};
```

ВЫЧИСЛИТЬ  
ВЫХОД



Зачем хранить результат?

можно моделировать элементы  
с памятью (триггеры)



Что плохо?

# Базовый класс

```
class TLogElement
{
    private:
        bool FIn1, Fin2, FRes;
        void calc();
    public:
        bool getIn1() { return FIn1; }
        void setIn1 ( bool newIn1 );
        bool getIn2() { return Fin2; }
        void setIn2 ( bool newIn2 );
        bool getRes() { return FRes; }
};
```



Почему здесь?

ТОЛЬКО ДЛЯ  
ЧТЕНИЯ

# Установка входа

```
void TLogElement::setIn1 (bool newIn1)
{
    FIn1 = newIn1 ;
    calc () ;
}
```



Что ещё?

пересчёт при изменении  
входа



Как написать процедуру **calc**?

```
void TLogElement::calc ()
{
}
```

заглушка



Проблема: наследники должны менять **calc**!

# Что такое полиморфизм?

греч.: *πολυ* — много, *μορφη* — форма

**Полиморфизм** — это возможность классов-наследников по-разному реализовать метод, описанный для класса-предка.



Проблема: открыть данные и методы для наследников и закрыть для остальных!

```
class TLogElement
{
    ...
    protected:
        void calc();
};
```

защищённые элементы:  
доступны только  
наследникам



# Базовый класс

```
class TLogElement
{
    private:
        bool FIn1, FIn2;
    protected:
        bool FRes;
        virtual void calc()=0;
        bool getIn2() { return FIn2; }
        void setIn2 ( bool newIn2 );
    public:
        bool getIn1() { return FIn1; }
        void setIn1 ( bool newIn1 );
        bool getRes() { return FRes; }
};
```

наследники будут  
изменять поле

# Базовый класс

```
class TLogElement
{
    protected:
        bool FRes;
        virtual void calc()= 0;
        bool getIn2() { return Fin2; }
        void setIn2 ( bool newIn2 );
        ...
};
```

наследники будут  
изменять поле

для элементов с одним  
входом не нужно!

**virtual** (виртуальный) – этот метод могут переопределять классы-наследники

**= 0** (абстрактный метод) – этот метод базовый класс не будет реализовывать (оставляет наследникам)

# Абстрактный класс

- все логические элементы должны иметь метод `calc`
- метод `calc` невозможно написать, пока неизвестен тип логического элемента

**Абстрактный метод** – это метод класса, который объявляется, но не реализуется в классе.

**Абстрактный класс** – это класс, содержащий хотя бы один абстрактный метод.

нет логического элемента «вообще», как не «фрукта вообще», есть конкретные виды



Нельзя создать объект абстрактного класса!

`TLogElement` – абстрактный класс из-за метода `calc`

# Элемент «НЕ»

наследник от  
**TLogElement**

```
class TNot: public TLogElement
{
    protected:
        void calc();
};
```

переопределяет метод  
базового класса

```
void TNot::calc()
{
    FRes = !getIn1();
}
```



Почему не **! FIn1**?



Это уже не абстрактный класс!

# Элемент «НЕ»

---

## Использование:

```
TNot n;
```

создание объекта

```
n.setIn1 ( false );
```

установка входа

```
cout << n.getRes ();
```

вывод результата

# Элементы с двумя входами

сохранить права  
доступа

наследник от  
**TLogElement**

```
class TLog2In: public TLogElement
{
    public:
        TLogElement::setIn2;
        TLogElement::getIn2;
};
```

ПОВЫСИТЬ «ВИДИМОСТЬ»  
(**protected** → **public**)



Можно ли создать объект этого класса?

нельзя, он абстрактный

## Элементы с двумя входами

```
class TAnd: public TLog2In
{
    protected:
        void calc();
};
class TOr: public TLog2In
{
    protected:
        void calc();
};
```

элемент «И»

элемент «ИЛИ»

## Элементы с двумя входами

```
void TAnd::calc()  
{  
    FRes = getIn1() && getIn2();  
}
```

элемент «И»

```
void TAnd::calc()  
{  
    FRes = getIn1() || getIn2();  
}
```

элемент «ИЛИ»

доступ к защищённому  
полю (**protected**)



Почему не обратиться к **FIn1** и **FIn2**?



# Вызов виртуального метода

В базовом классе:

```
void TLogElement::setIn1 ( bool newIn1 )  
{  
    FIn1 = newIn1;  
    calc();  
}
```




Какой метод вызывается?

```
class TLogElement  
{  
    protected:  
    virtual void calc()=0;  
    ...  
};
```

# Виртуальный метод

---

## Статическое связывание:

транслятор  сывает в код адрес процедуры

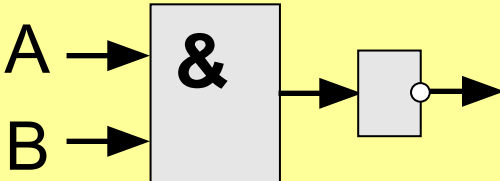
## Динамическое связывание:

адрес процедуры определяется во время выполнения программы в зависимости от типа объекта

**Виртуальный метод** – это метод базового класса, который могут переопределить классы-наследники так, что конкретный адрес вызываемого метода определяется только при выполнении программы.

## Пример: элемент «И-НЕ»

```
main ()
{
    TNot elNot;
    TAnd elAnd;
    int A, B;
    cout << "  A  B  !(A&B) " << endl;
    cout << "-----" << endl;
    for ( A=0; A<=1; A++ ) {
        elAnd.setIn1 ( A );
        for ( B=0; B<=1; B++ ) {
            elAnd.setIn2 ( B );
            elNot.setIn1 ( elAnd.getRes() );
            cout << "  " << A << "  " << B
                 << "      " << elNot.getRes() << endl;
        }
    }
}
```



# Модульность

---

Идея: выделить классы в отдельный модуль.

Интерфейс (`log_elem.h`):

```
class TLogElement
{ ... }
class TLog2In: public TLogElement
{ ... }
class TNot: public TLogElement
{ ... }
class TAnd: public TLog2In
{ ... }
class TOr: public TLog2In
{ ... }
```

# Модульность

Модуль (`log_elem.cpp`):

```
#include <log_elem.h>
```

```
void TLogElement::setIn1 ( bool newIn1 )  
{ ... }
```

```
void TLogElement::setIn2 ( bool newIn1 )  
{ ... }
```

```
void TNot::calc()  
{ ... }
```

```
void TAnd::calc()  
{ ... }
```

```
void TOr::calc()  
{ ... }
```

реализация методов  
классов



Чего не хватает?

В основную программу:

```
#include <log_elem.h>
```

# Сообщения между объектами



Задача – автоматическая передача сигналов по цепочке!

```
class TLogElement
```

```
{
```

```
    private:
```

```
        TLogElement *FNextEl;
```

```
        int FNextIn;
```

```
        ...
```

```
    public:
```

```
        void Link ( TLogElement *nextElement,
```

```
                    int nextIn=1 );
```

```
};
```

адрес следующего  
элемента в цепочке

номер входа  
следующего элемента

# Сообщения между объектами

---

## Установка связи:

```
void TLogElement::Link(  
    TLogElement *nextElement,  
    int nextIn )  
{  
    FNextEl = nextElement;  
    FNextIn = nextIn;  
}
```

# Сообщения между объектами

После изменения выхода «дергаем» следующий элемент:

```
void TLogElement::setIn1 ( bool newIn1 )
{
    FIn1 = newIn1 ;
    calc ( ) ;
    if ( FNextEl )
        switch ( FNextIn ) {
            case 1:
                FNextEl->setIn1 ( getRes ( ) ) ;
            case 2:
                FNextEl->setIn2 ( getRes ( ) ) ;
        }
}
```

если следующий  
элемент установлен...

передать результат на  
нужный вход



# Сообщения между объектами

---



Как сделать, чтобы сначала **FNextEl = NULL**?

Новый конструктор:

```
TLogElement::TLogElement()  
{  
    FNextEl = NULL;  
}
```

# Сообщения между объектами

## Изменения в основной программе:

```
TNot elNot;
```

```
TAnd elAnd;
```

```
elAnd.Link ( &elNot );
```

установить  
связь

```
...
```

```
for ( A=0; A<=1; A++ ) {
```

```
    elAnd.setIn1 ( A );
```

```
    for ( B=0; B<=1; B++ ) {
```

```
        elAnd.setIn2 ( B );
```

```
        elNot.setIn1 ( elAnd.getRes () );
```

```
        ...
```

```
    }
```

```
}
```

это уже не  
нужно!

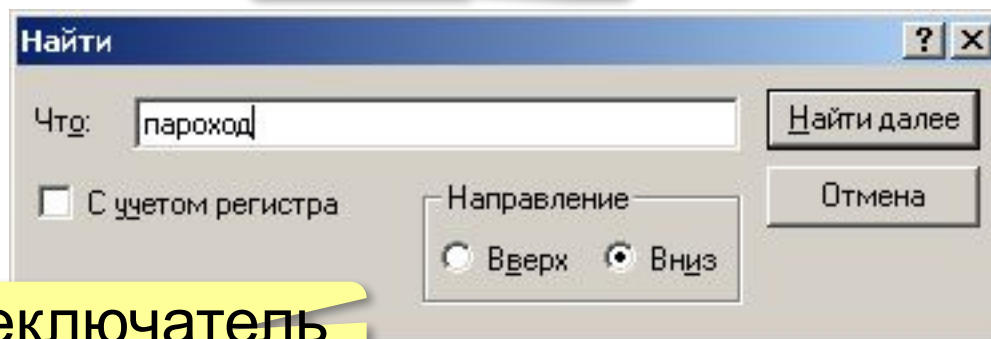
# Объектно-ориентированное программирование. Языки C++ и C#

## § 51. Программы с графическим интерфейсом

# Интерфейс: объекты и сообщения

поле ввода

флажок



кнопка

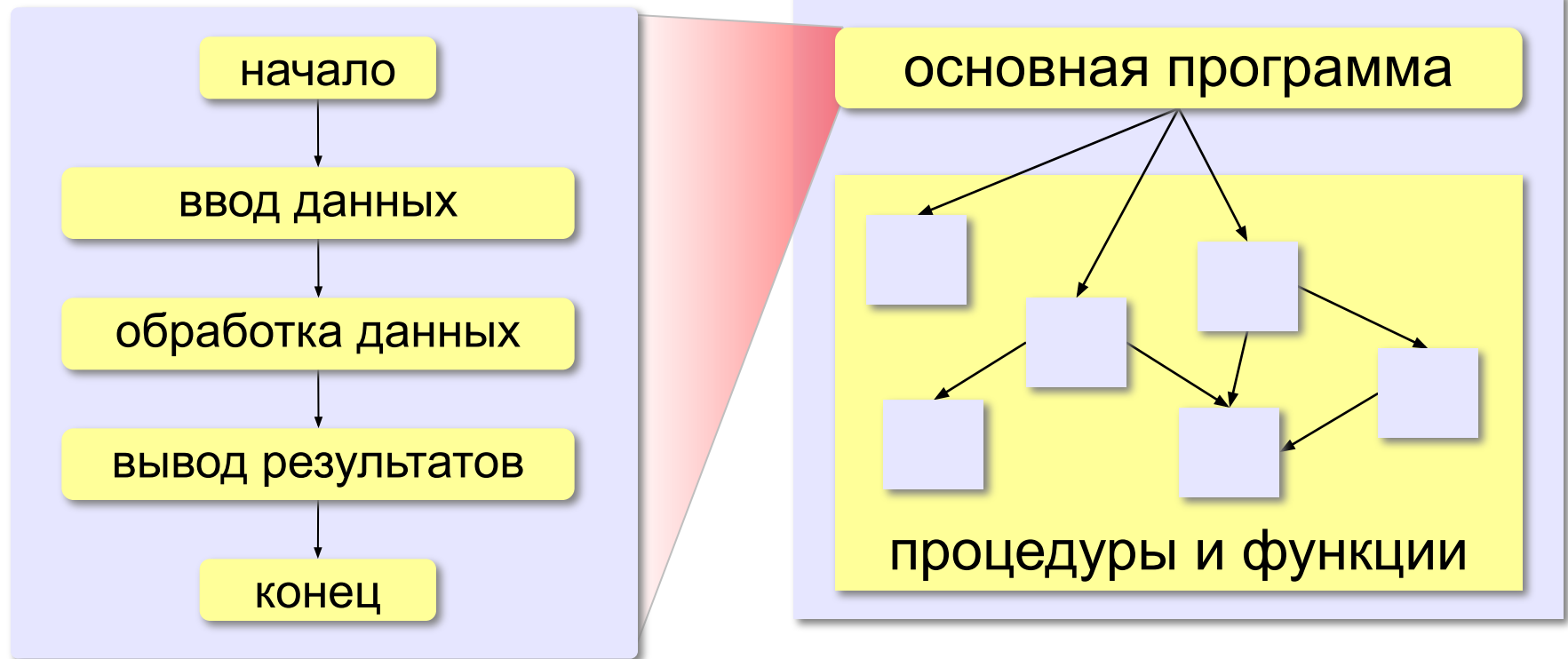
переключатель

Все элементы окон – объекты, которые обмениваются данными, посылая друг другу сообщения.

**Сообщение** – это блок данных определённой структуры, который используется для обмена информацией между объектами.

- адресат (кому) или *широковещательное*
- числовой код (тип) сообщения
- параметры (дополнительные данные)

# Классические программы



Порядок выполнения команд определяется программистом, пользователь не может вмешаться!

# Программы, управляемые событиями

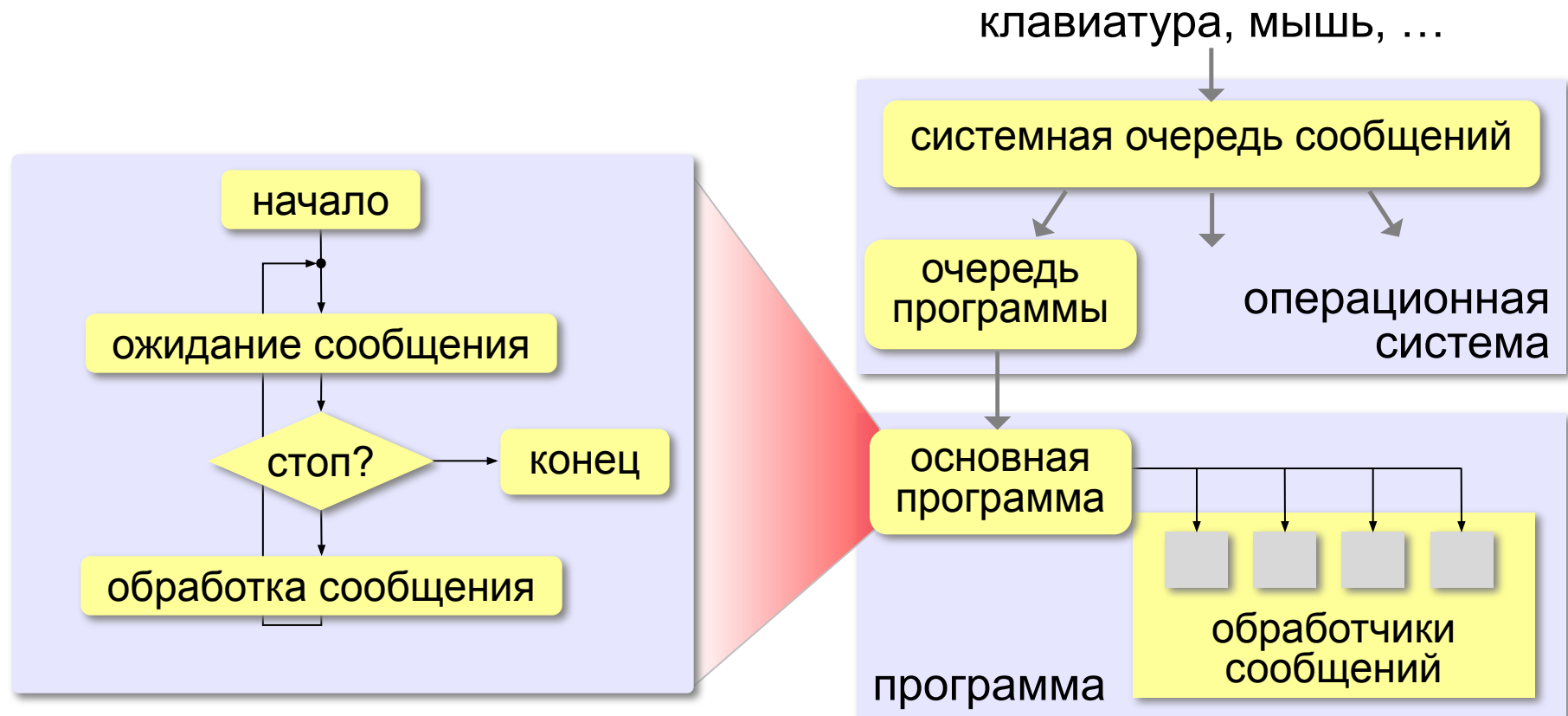
**Событие** – это переход какого-либо объекта из одного состояния в другое.

- нажатие на клавишу
- щелчок мышью
- перемещение окна
- поступление данных из сети
- запрос к веб-серверу
- завершение вычислений
- ...



Программа начинает работать при наступлении событий!

# Программы, управляемые событиями



**Программа управляется событиями!**

# Что такое RAD-среда?

**RAD** = *Rapid Application Development* — быстрая разработка приложений

## Этапы разработки:

- создание **формы**
- минимальный код добавляется автоматически
- расстановка **элементов интерфейса** с помощью мыши и настройка их свойств
- создание **обработчиков** событий
- написание **алгоритмов** обработки данных

**Форма** — это шаблон, по которому строится окно программы или диалога

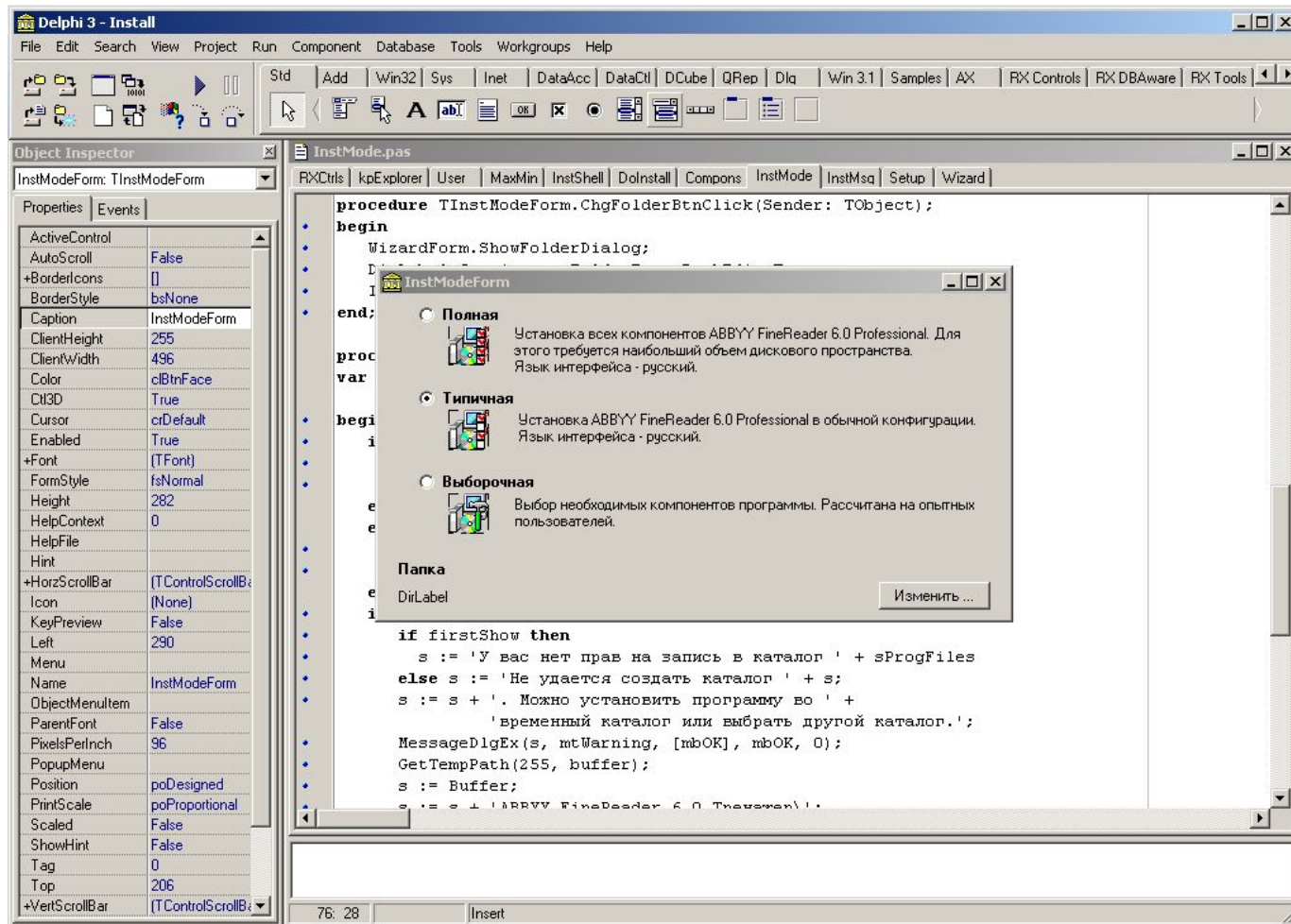
выполняются при  
возникновении событий



# RAD-среды: Delphi

**Язык:** *Object Pascal*, позднее *Delphi*:

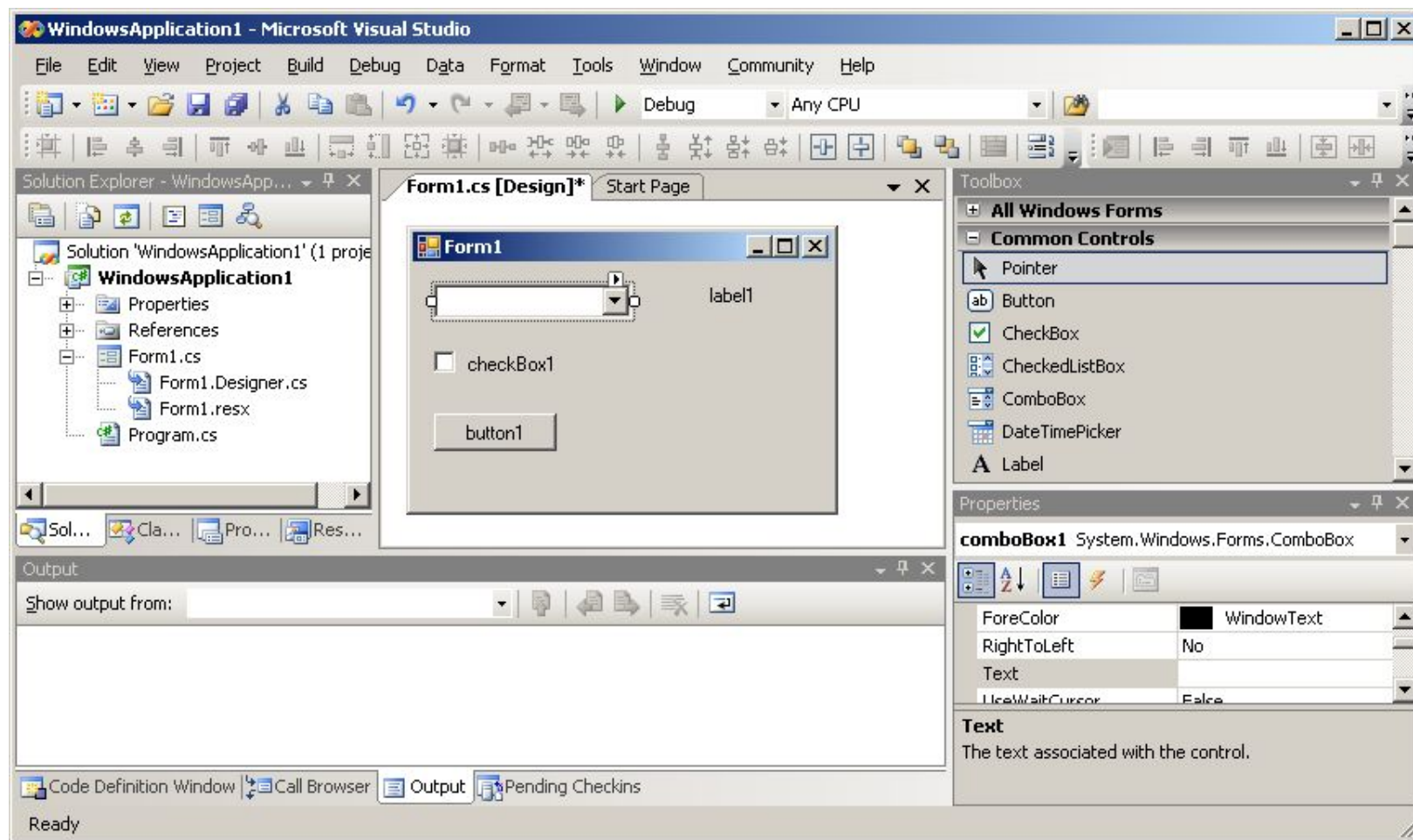
1995: *Borland*, сейчас: *Embarcadero Technologies*



# RAD-среды: MS Visual Studio

**ЯЗЫКИ:** *Visual Basic, Visual C++, Visual C#, Visual F#*

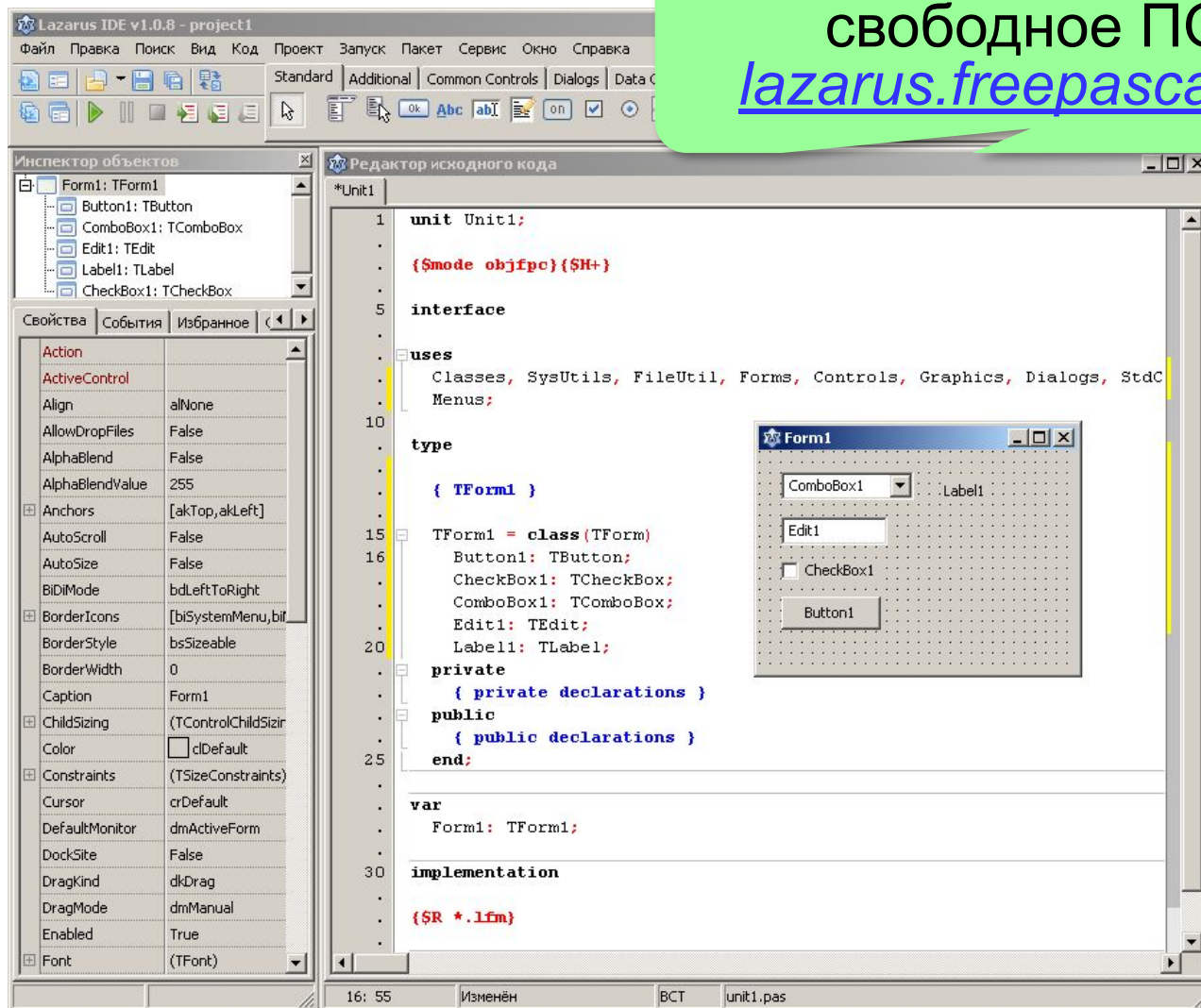
с 1995 по н.в.: *Microsoft*



# RAD-среды: Lazarus

Языки: *FreePascal, Delphi*

свободное ПО:  
[lazarus.freepascal.org](http://lazarus.freepascal.org)



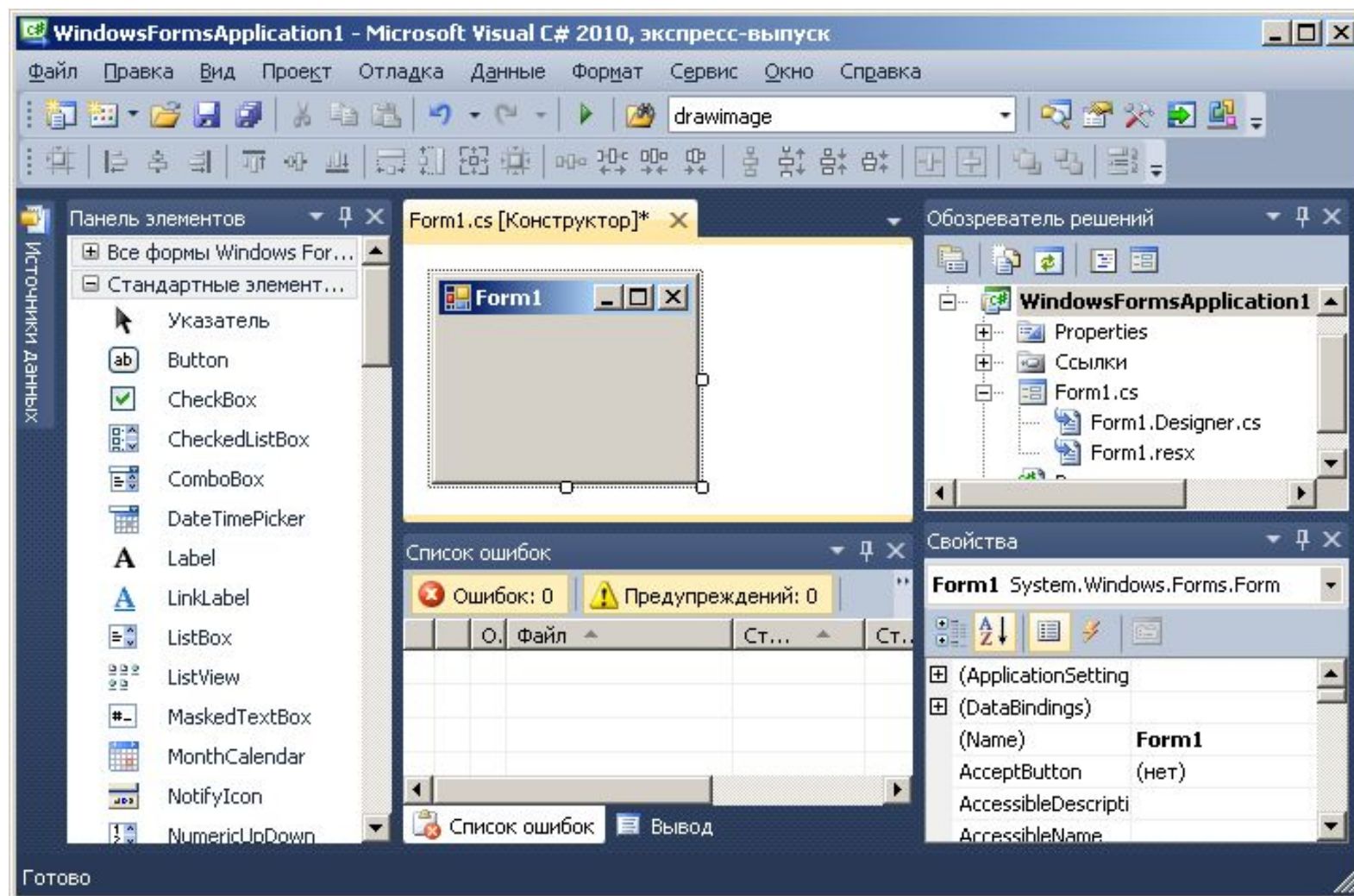
# Объектно-ориентированное программирование. Языки C++ и C#

## **§ 52. Программирование в RAD-средах**

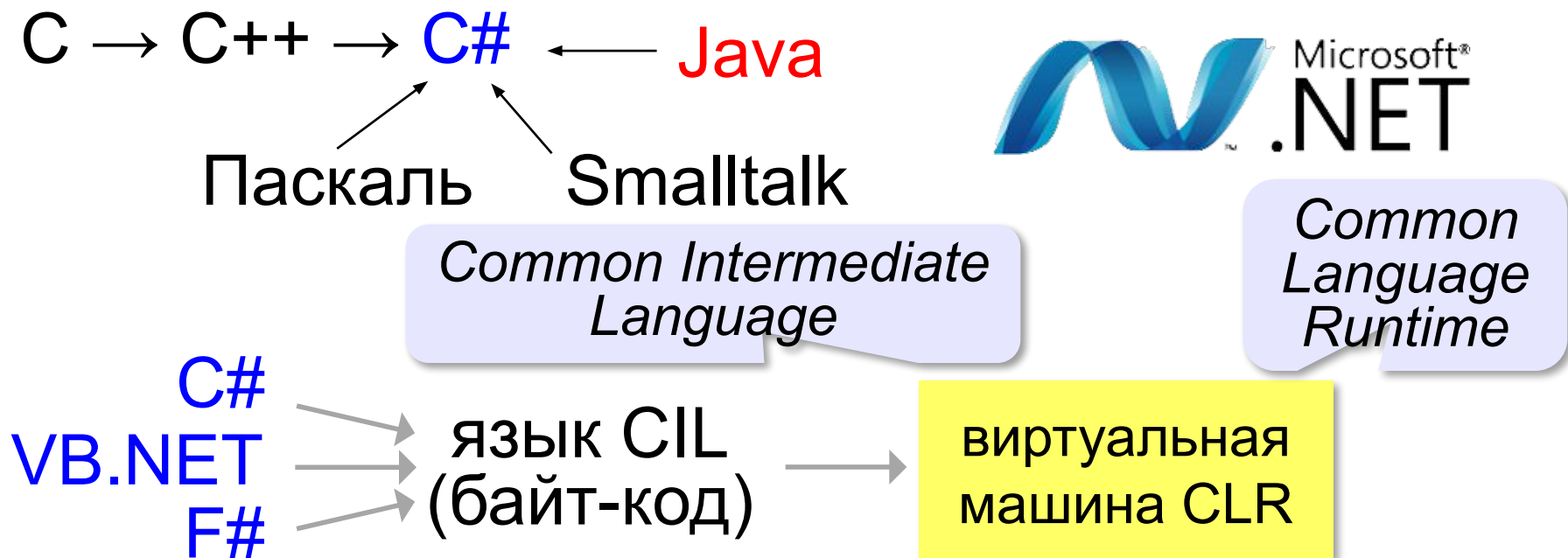


# Visual Studio Express (C#)

<http://www.visualstudio.com/ru-ru/products/visual-studio-express-vs.aspx>



# Язык C#



- ⊕ объединение программ на разных языках
- полностью ООП – для больших программ
- большая библиотека функций и компонентов

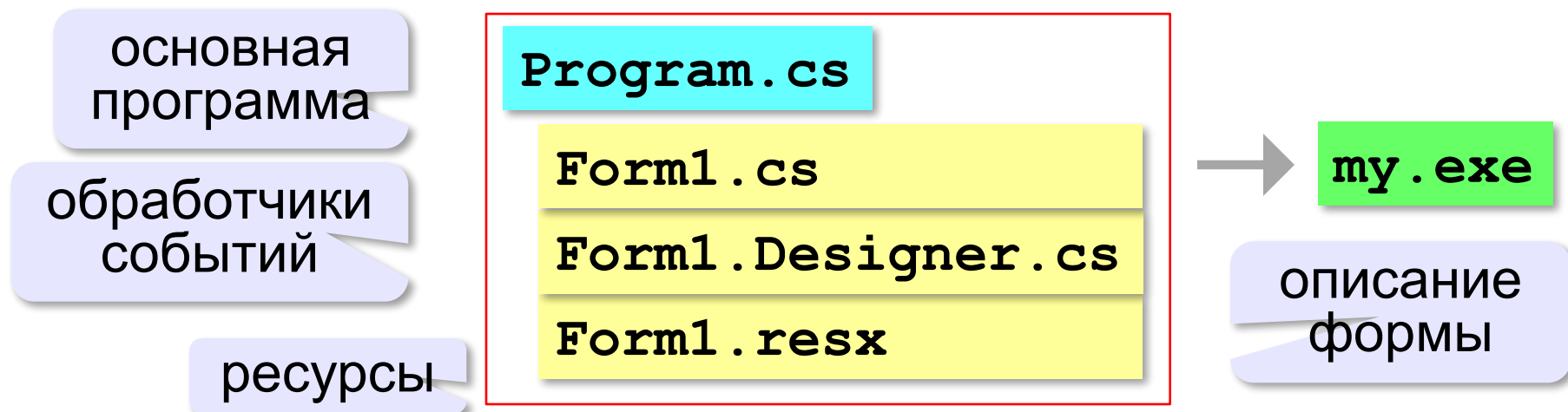
- ⊖ требовательна к ресурсам
- надёжно – только под *Windows*

*Linux* – проект  
*Mono*

# Проекты и решения

**Проект** – это набор файлов, из которых компилятор строит исполняемый файл программы.

- **проект** (`.csproj`, *CSharp Project*) – описание (XML)
- **модули**, из которых состоит программа (`*.cs`);
- **ресурсы** (`*.resx`) – строки (перевод сообщений).



**Решение** = один или несколько проектов.

# Простейший проект

Файл – Создать проект – Приложение Windows Forms

**!** F5 – запуск!

Редактор кода

Конструктор формы

Структура проекта

Панель элементов

Свойства



# Модуль формы

**F7** – перейти из конструктора к коду формы

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace Project1
{
    public partial class Form1: Form {
        public Form1() { InitializeComponent(); }
    }
}
```

библиотеки

пространство имён

конструктор

# Модуль формы

открытый  
класс

частичное  
описание

наследник  
класса **Form**

```
public partial class Form1: Form
{
    public Form1()
    {
        InitializeComponent();
    }
}
```

начальные  
установки



Код этого метода – в  
**Form1.Designer.cs!**

# Основная программа

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace Project1
{
    static class Program {
        static void Main() {
            ...
            Application.Run ( new Form1 () );
        }
    }
}
```

библиотеки

статический класс

создание формы

запуск цикла  
обработки сообщений



Статический класс – набор методов!

# Свойства формы



**MainForm**

**Name** – имя формы

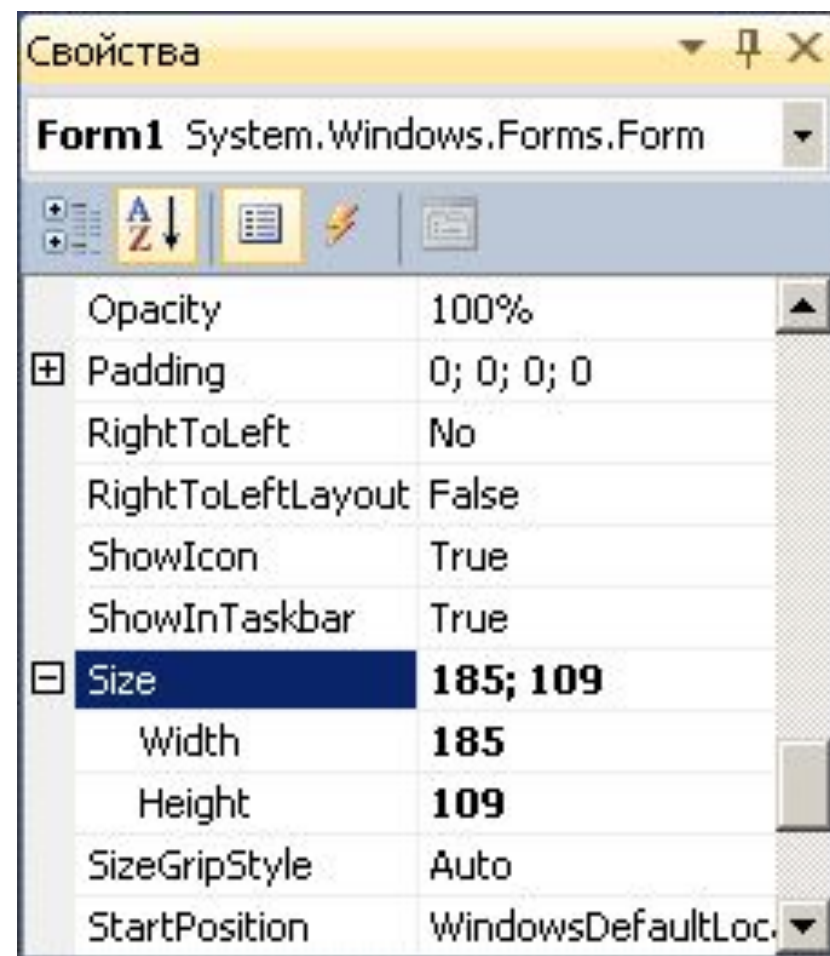
**Size.Width** – ширина

**Size.Height** – высота

**Text** – текст в заголовке окна

**BackColor** – цвет фона

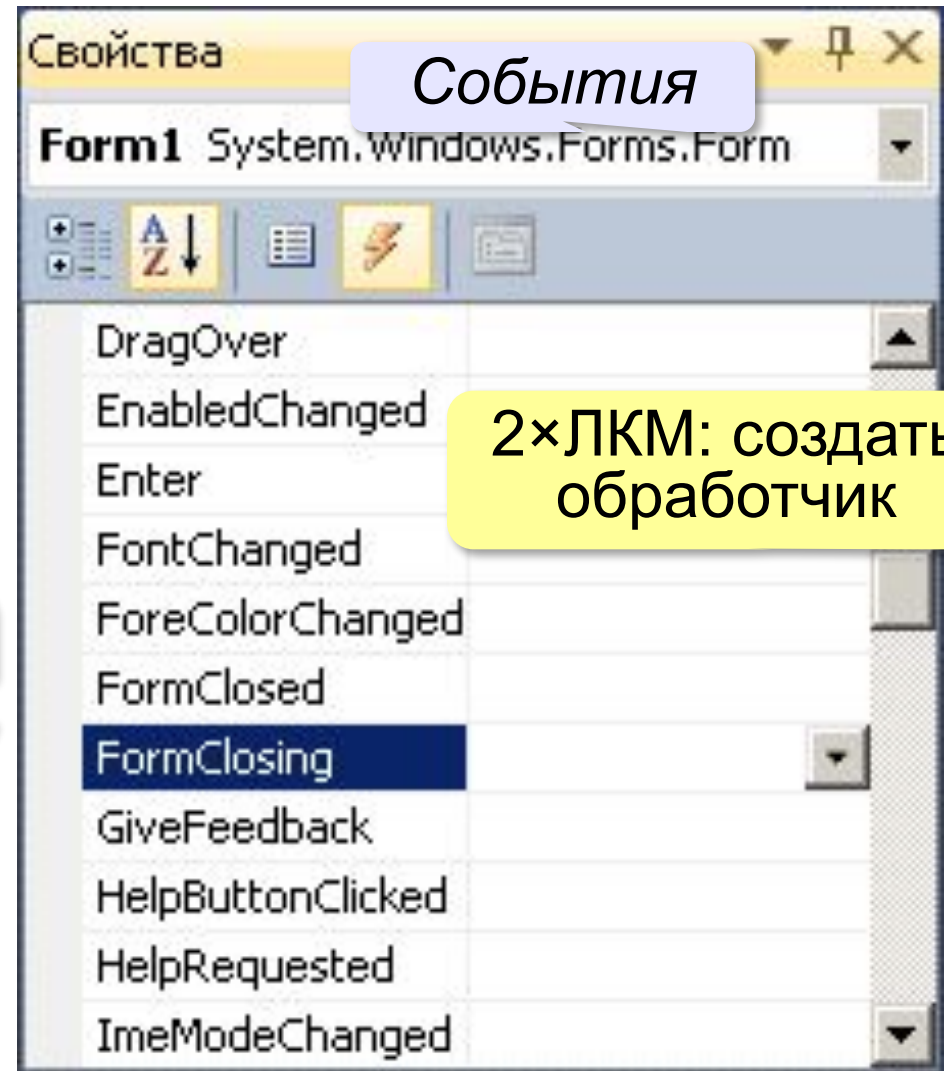
**Font** – шрифт надписей



# Обработчик событий



**FormClosing:**  
форма закрывается



# Обработчик события

закрытый  
метод класса  
**MainForm**

название  
обработчика

общий предок  
всех объектов

```
private void MainForm_FormClosing (  
    object sender,  
    FormClosingEventArgs e )  
{  
}  
}
```

кто послал  
сообщение

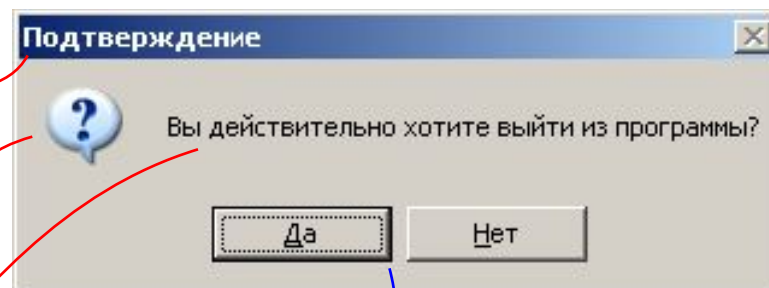
дополнительные  
данные о событии



Автоматически добавлен в **Form1.Designer.cs**!

# Диалог с вопросом

Метод `MessageBox.Show`:



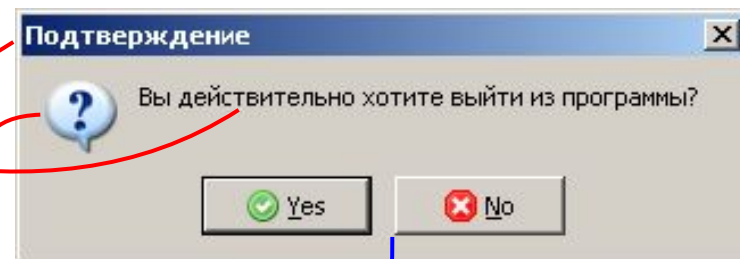
```
private void MainForm_FormClosing (
    object sender, FormClosingEventArgs e)
{
    DialogResult res;
    res = MessageBox.Show (
        "Вы действительно хотите выйти из программы?",
        "Подтверждение",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question
    );
    if ( res == DialogResult.No )
        e.Cancel = true;
}
```

тип: результат диалога

нажали «Нет»,  
отменить закрытие

# Параметры `MessageBox.Show`

- сообщение пользователю
- заголовок окна
- тип запроса `MessageBoxIcon`
  - `Error` ошибка
  - `Warning` предупреждение
  - `Information` информация
  - `Question` вопрос
- набор (множество) кнопок `MessageBoxButtons`:
  - `YesNo` «Да», «Нет»
  - `YesNoCancel` «Да», «Нет», «Отмена»
  - `OK` «ОК»
  - `OKCancel` «ОК», «Отмена»

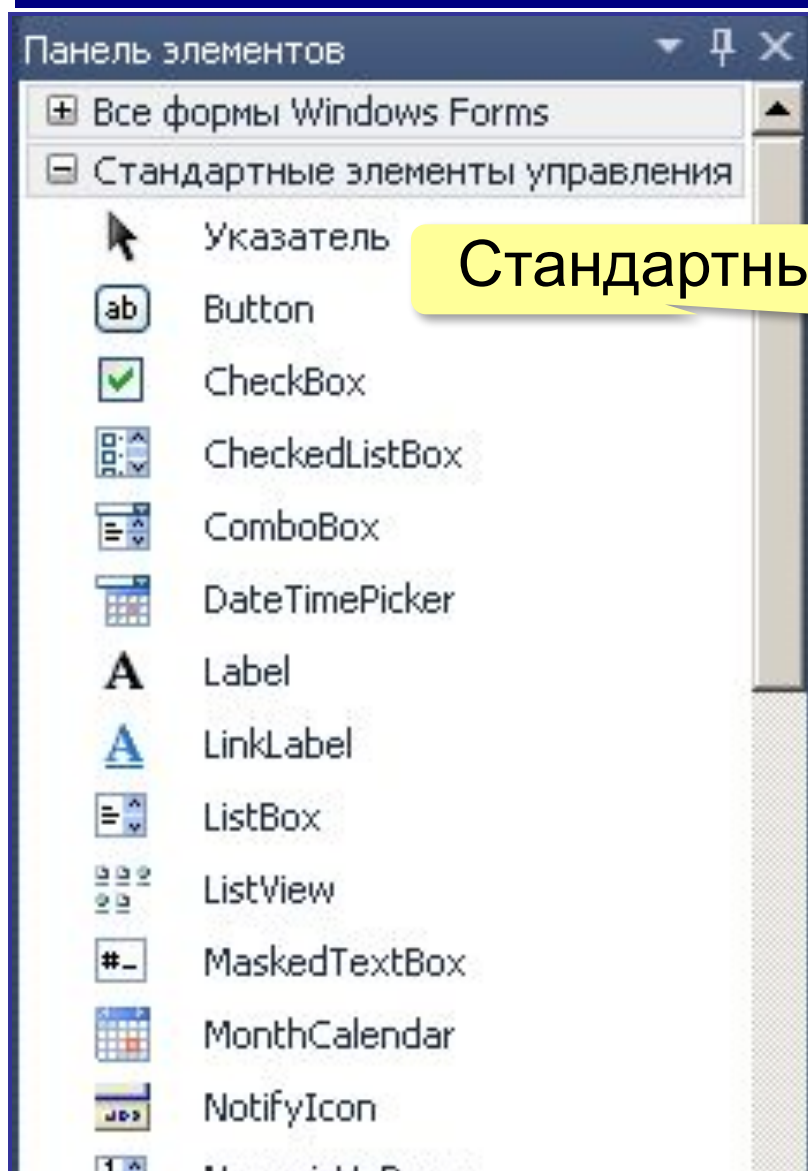




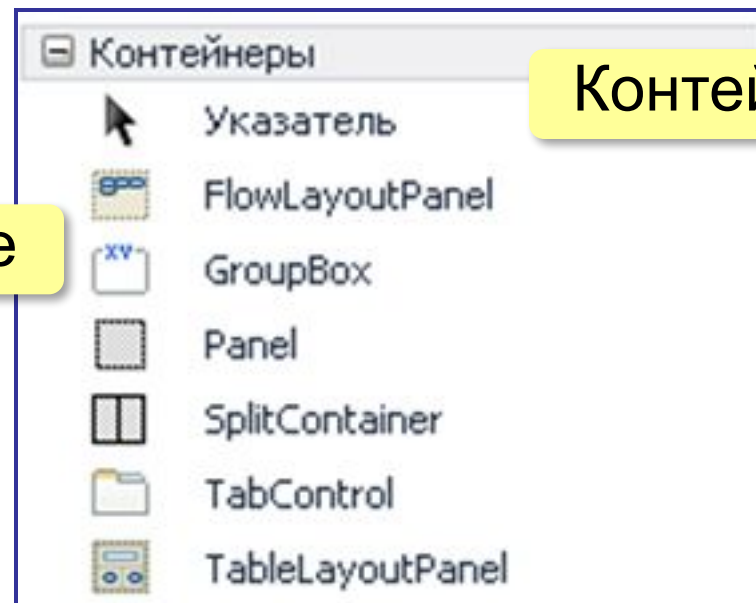
# Объектно-ориентированное программирование. Языки C++ и C#

## **§ 53. Использование компонентов**

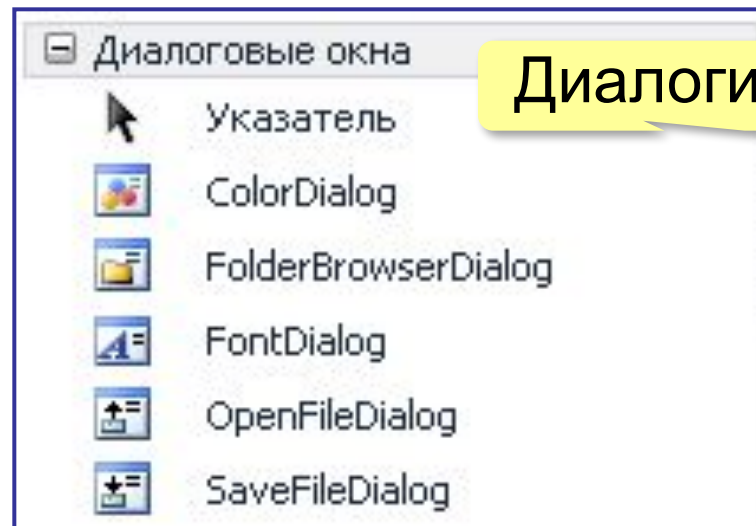
# Панель компонентов



Стандартные

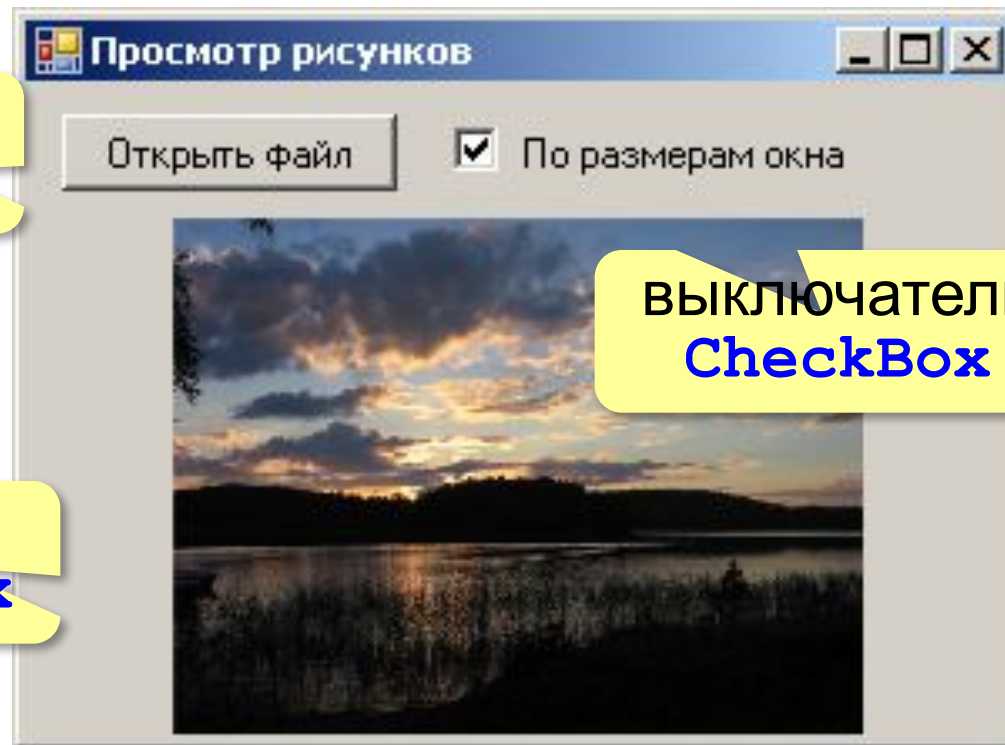


Контейнеры



Диалоги

# Просмотр рисунков



кнопка  
**Button**

панель  
**Panel**

рисунок  
**PictureBox**

выключатель  
**CheckBox**

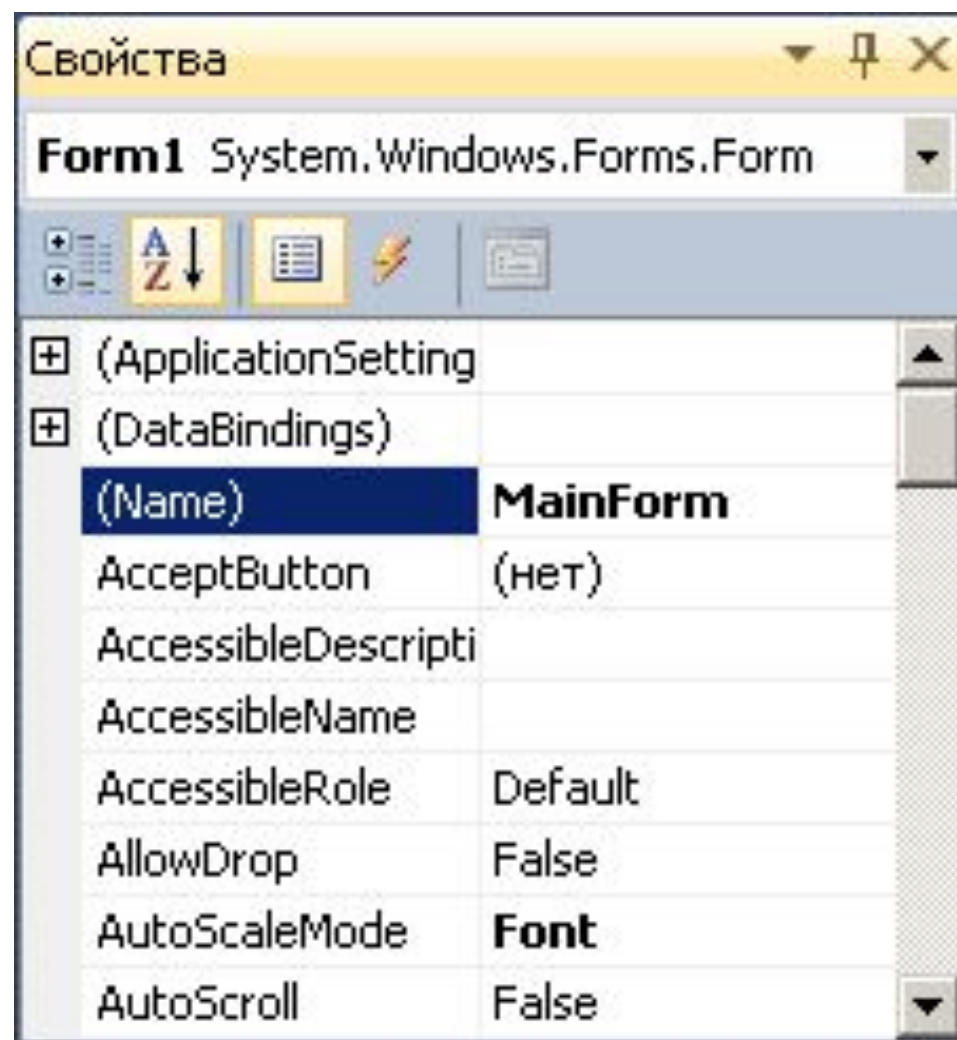
# Настройка формы

*Файл – Создать проект – Приложение Windows Forms*

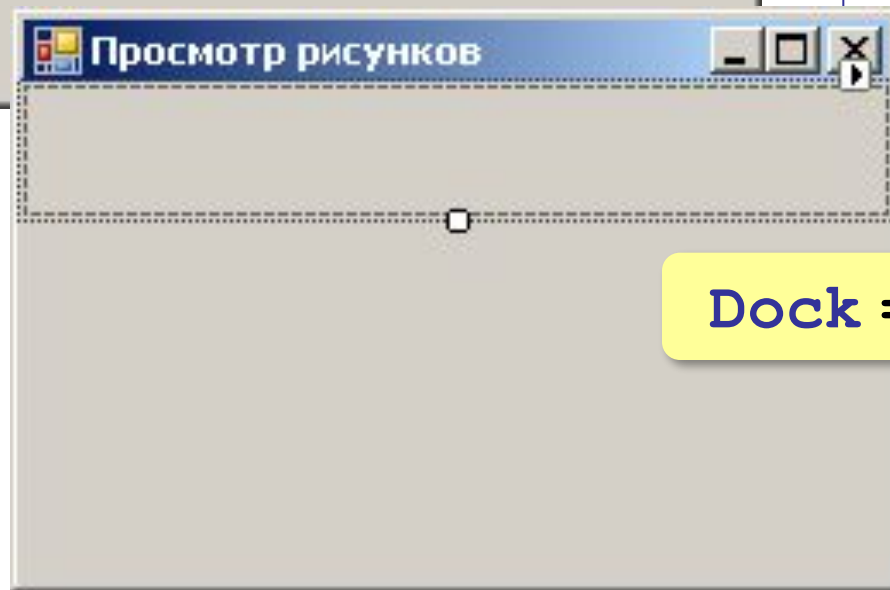
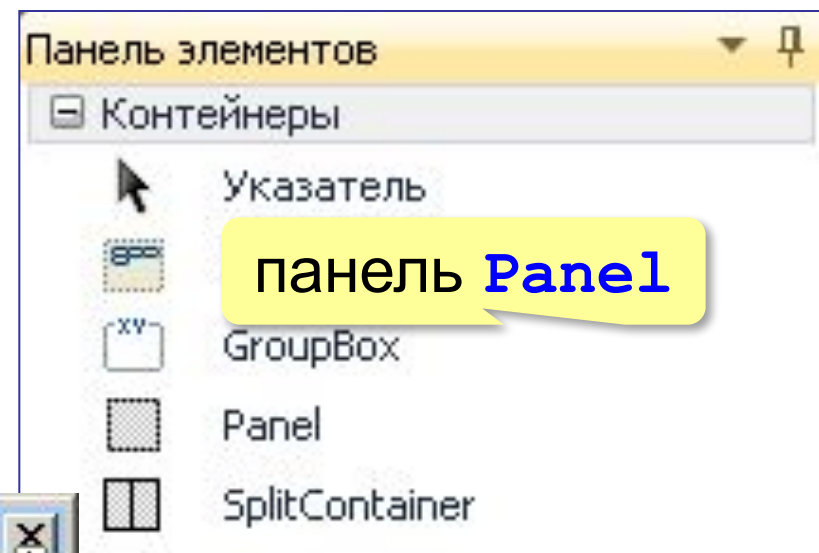


**Name** → MainForm

**Text** → Просмотр  
рисунков



# Верхняя панель

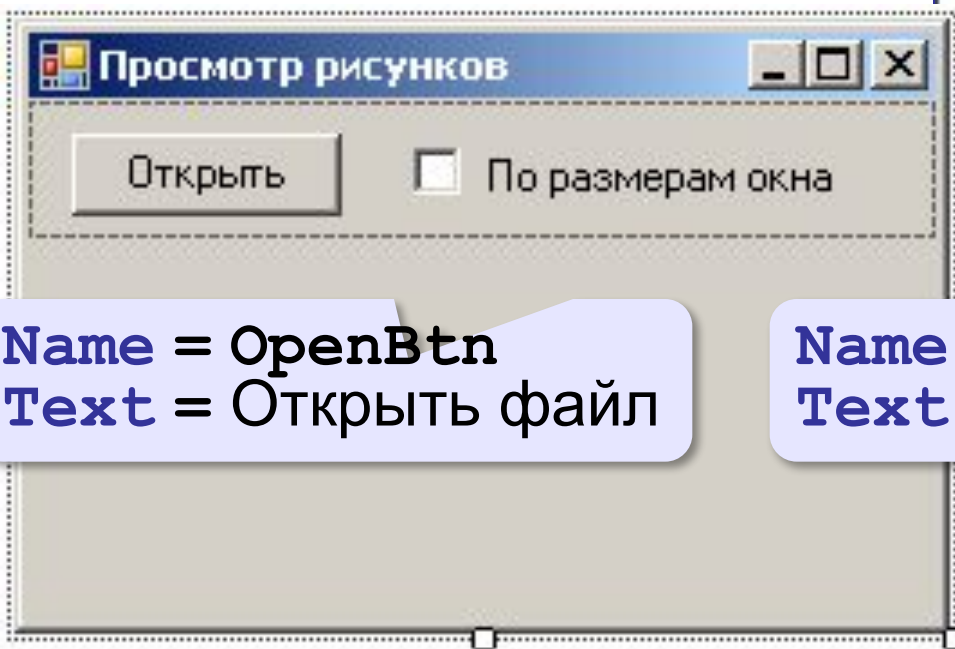


**Dock = Top**

# Кнопка и выключатель

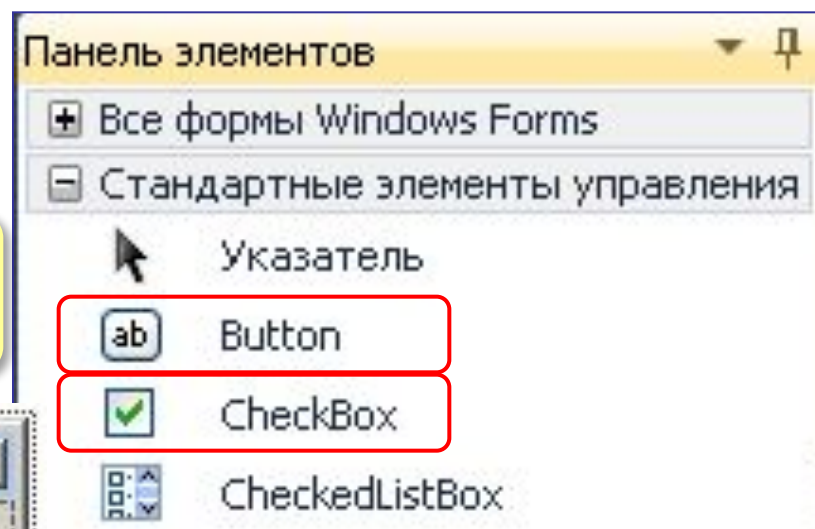
КНОПКА  
**Button**

ВЫКЛЮЧАТЕЛЬ  
**CheckBox**



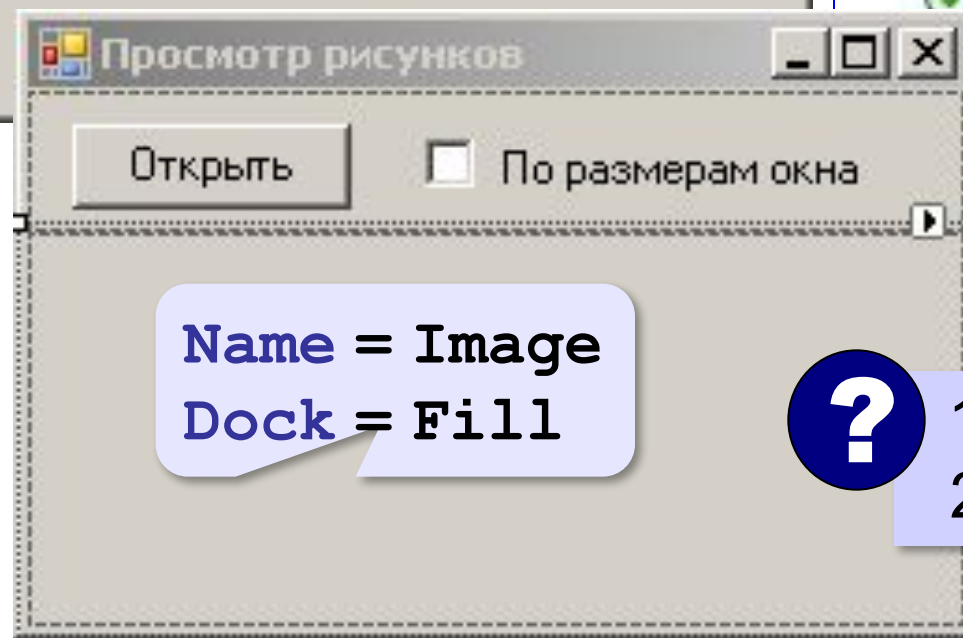
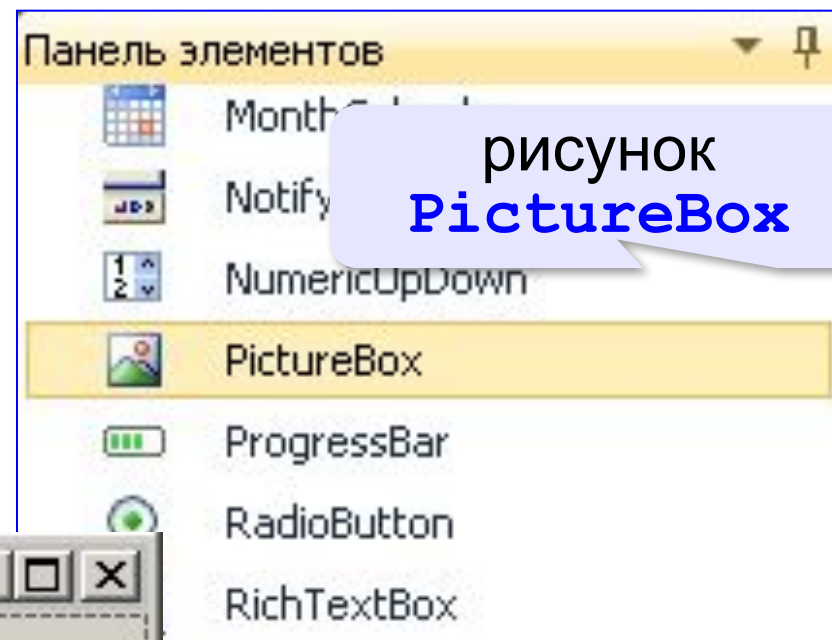
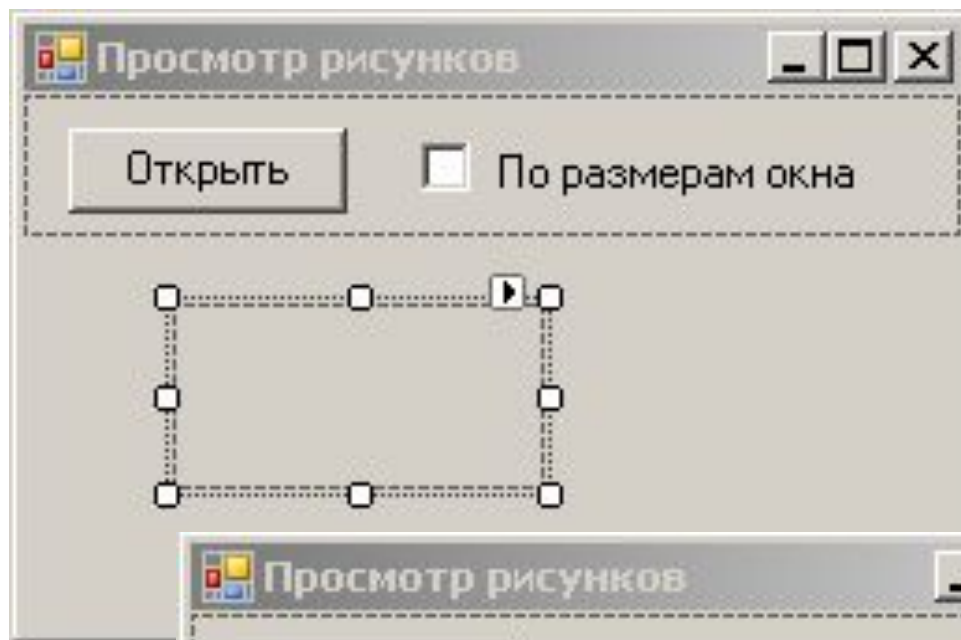
**Name** = OpenBtn  
**Text** = Открыть файл

**Name** = SizeCB  
**Text** = По размерам окна



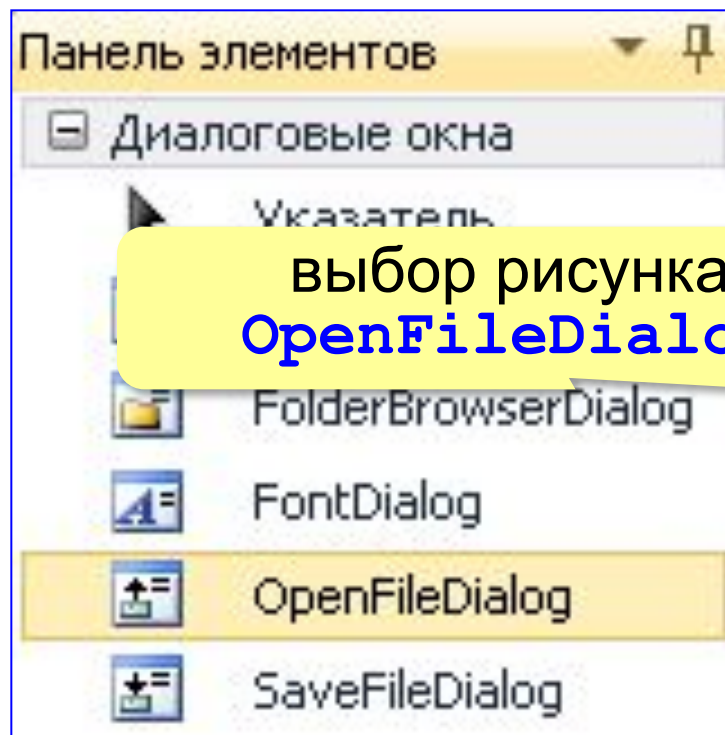


# Компонент PictureBox

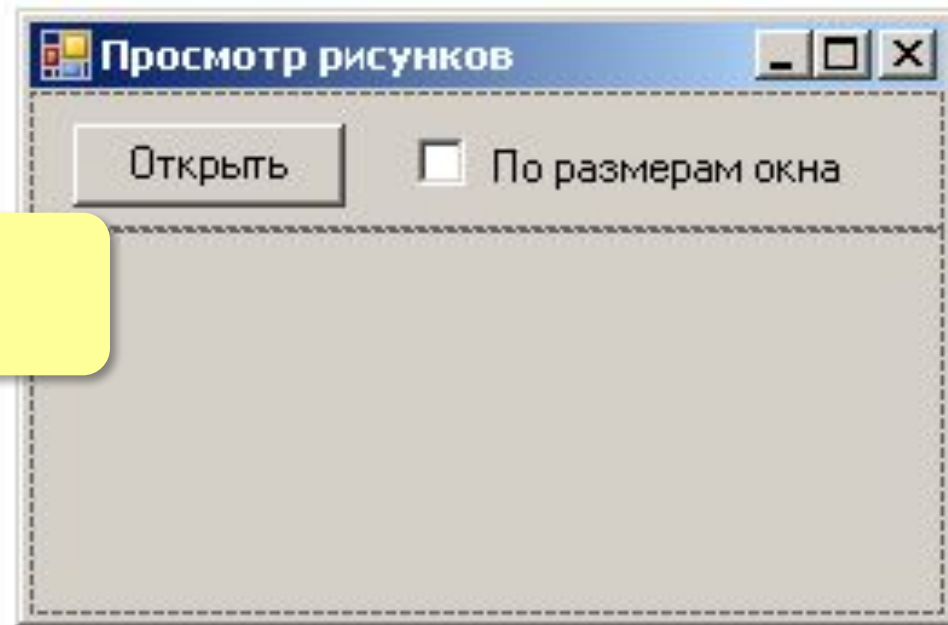


- 1) Загрузка файла?
- 2) Масштабирование?

# Выбор файла



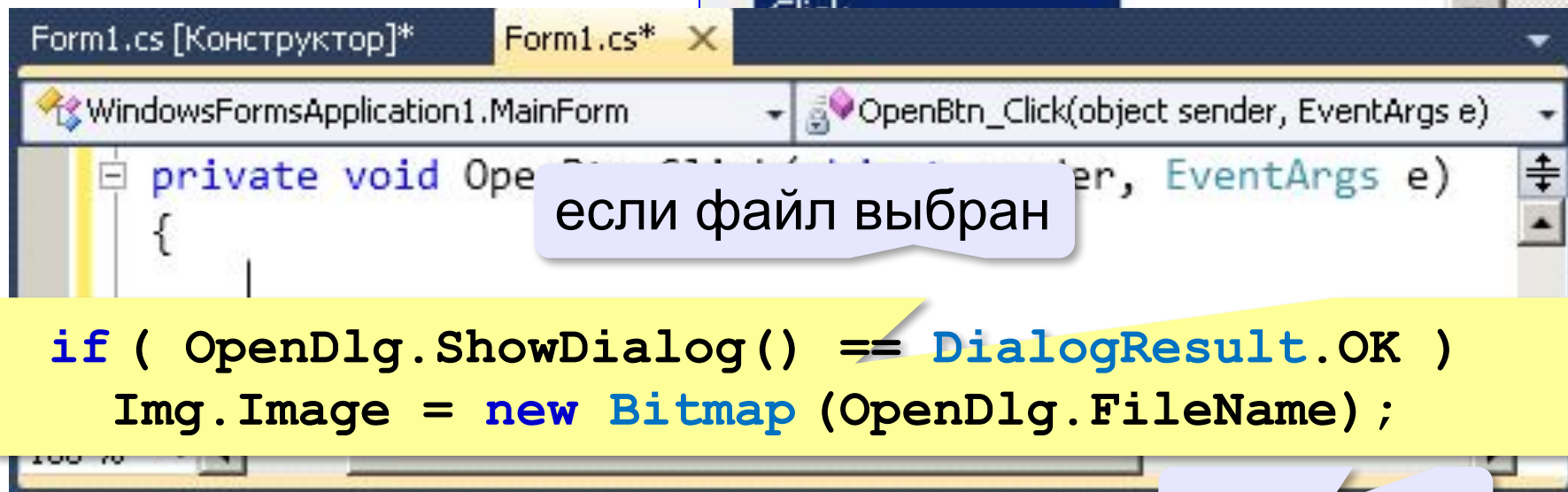
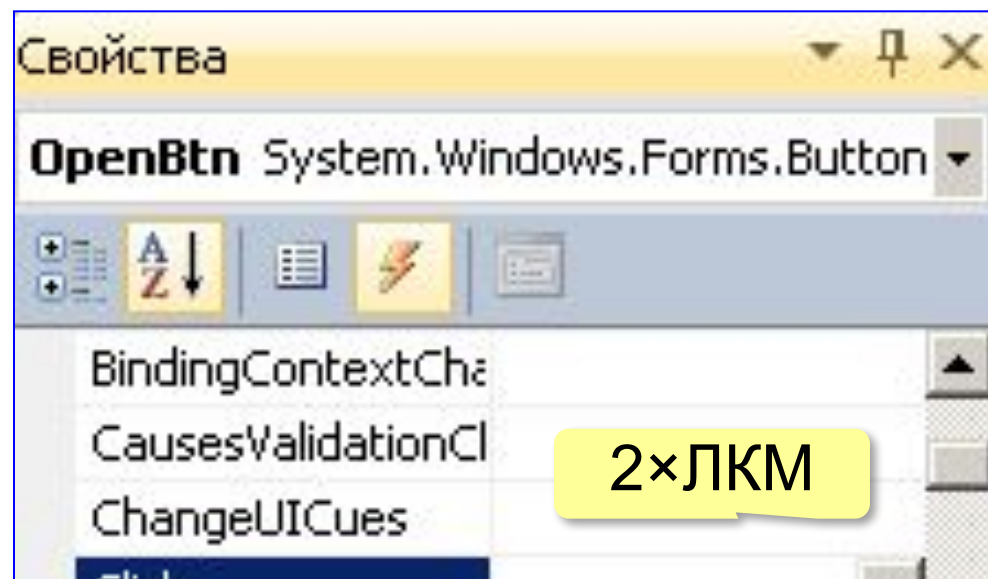
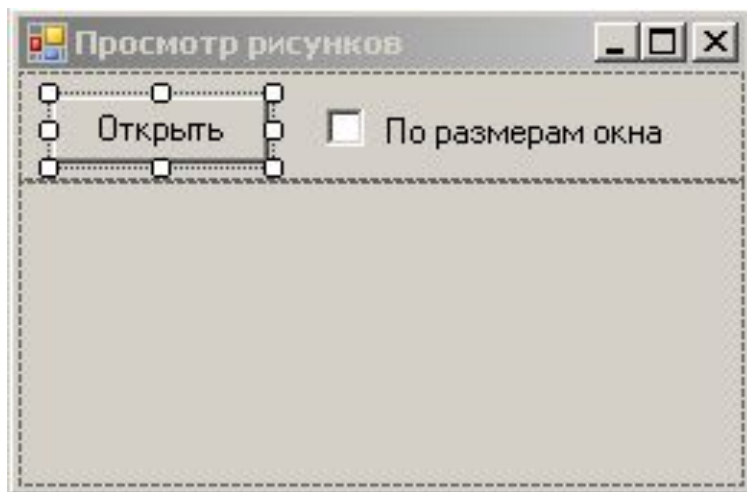
выбор рисунка  
**OpenFileDialog**



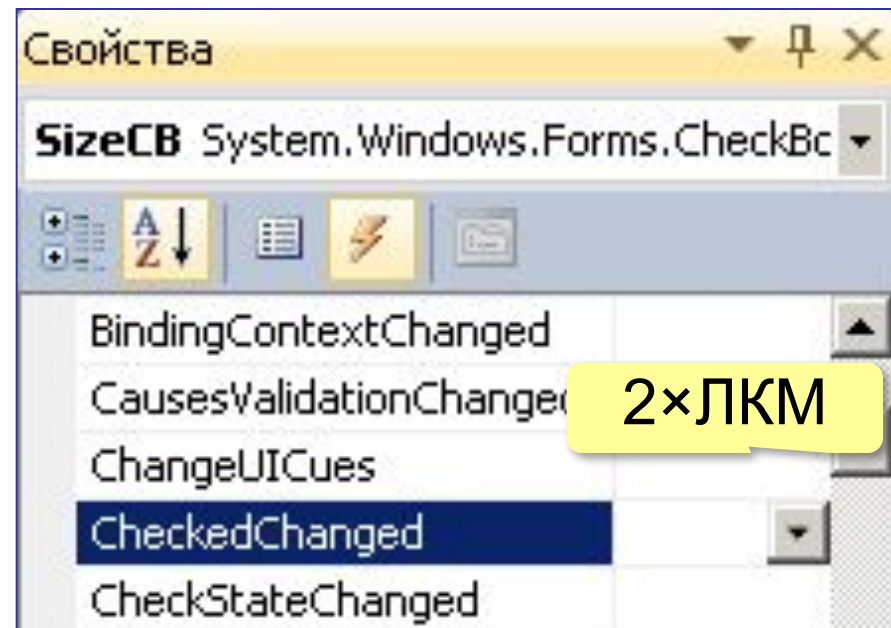
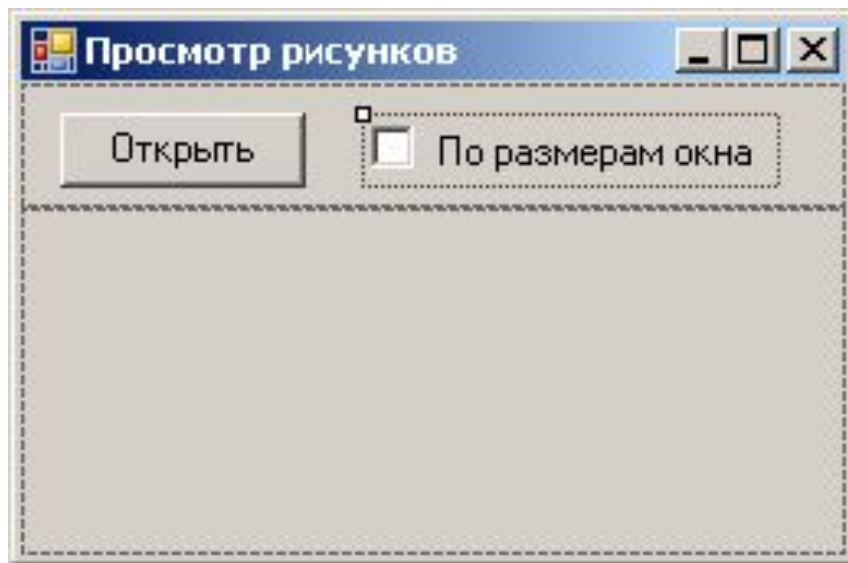
**Name = OpenFileDialog**



# Выбор файла



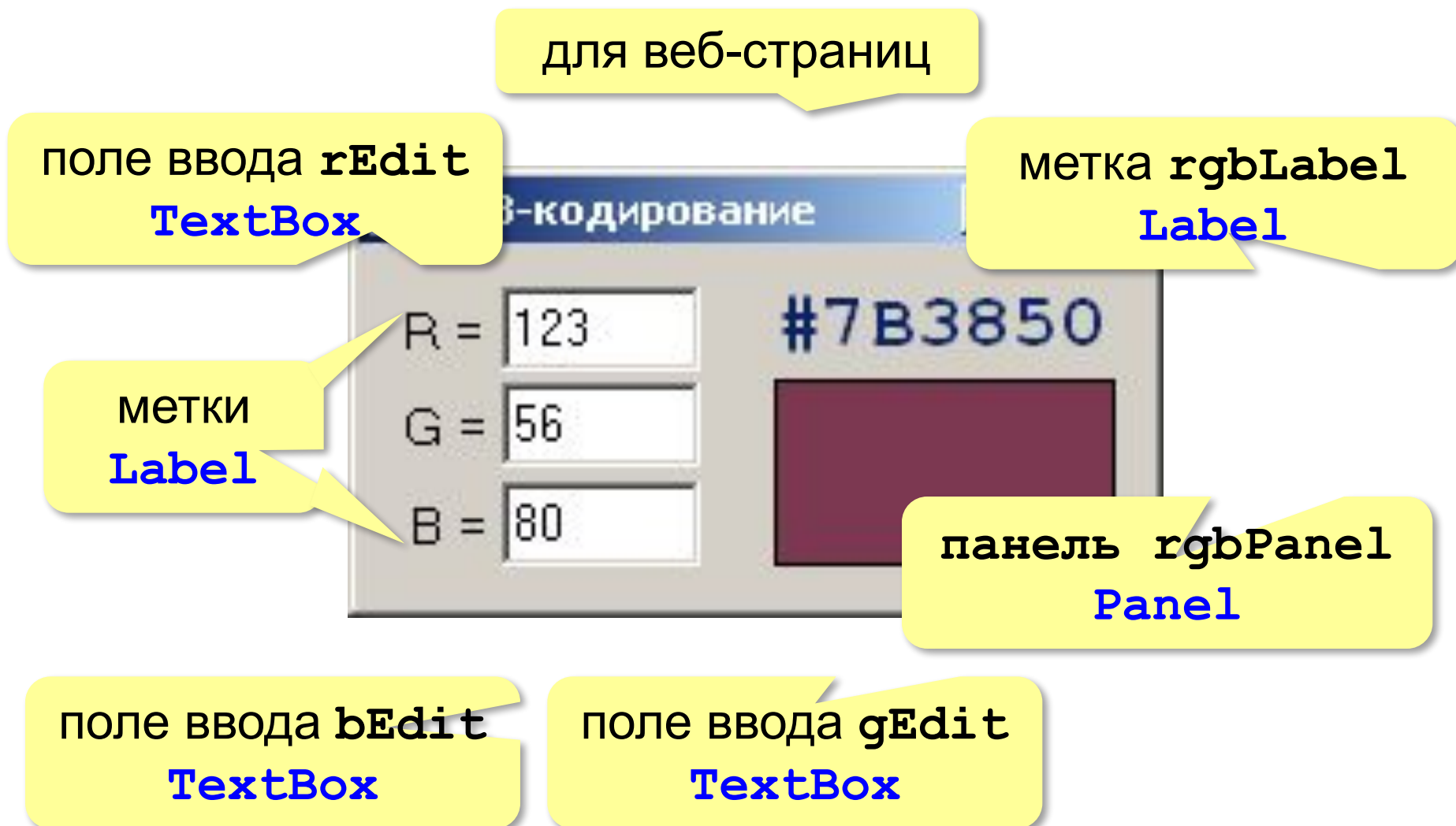
# Масштабирование



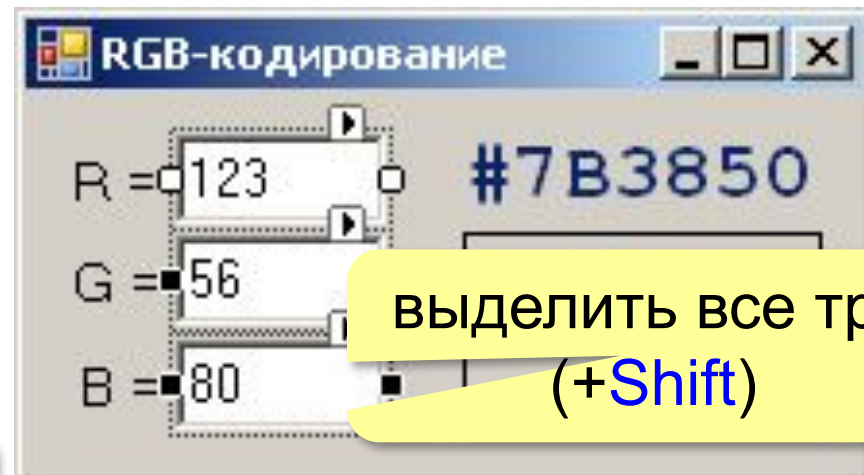
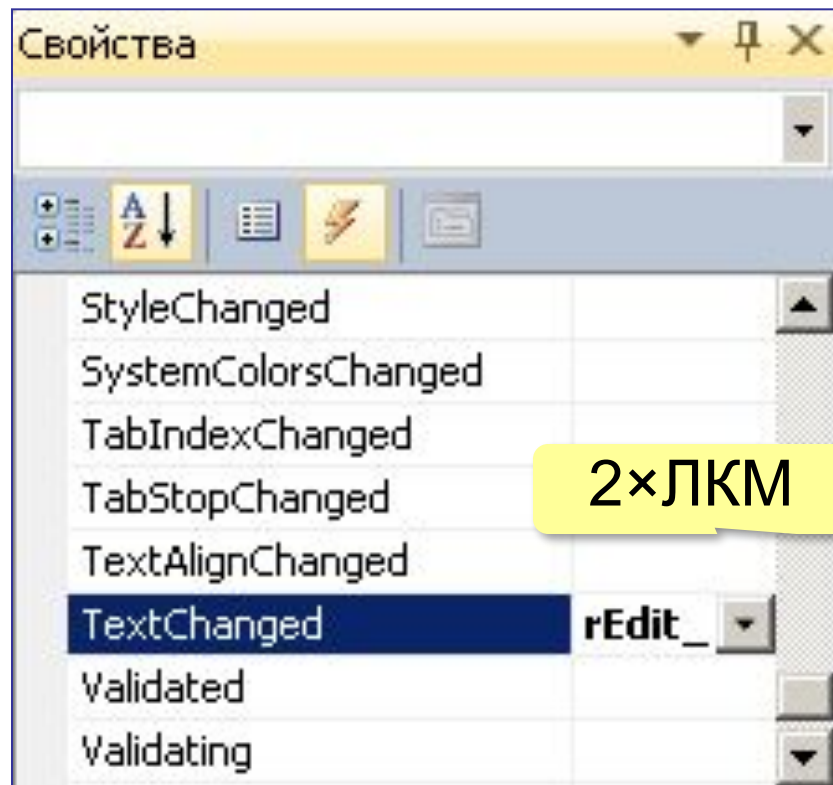
```
private void SizeCB_CheckedChanged(object sender, EventArgs e)
{
    |
}
```

```
if ( SizeCB.Checked )
    Img.SizeMode = PictureBoxSizeMode.Zoom;
else Img.SizeMode = PictureBoxSizeMode.Normal;
```

# Ввод и вывод данных



# Обновление компонентов вывода



```
private void rEdit_TextChanged(object sender, EventArgs e)
{
}
}
```

## Обновление компонентов вывода

```
private void rEdit_TextChanged (
    object sender, EventArgs e )
{
    int r, g, b;
    r = Int32.Parse ( rEdit.Text );
    g = Int32.Parse ( gEdit.Text );
    b = Int32.Parse ( bEdit.Text );
    rgbPanel.BackColor =
        Color.FromArgb ( r, g, b );
    rgbLabel.Text = "#" + r.ToString("X2")
        + g.ToString("X2") + b.ToString("X2");
}
```

из строки в число

построить  
цвет

в шестнадцатеричную  
систему, 2 знака

# Вызов при запуске

```
private void MainForm_Load (
    object sender, EventArgs e )
{
    rEdit_TextChanged ( rEdit, e );
}
```

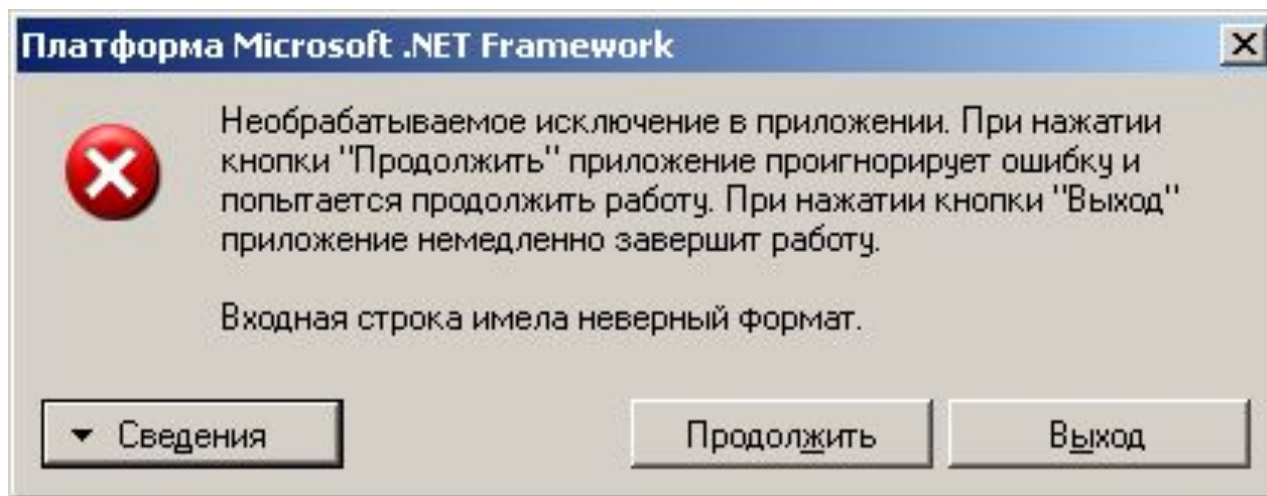
ВЫЗЫВАЮЩИЙ  
объект – **rEdit**  
(здесь – всё равно!)

```
private void MainForm_Load (
    object sender, EventArgs e )
{
    rEdit_TextChanged ( null, null );
}
```

пустой объект

# Обработка ошибок

**?** Если вместо числа ввести букву?



**!** Программа не должна «вылетать»!



# Обработка ошибок

попытаться выполнить

```
try
{
    // «опасные» команды
}
catch
{
    // обработка ошибки
}
```

если **исключение**  
(аварийная ситуация)



Какие у нас опасные операции?



# Обработка ошибок

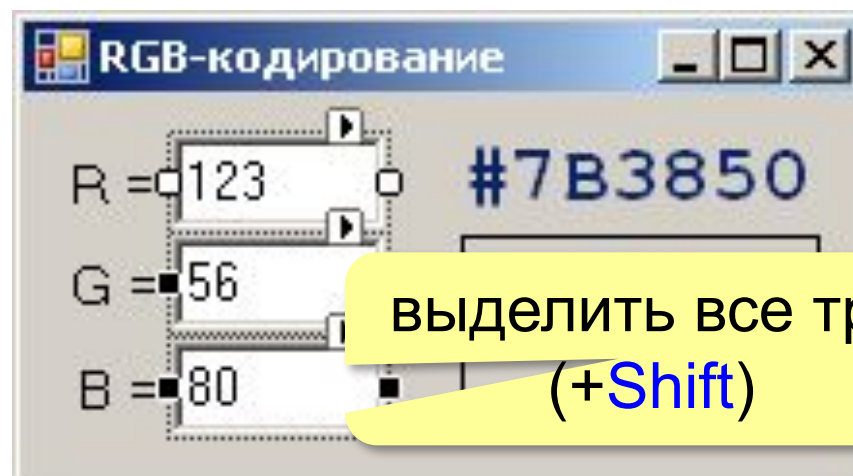
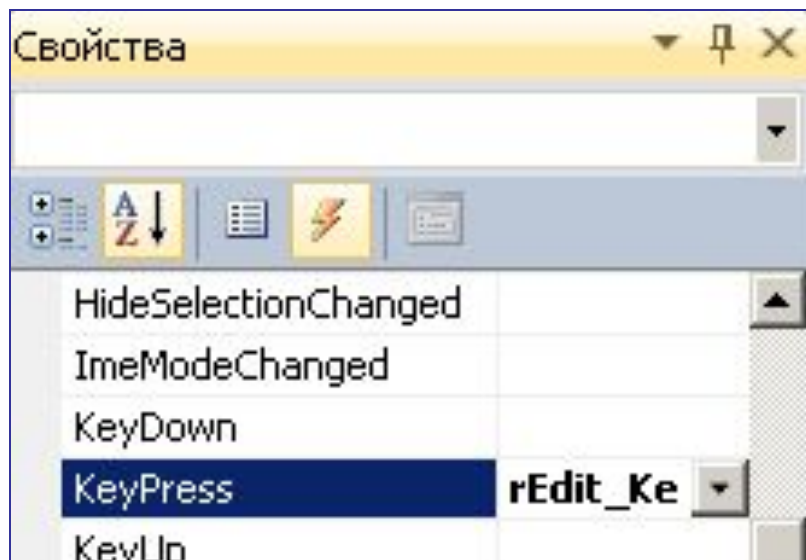
```
try {  
    r = Int32.Parse ( rEdit.Text );  
    g = Int32.Parse ( gEdit.Text );  
    b = Int32.Parse ( bEdit.Text );  
    rgbPanel.BackColor =  
        Color.FromArgb ( r, g, b );  
    rgbLabel.Text = "#" + r.ToString("x2")  
        + g.ToString("x2") + b.ToString("x2");  
}  
catch {  
    rgbLabel.Text = "?";  
}
```

если ошибка, записать "?"



Что делать, если ошибка?

# Блокирование неверных символов



```
private void rEdit_KeyPress (
    object sender, KeyPressEventArgs e )
{
    if ( ! ( Char.IsDigit(e.KeyChar) ||
        e.KeyChar == (char) 8) )
        e.Handled = true;
}
```

это цифра

Backspace

обработка завершена

# Объектно-ориентированное программирование. Языки C++ и C#

## **§ 54. Совершенствование компонентов**

# Новый класс (модуль)

*Задача:* построить поле для ввода целых чисел, в котором

- есть защита от ввода неверных символов
- есть методы для чтения/записи целого числа



На основе класса **TextBox**!

*Проект – Добавить класс*

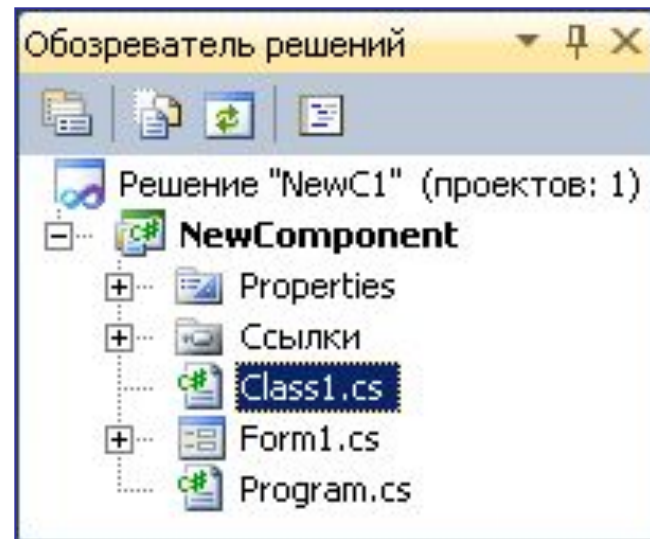
```
using System.Windows.Forms;
```

там объявлен  
**TextBox**

```
class IntTextBox: TextBox  
{  
}
```



F5 – запуск! И компонент в палитре!



# Обработчик `KeyPress`

```
class IntTextBox: TextBox
```

```
{
```

только для  
наследников

перекрыть метод  
базового класса

```
    protected override void OnKeyPress (  
        KeyPressEventArgs e )
```

```
{
```

```
    if ( !(Char.IsDigit(e.KeyChar) ||  
        e.KeyChar == (char)8) )
```

```
        e.Handled = true;
```

```
        base.OnKeyPress(e);
```

```
}
```

```
}
```

вызвать метод  
базового класса

## СВОЙСТВО Value

```
class IntTextBox: TextBox
{
    ...
    public int Value
    {
        set { Text = value.ToString(); }
        get {
            try { return Int32.Parse(Text); }
            catch { return 0; }
        }
    }
}
```

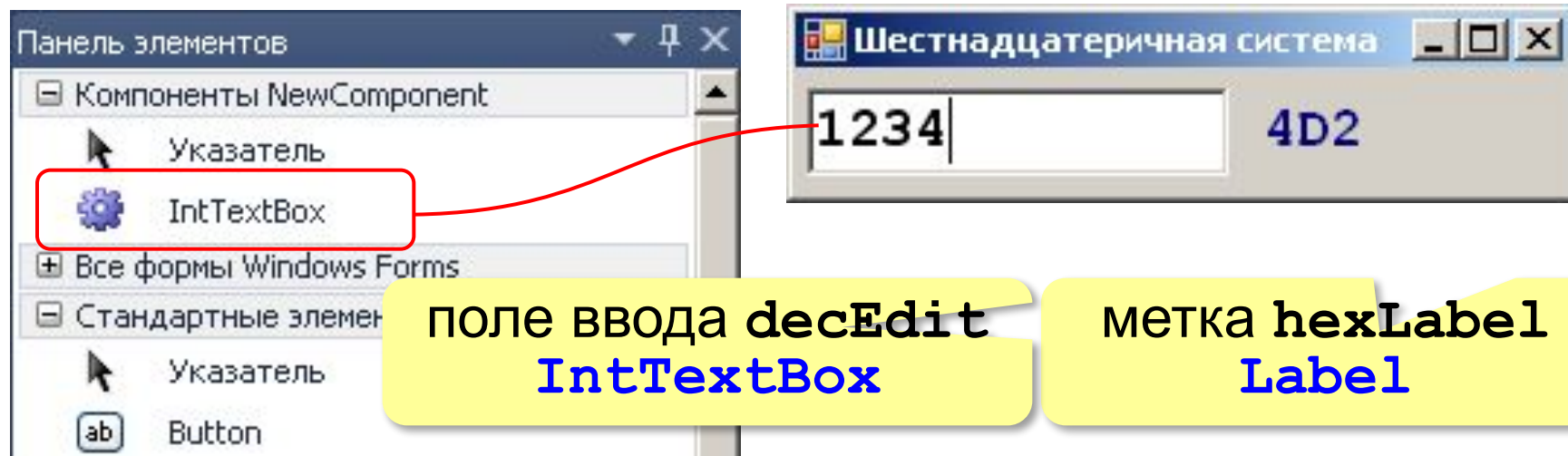
общедоступное  
СВОЙСТВО

число в строку

из строки в  
число

# Поле для ввода целых чисел

## Использование:



```
private void decEdit_TextChanged (
    object sender, EventArgs e )
{
    hexLabel.Text =
        decEdit.Value.ToString ( "X" );
}
```

СВОЙСТВО

в шестнадцатеричную

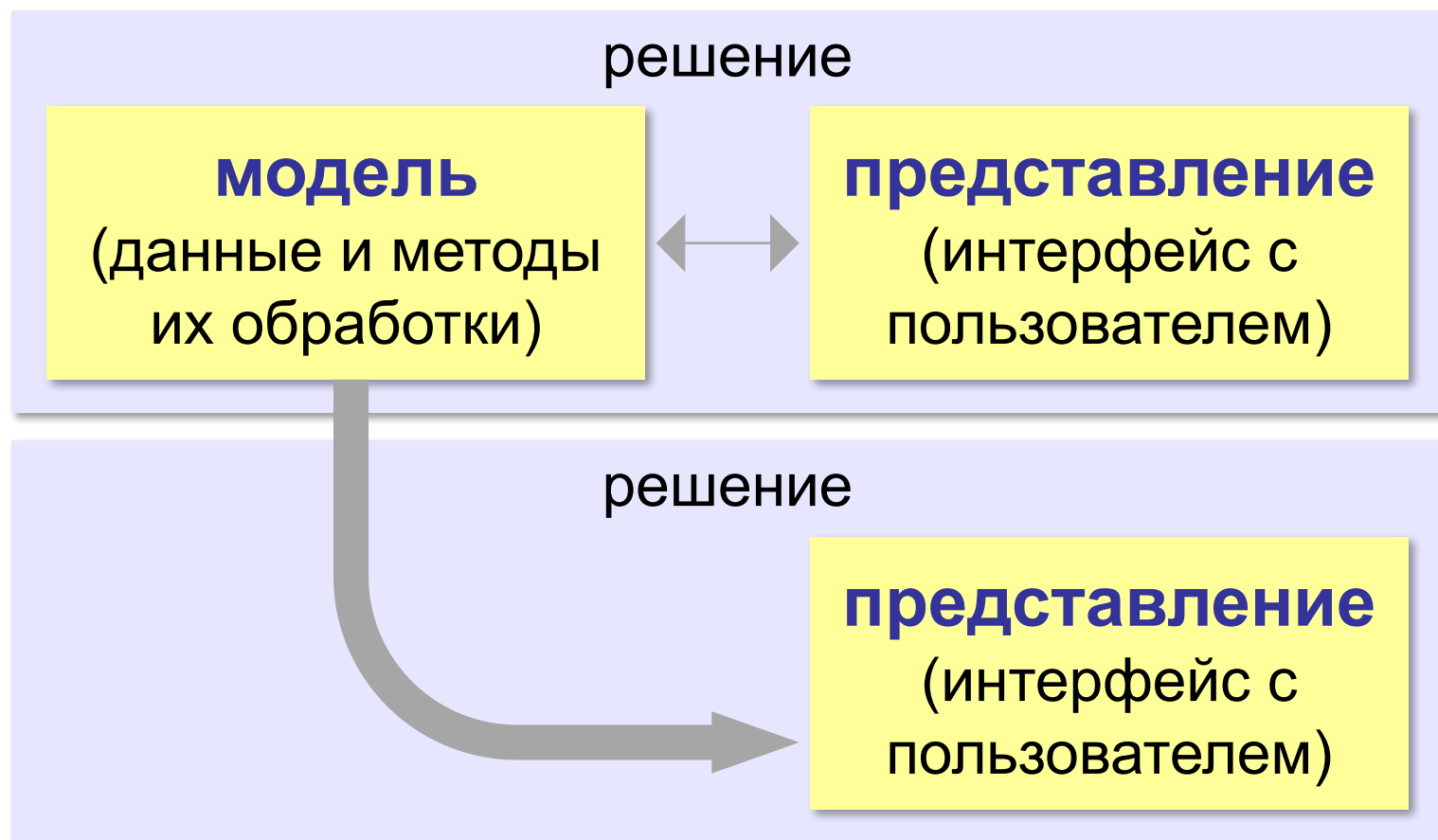
# Объектно-ориентированное программирование. Языки C++ и C#

## **§ 55. Модель и представление**



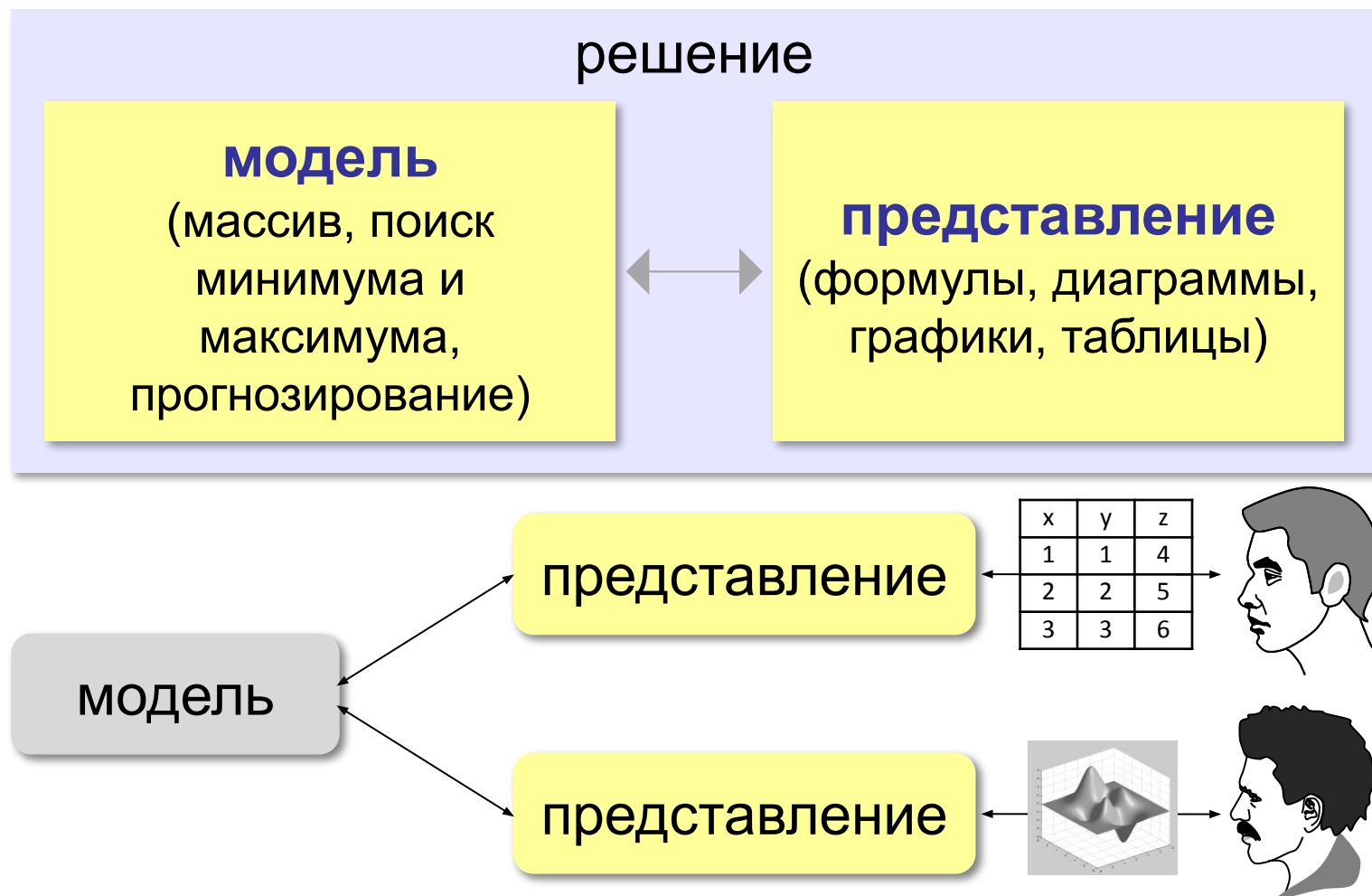
## Еще одна декомпозиция

**Задача:** повторное использование написанного ранее готового кода.



# Модель и представление

Задача: хранить и использовать данные об изменении курса доллара.



# Модель и представление

**Задача:** вычисление арифметического выражения:

- целые числа
- знаки арифметических действий  $+$   $-$   $*$   $/$

**Модель:**

- символьная строка
- алгоритм вычисления:

функция `LastOp`  
(глава 6)

$k$  = номер последней операции

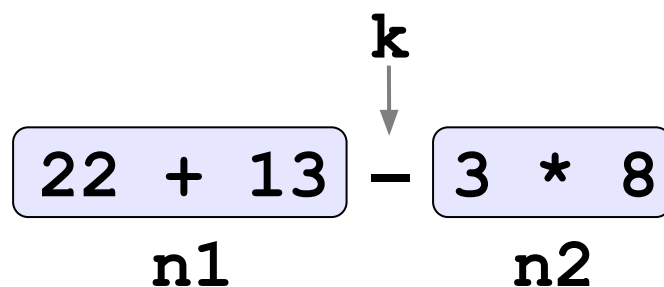
$n1$  = значение левой части

$n2$  = значение правой части

результат = операция( $n1$ ,  $n2$ )



Рекурсия!



Чего не хватает?

# Модель

---

## Псевдокод:

```
k = номер последней операции
if ( k < 0 )
    результат := строка в число
else {
    n1 = значение левой части
    n2 = значение правой части
    результат = операция(n1, n2)
}
```

# Статический класс – набор функций

Проект – Добавить класс

```
static class Calculator
{
    static int Priority ( char op ) {
        ...
    }
    static int LastOp ( string s ) {
        ...
    }
    public static int Calc ( string s ) {
        ...
    }
}
```

приоритет операции

последняя операция

ВЫЧИСЛИТЬ

! **Calc** – открытый метод!

# Модель: приоритет операций

---

```
int Priority ( char op )
{
    switch ( op )
    {
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 2;
    }
    return 100;
}
```

## Модель: номер последней операции

```
int LastOp ( string s )  
{  
    int i, minPrt, res;  
    minPrt = 50; // любое между 2 и 100  
    res = -1;  
    for ( i = 0; i < s.Length; i++ )  
        if ( Priority(s[i]) <= minPrt )  
        {  
            minPrt = Priority(s[i]);  
            res = i;  
        }  
    return res;  
}
```

вернёт номер  
СИМВОЛА

?

Почему <=?

## Модель: ВЫЧИСЛЕНИЯ

```
public static int Calc(string s)
{
    int k, n1, n2, res = 0;
    k = LastOp ( s );
    if ( k < 0 ) return Int32.Parse(s);
    n1 = Calc( s.Substring(0, k) ); // левая
    n2 = Calc( s.Substring(k+1) ); // правая
    switch ( s[k] ) {
        case '+': res = n1 + n2; break;
        case '-': res = n1 - n2; break;
        case '*': res = n1 * n2; break;
        case '/': res = n1 / n2; break;
    }
    return res;
}
```



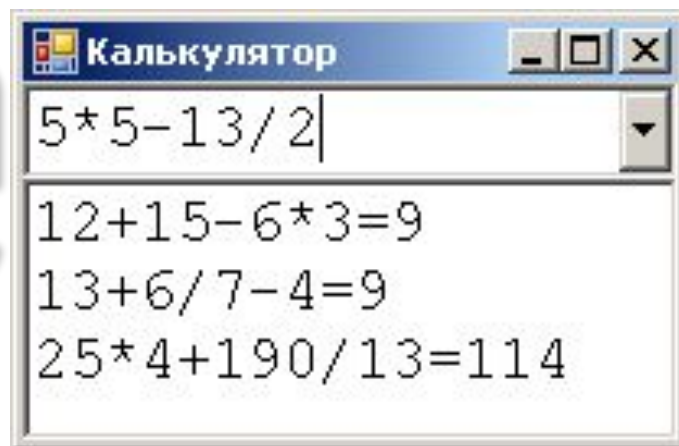
# Представление

выпадающий  
СПИСОК

**ComboBox**

**Name = Input**

**Dock = Top**



многострочное  
поле **TextBox**

**Name = Answers**

**Dock = Fill**

**ReadOnly = True**

**Multiline = True**

```
if ( нажата клавиша Enter )  
{  
    x = значение выражения  
    добавить результат в конец поля вывода  
    if ( выражения нет в списке )  
        добавить его в список  
}
```

# Перехват нажатия на клавишу **Enter**

**KeyPress** для элемента **Input**:

```
private void Input_KeyPress (
    object sender, KeyPressEventArgs e )
{
    if ( e.KeyChar == (char)13 )
    {
        ...
    }
}
```

КОД КЛАВИШИ  
**Enter**

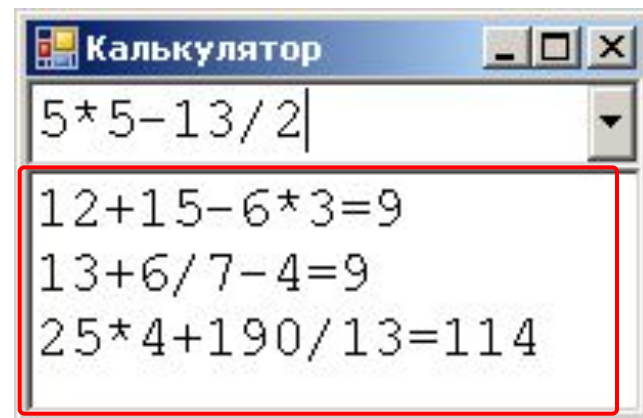
# Обработка и вывод данных

**Вычисления** (обращение к модели):

```
int x;  
x = Calculator.Calc(  
    Input.Text);
```

**Добавление строки в TextBox:** →

добавить строку



```
Answers.Text += Input.Text + "=" +  
    x.ToString() + "\r\n";
```

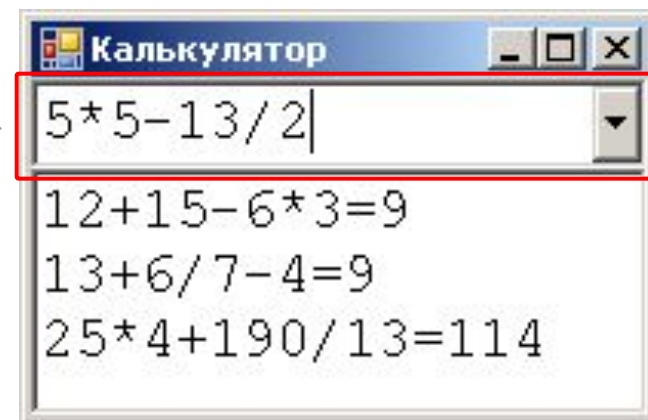
число в строку

новая строка

# Обработка и вывод данных

## Добавление строки в ComboBox:

найти индекс строки



```
int i = Input.FindString(Input.Text);  
if ( i < 0 )  
    Input.Items.Insert ( 0, Input.Text );
```

массив строк в  
**ComboBox**

вставить  
строку

позиция  
списка

что  
вставлять

## Перехват нажатия на клавишу Enter

KeyPress для элемента Input:

```
private void Input_KeyPress (
    object sender, KeyPressEventArgs e )
{
    if ( e.KeyChar == (char)13 )
    {
        int x = Calculator.Calc ( Input.Text );
        Answers.Text += Input.Text + "="
                        + x.ToString() + "\r\n";
        int i = Input.FindString(Input.Text);
        if ( i < 0 )
            Input.Items.Insert(0, Input.Text);
    }
}
```

# Калькулятор



Самостоятельно!

# Конец фильма

---

**ПОЛЯКОВ Константин Юрьевич**

д.т.н., учитель информатики

ГБОУ СОШ № 163, г. Санкт-Петербург

[kpolyakov@mail.ru](mailto:kpolyakov@mail.ru)

**ЕРЕМИН Евгений Александрович**

к.ф.-м.н., доцент кафедры мультимедийной

дидактики и ИТО ПГГПУ, г. Пермь

[eremin@pspu.ac.ru](mailto:eremin@pspu.ac.ru)

# Источники иллюстраций

---

1. [www.picstopin.com](http://www.picstopin.com)
2. [maugav.info](http://maugav.info)
3. [yoursourceisopen.com](http://yoursourceisopen.com)
4. [ru.wikipedia.org](http://ru.wikipedia.org)
5. иллюстрации художников издательства «Бином»
6. авторские материалы