# Объектно-ориентированное программирование

§ 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

## § 46. Что такое ООП?

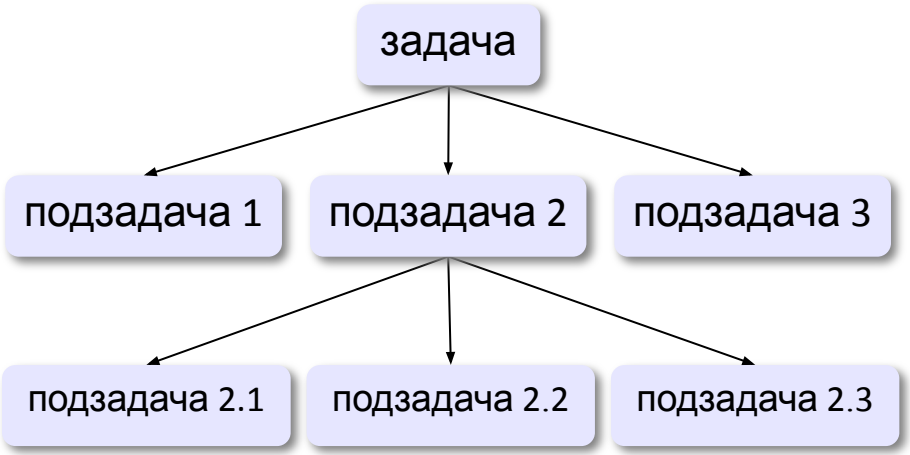
# Зачем нужно что-то новое?

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**»».

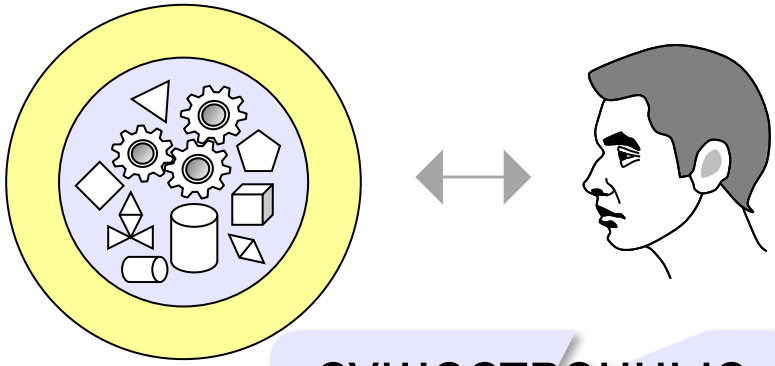
**Структурное программирование**:




**декомпозиция** по задачам

человек мыслит иначе, объектами

# Как мы воспринимаем объекты?



существенные свойства

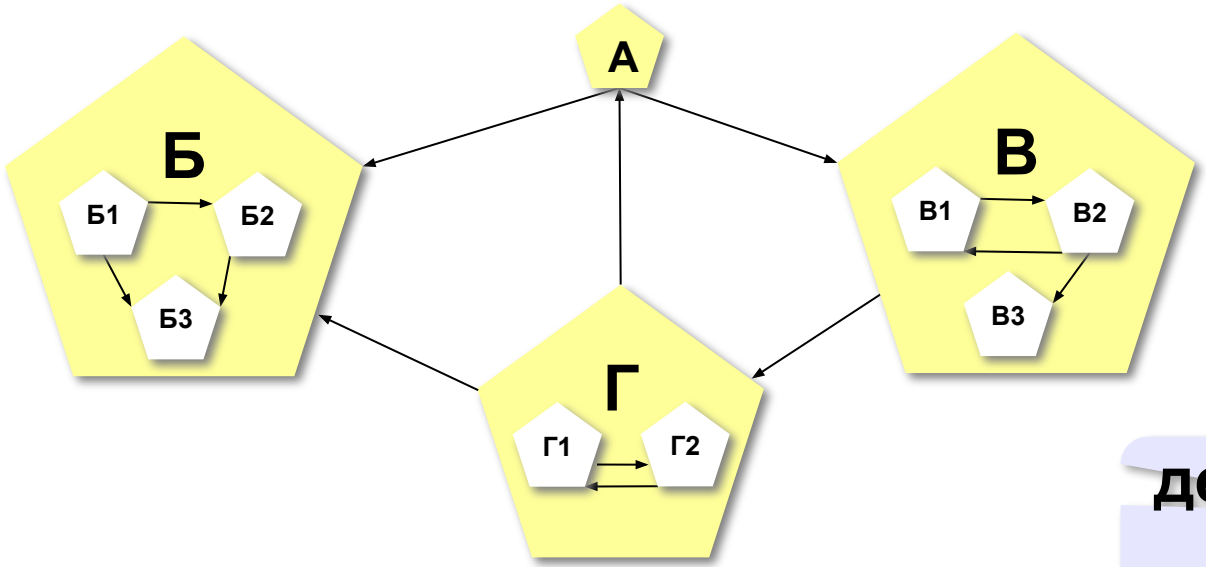
**Абстракция** – это выделение существенных свойств объекта, отличающих его от других объектов.

! 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

## § 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*.

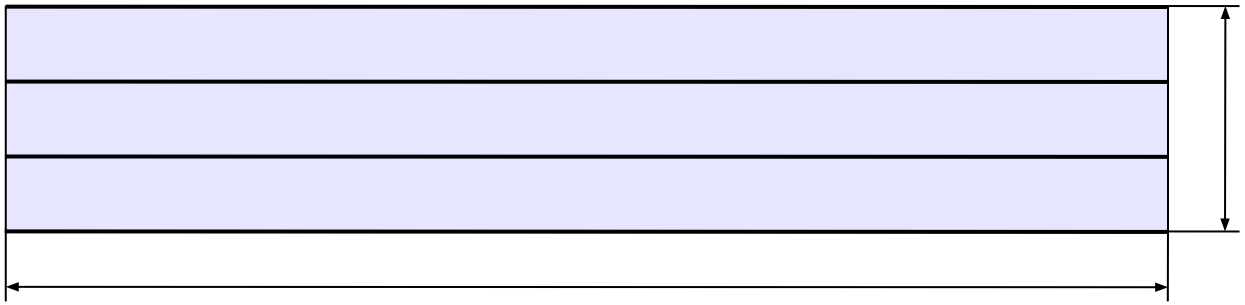
**Состояние определяет поведение:**

- лежачий 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* |
| --- |
| длина<br>ширина |
| |

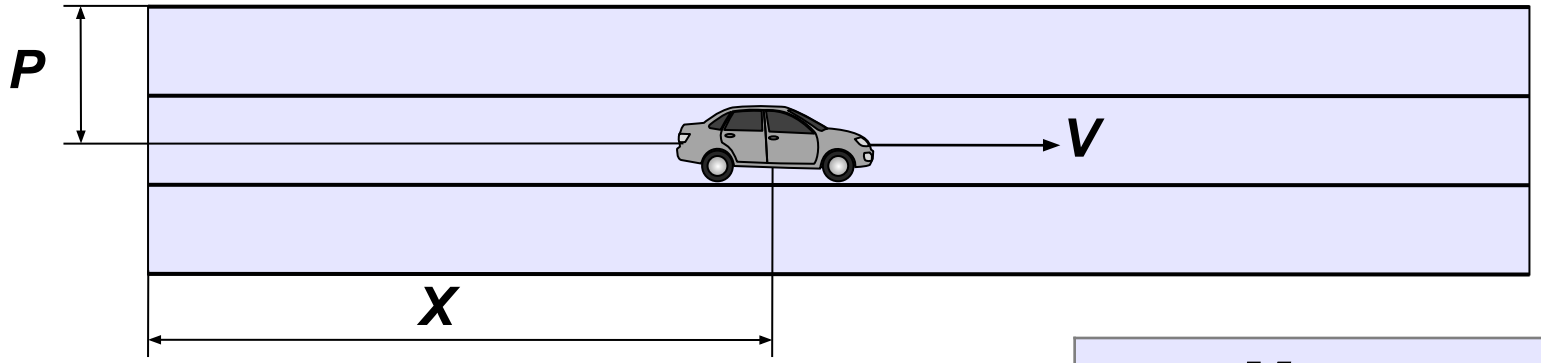
**свойства** (состояние)

**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

| *Машина* |
|---|
| X (координата)<br>P (полоса)<br>V (скорость) |
| двигаться |

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

# Модель дороги с автомобилями

## Взаимодействие объектов:

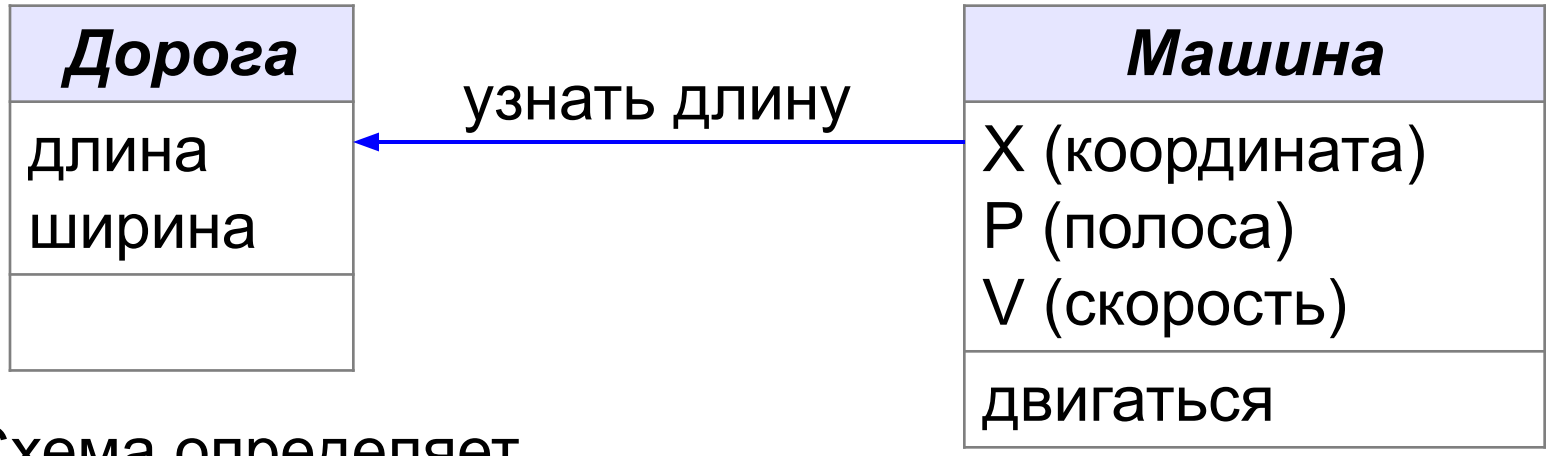



Схема определяет

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

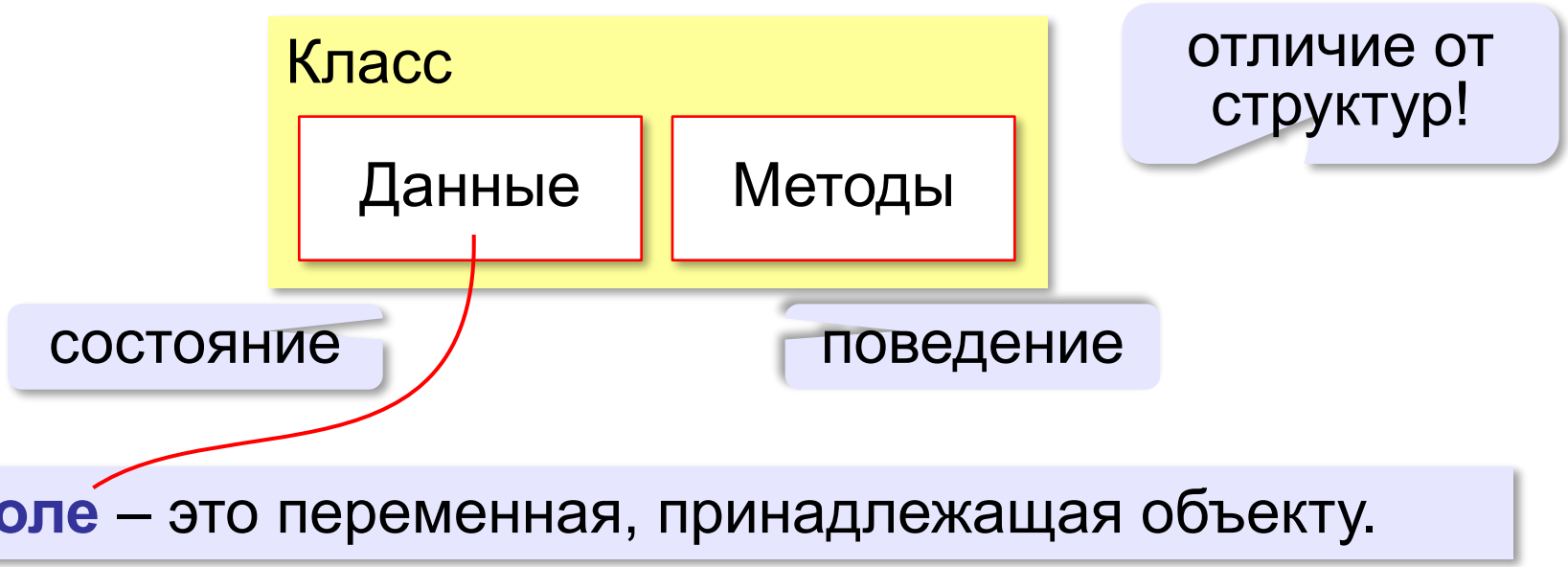
! Ни слова о внутреннем устройстве объектов!

# Объектно-ориентированное программирование

## § 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» *(FreePascal)*

**Объявление класса**:

```
type
  TRoad = class
    Length: real;
    Width: integer
  end;
```

Память не выделяется!

**Объявление переменной**:

```
var road: TRoad;
```

Это только указатель!

**Создание объекта в памяти**:

```
road:= TRoad.Create;
```

вызов конструктора

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

# Класс «Дорога» *(FreePascal)*

**Простая программа**:

```
{$mode objfpc}
type
  TRoad = class
    Length: real;
    Width: integer;
  end;
var road: TRoad;
begin
  road:=TRoad.Create;
  road.Length:=60;
  road.Width:=3
end.
```

режим работы с объектами

изменение данных (полей)

# Новый конструктор

```
type
  TRoad = class
    Length: real;
    Width: integer;
    constructor Create(length0: real;
                       width0: integer)
  end;
```

параметры

**Использование**:

```
road:= TRoad.Create(60, 3);
```

длина (length0)

ширина (width0)

? Что будет делать конструктор?

# Новый конструктор

```
constructor TRoad.Create(length0: real;
                         width0: integer);
begin
  if length0 > 0 then
      Length:= length0
  else Length:= 1;
  if width0 > 0 then
      Width:= width0
  else Width:= 1
end;
```

защита от неверных данных

# Класс «Машина»

```
type
  TCar = class
    X, V: real;
    P: integer;
    road: TRoad;
    procedure move;
    constructor Create(road0: TRoad;
                p0: integer; v0: real)
  end;
```

координата, скорость

полоса

дорога, по которой едет

# Конструктор класса «Машина»

```
constructor TCar.Create(road0: TRoad;
                        p0: integer; v0: real);
begin
   road:=road0;
   P:=p0;
   V:=v0
end;
```

защита от ошибок – самостоятельно

# Класс «Машина»: метод move

**Равномерное движение**:

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

$\Delta t = 1$ интервал дискретизации

перемещение за одну единицу времени

```
procedure TCar.move;
begin
  X:=X+V;
  if X>road.Length then X:=0
end;
```

выезжает с другой стороны

# Основная программа

```
{ описание классов }
var road: TRoad;
    car: TCar;
    i: integer;
begin
  road := TRoad.Create( 65, 1 );
  car := TCar.Create( road, 1, 10 );
  car.move;
  writeln( 'После 1 шага:' );
  writeln( car.X );
  for i:=1 to 10 do begin
    car.move;
    writeln( car.X );
  end;
end.
```

? Что выведет?

```
10
10
20
30
40
50
60
0
10
20
30
40
```

дошли до конца дороги

# Много машин

```
const N = 3;
var road: TRoad;
    cars: array [1..N] of TCar;
    i: integer;
begin
  road:= TRoad.Create(60, N);
  for i:=1 to N do
    cars[i]:= TCar.Create(road, i, 2.0*i);
  for i:=1 to N do
    cars[i].move;
end.
```

# 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

# Задание

**«A»:** Построить класс Попугай (**TParrot**), который умеет говорить какую-то фразу, заранее определённую при описании класса.

**Пример**:

```
p := TParrot.Create;
p.Say;              Привет, друзья!
```

**«B»:** Изменить класс из задания A так, чтобы фраза задавалась при создании конкретного экземпляра.

**Пример**:

```
p1 := TParrot.Create( "Гав!" )
p2 := TParrot.Create( "Мяу!" )
p1.Say;              Гав!
p2.Say;              Мяу!
```

# Задание

**«C»:** Изменить класс из задания B так, чтобы фразу можно было изменять во время работы программы.

**Пример**:

```
p := TParrot.Create( "Гав!" );
p.Say;                    Гав!
p.NewText( "Мяу!" );
p.Say;                   Мяу!
```

**«D»:** Изменить класс из задания C так, чтобы при вызове метода `say` можно было задать число повторений.

**Пример**:

```
p := TParrot.Create( "Гав!" );
p.Say( 1 );                Гав!
p.NewText( "Мяу!" );
p.Say( 3 );                Мяу! Мяу! Мяу!
```

# Задание

**«E»:** Изменить класс из задания D так, чтобы можно было добавлять фразы в набор фраз, которые знает попугай. При вызове метода **say** попугай выдаёт случайную фразу из своего набора.

**Пример**:

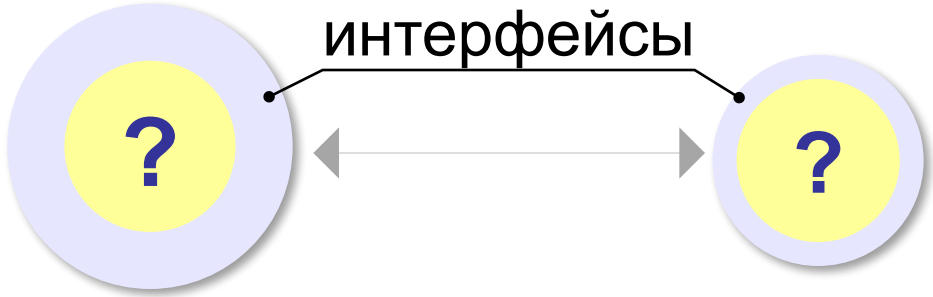
```
p := TParrot.Create( "Гав!" );
p.Say( 1 );                  Гав!
p.learn( "Мяу!" )
p.Say( 1 );                  Гав!
p.Say( 3 );                  Мяу! Мяу! Мяу!
```

# Объектно-ориентированное программирование

## § 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») – скрытие внутреннего устройства объектов.

Также объединение данных и методов в одном объекте!

# Защита внутренних данных

**Cat**

состояние
- энергия
- настроение
- голод

методы

**?** Можно изменять вучную?

```
метод есть
  + энергия
  + настроение
  - голод
```

```
метод спать
  + энергия
  + голод
```

```
метод играть
  - энергия
  + настроение
  + голод
```

**!** Меняем состояние только через методы!

# Пример: класс «перо»

```
type
  TPen = class
    color: string     { цвет, 'FF00FF'}
  end;
```

R G B

По умолчанию все члены класса открытые (общедоступные) – **public**!

```
type
  TPen = class
  private             { частные (закрытые) }
    FColor: string
  end;
```

*Field* – поле

Как к ним обращаться?

# Пример: класс «перо»

```
type
  TPen = class
  private              { частные (закрытые) }
    FColor: string;
  public    { общедоступные (открытые) }
    function getColor: string;
    procedure setColor(newColor: string)
  end;
```

методы доступа к закрытому полю

**Получить значение**:

член класса **TPen**

```
function TPen.getColor: string;
begin
  Result:=FColor
end;
```

# Пример: класс «перо»

**Записать значение**:

```
procedure TPen.setColor(newColor: string);
begin
  if Length(newColor) <> 6 then
      FColor:='000000'
  else FColor:=newColor
end;
```

если ошибка, чёрный цвет

Защита от неверных данных!

# Пример: класс «перо»

**Использование**:

```
var pen: TPen;
...
pen.setColor( 'FFFF00' );
writeln( 'цвет пера: ',
         pen.getColor );
```

установить цвет

прочитать цвет

Не очень удобно!

# Свойство

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

```
type
  TPen = class
  private
    FColor: string;
    function getColor: string;
    procedure setColor(newColor: string);
  public
    property color: string read getColor
                           write setColor
  end;
```

скрытые методы!

метод чтения

свойство

метод записи

# Свойство: использование

```
property color: string read getColor
                        write setColor;
```

вызов `TPen.setColor`

```
pen.color := 'FFFF00';
writeln( 'цвет пера: ', pen.color );
```

вызов `TPen.getColor`

Такое же обращение, как к полю!

# Свойство: прямой доступ к полю

```
function TPen.getColor: string;
begin
  Result:= FColor
end;
```

```
type
  TPen = class
  private
    FColor: string;
    procedure setColor(newColor: string);
  public
    property color: string read FColor
                           write setColor
  end;
```

# Изменение внутреннего устройства

**Удобнее хранить цвет в виде числа**:

```
type
  TPen = class
  private
    FColor: integer;
    ...
  public
    property color: string read getColor
                           write setColor
  end;
```

изменилось внутреннее устройство

! Интерфейс не изменился!

# Изменение внутреннего устройства

```
function TPen.getColor: string;
begin
  Result:= IntToHex(FColor, 6)
end;
```

6 знаков

в шестнадцатеричную систему

```
procedure TPen.setColor(newColor: string);
begin
  if Length(newColor) <> 6 then
    FColor:= 0   { если ошибка, то чёрный }
  else
    FColor:= StrToInt('$' + newColor)
end;
```

из строки в число

признак шестнадцатеричной системы

Данные – целое число, свойство – строка!

# Свойство «только для чтения»

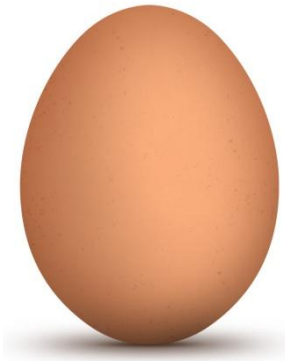
**Скорость машины** можно только читать:

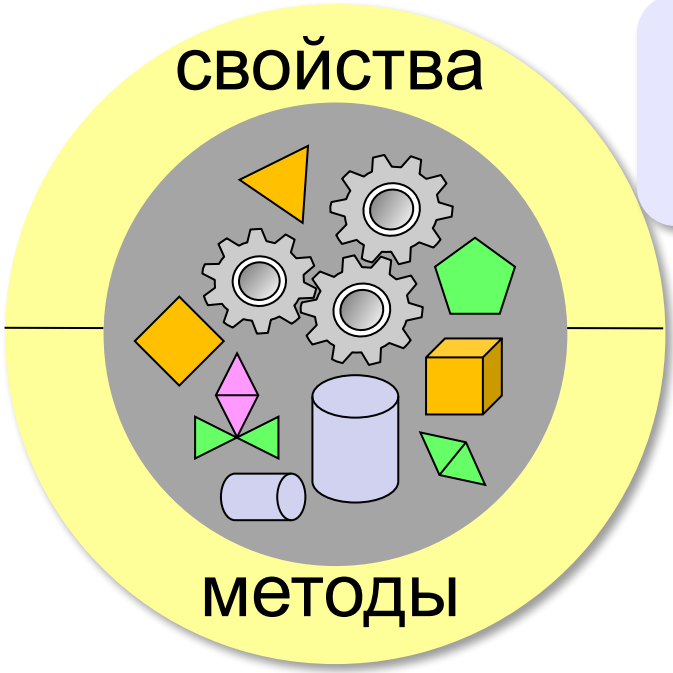
```
type
  TCar = class
  private
    Fv: real;
    ...
  public
    property v: real read Fv;
    ...
  end;
```

нет возможности записи

# Скрытие внутреннего устройства

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






внутреннее устройство (`private`)

интерфейс (`public`)

# Задание

**«А»:** Построить класс РядЛампочек (**TLampRow**), который хранит состояние ряда из 8 лампочек в виде символьной строки. Цифра 0 обозначает выключенную лампочку, цифра 1 – включенную.

Методы **getState** и **setState** скрывают внутреннюю переменную **state**, которая хранит состояние лампочек. При записи нового значения проверяется, что длина строки состояния равна 8, иначе записываются все нули.

Метод **Show** выводит на экран состояние лампочек, обозначая выключенную лампочку как минус, а включённую – как «*».

**Пример:**

```
lamps := TLampRow.Create;
lamps.Show;                  --------
lamps.setState( '10101010' );
writeln( lamps.getState );     10101010
    lamps.Show;                        *-*-*-*-
```

# Задание

**«B»:** Дополните класс **TLampRow** из задания A так, чтобы количество лампочек в цепочке можно было задавать в конструкторе.

**Пример**:

```
lamps := TLampRow.Create( 6 );
lamps.Show;                  ------
lamps.setState( '101010' );
writeln( lamps.getState );     101010 lamps.Show;
         *-*-*-
lamps.setState( '10101010' ); ошибка
writeln( lamps.getState );     000000 lamps.Show;
         ------
```

# Задание

**«C»:** Дополните **TLampRow** из задания B так, чтобы лампочки могли гореть одним из двух цветов – красный цвет имеет код 1 и обозначается при выводе как «*», а зелёный цвет имеет код 2 и обозначается как «o».

**Пример:**

```
lamps := TLampRow.Create( 6 );
lamps.Show;                  ------
lamps.setState( "102102" );
writeln( lamps.setState );     102102 lamps.Show;
         *-o*-o
lamps.setState( "10201010" ); ошибка
writeln( lamps.getState );     000000
lamps.Show;                    ------
```

# Задание

**«D»:** Дополните **TLampRow** из задания C так, чтобы код состояния хранился как целое число. При этом интерфейс (способ вызова методов **getState** и **setState**) не должен измениться.

**Пример**:

```
lamps := TLampRow.Create( 6 );
lamps.Show;                  ------
lamps.setState( "102102" );
writeln( lamps.setState );     102102 lamps.Show;
         *-o*-o*
lamps.setState( "10201010" ); ошибка
writeln( lamps.getState );     000000
lamps.Show;                    ------
```

# Объектно-ориентированное программирование

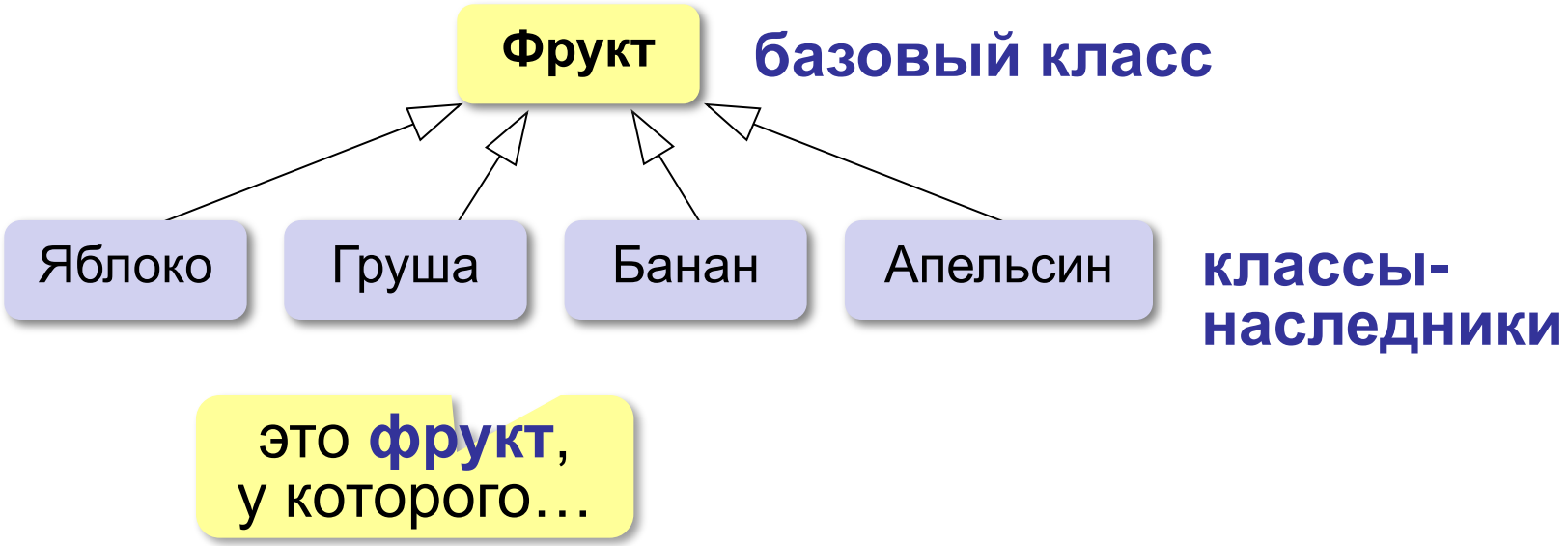
## § 50. Иерархия классов

# 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.

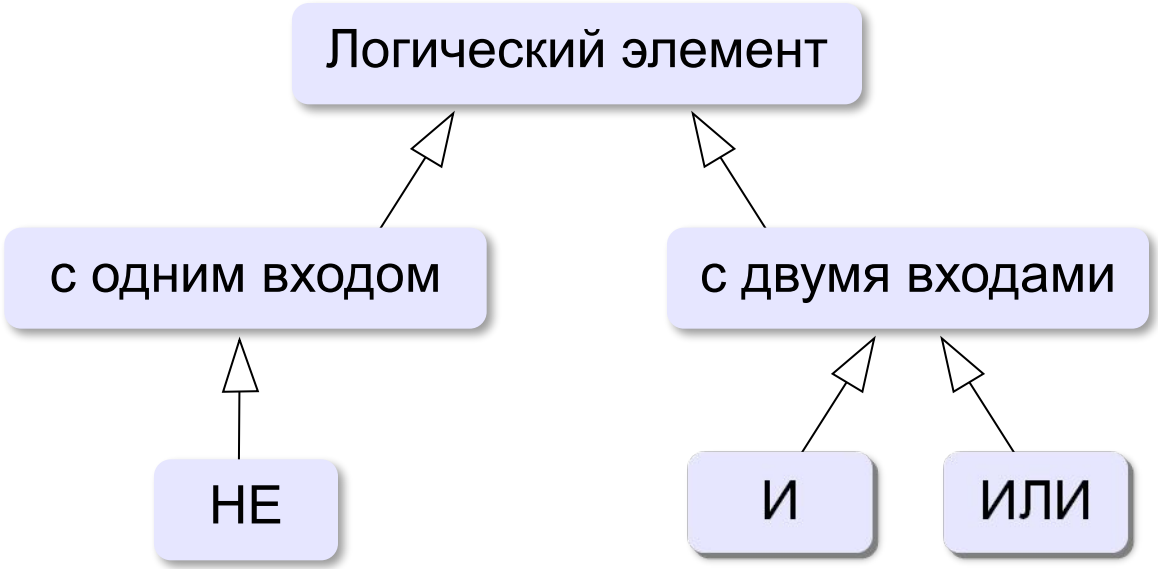
✓ яблоко – фрукт
✓ горный клевер – клевер
✗ машина – двигатель

яблоко – **это** фрукт

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

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)<br>In2 (вход 2)<br>Res (результат) |
| calc |

```
type
  TLogElement = class
    In1, In2: boolean;
    Res: boolean;
    procedure calc
  end;
```

вычислить выход

? Зачем хранить результат?

можно моделировать элементы с памятью (триггеры)

? Что плохо?

# Базовый класс

```
type
  TLogElement = class
  private
    FIn1, FIn2: boolean;
    FRes: boolean;
    procedure setIn1(newIn1: boolean);
    procedure setIn2(newIn2: boolean);
    procedure calc;
  public
    property In1: boolean read FIn1
                          write setIn1;
    property In2: boolean read FIn2
                          write setIn2;
    property Res: boolean read FRes
  end;
```

? Почему здесь?

только для чтения

# Установка входа

```
procedure TLogElement.setIn1(newIn1:
                                    boolean);
begin
   FIn1:= newIn1;
   calc
end;
```

? Что ещё?

пересчёт при изменении входа

? Как написать процедуру `calc`?

```
procedure TLogElement.calc;
begin
end;
```

заглушка

! Проблема: наследники должны менять `calc`!

# Что такое полиморфизм?

греч.: $\pi o \lambda \upsilon$ — много, $\mu o \rho \varphi \eta$ — 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!

```
type
  TLogElement = class
  protected
    procedure calc;
    ...
  end;
```

защищённые элементы: доступны только наследникам

# Базовый класс

```
type
  TLogElement = class
  private
    FIn1, FIn2: boolean;
    procedure setIn1(newIn1: boolean);
    procedure setIn2(newIn2: boolean);
  protected
    FRes: boolean;
    procedure calc; virtual; abstract;
    property In2: boolean read FIn2
                          write setIn2;
  public
    property In1: boolean read FIn1
                          write setIn1;
    property Res: boolean read FRes
  end;
```

наследники будут изменять поле

# Базовый класс

```
type
  TLogElement = class
    ...
  protected
    FRes: boolean;
    procedure calc; virtual; abstract;
    property In2: boolean read FIn2
                          write setIn2;
    ...
  end;
```

наследники будут изменять поле

для элементов с одним входом не нужно!

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

**abstract** (абстрактный) – 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**

```
type
  TNot = class(TLogElement)
  protected
    procedure calc; override;
  end;
```

переопределяет метод базового класса

```
procedure TNot.calc;
begin
  FRes:= not In1
end;
```

? Почему не **not FIn1**?

! Это уже не абстрактный класс!

# Элемент «НЕ»

**Использование**:

```
var n: TNot;
...
n:=TNot.Create;
n.In1:=False;
writeln(n.Res);
```

объявление указателя

создание

установка входа

вывод результата

# Элементы с двумя входами

наследник от TLogElement

```
type
  TLog2In = class(TLogElement)
  public
    property In2
  end;
```

повысить «видимость» свойства

? Можно ли создать объект этого класса?

нельзя, он абстрактный

# Элементы с двумя входами

```
type
  TAnd = class(TLog2In)
  protected
    procedure calc; override;
  end;

  TOr = class(TLog2In)
  protected
    procedure calc; override;
  end;
```

элемент «И»

элемент «ИЛИ»

# Элементы с двумя входами

```
procedure TAnd.calc;
begin
  FRes:= In1 and In2
end;

procedure TOr.calc;
begin
  FRes:= In1 or In2
end;
```

элемент «И»

элемент «ИЛИ»

доступ к защищённому полю (**protected**)

# Вызов виртуального метода

## В базовом классе:

```
procedure TLogElement.setIn1(newIn1:
                                    boolean);
begin
   FIn1 := newIn1;
   calc
end;
```


Какой метод вызывается?

```
type
  TLogElement = class
    ...
  protected
    procedure calc; virtual; abstract;
    ...
  end;
```

# Виртуальный метод

**Статическое связывание**:

транслятор записывает в код адрес процедуры

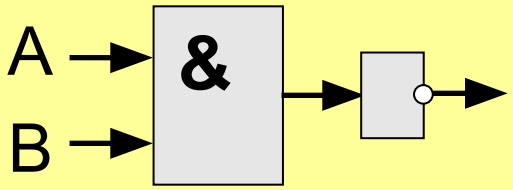
**Динамическое связывание**:

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»

```
var elNot: TNot;
    elAnd: TAnd;
    A, B: boolean;
begin
  elNot:=TNot.Create;
  elAnd:=TAnd.Create;
  writeln('| A | B | not(A&B) ');
  writeln('-------------------');
  for A:=False to True do begin
    elAnd.In1:=A;
    for B:=False to True do begin
      elAnd.In2:=B;
      elNot.In1:=elAnd.Res;
      writeln('| ', integer(A), ' | ', integer(B),
              ' | ', integer(elNot.Res))
    end
  end
end.
```



# Модульность

```
program logic;
{$mode objfpc}
uses log_elem;
var elNot: TNot;
    elAnd: TAnd;
    ...
begin
  elNot:=TNot.Create;
  elAnd:=TAnd.Create;
  ...
end.
```

```
unit log_elem;
{$mode objfpc}
interface
type
  TLogElement = class
    ...
  end;
  TNot = class(TLogElement)
    ...
  TLog2In = class(TLogElement)
    ...
implementation
  ...
procedure TOr.calc;
begin
  FRes:= FIn1 or FIn2;
end;
end.
```

# Сообщения между объектами

Задача – автоматическая передача сигналов по цепочке!

```
type
  TLogElement = class
  private
    FNextEl: TLogElement;
    FNextIn: integer;
    ...
  public
    procedure Link(nextElement: TLogElement;
                   nextIn: integer);
    ...
  end;
```

следующий элемент в цепочке

номер входа следующего элемента

# Сообщения между объектами

## Установка связи:

```
procedure TLogElement.Link(
                    nextElement: TLogElement;
                nextIn: integer);
begin
  FNextEl:=nextElement;
  FNextIn:=nextIn
end;
```

# Сообщения между объектами

**После изменения выхода «дергаем» следующий элемент**:

```
procedure TLogElement.setIn1(
                          newIn1: boolean);
begin
  FIn1:=newIn1;
  calc;
  if FNextEl<>nil then
    case FNextIn of
      1: FNextEl.In1:=res;
      2: FNextEl.In2:=res
    end
end;
```

если следующий элемент установлен…

передать результат на нужный вход

# Сообщения между объектами

## Изменения в основной программе:

```
elNot:= TNot.Create;
elAnd:= TAnd.Create;
elAnd.Link(elNot, 1);
...
for A:=False to True do begin
  elAnd.In1:= A;
  for B:=False to True do begin
    elAnd.In2:= B;
    elNot.In1:= elAnd.Res;
...
```

установить связь

это уже не нужно!

# Задание

**«A»:** Постройте класс **TPet** (домашнее животное) с двумя скрытыми полями: **FName** (имя) и **FAge** (возраст). Они должны быть доступны для чтения через свойства **name** и **age** и недоступны для записи. Метод **gettingOlder** увеличивает возраст на 1 год. Класс **TPet** – абстрактный, он имеет абстрактный метод **Say**.

Постройте два класса-наследника – **TCat** (кошка) и **TDog** (собака).Они должны реализовать метод **Say**.

Описания классов должны быть в отдельном модуле **animals.pas**.

**Пример**: см. следующий слайд.

# Задание

**«А»:**

**Пример:**

```
uses Animals;
var pets: array of TPet;
    p: TPet;
begin
  SetLength(pets, 2);
  pets[0] := new TCat('Мурзик', 3);
  p := new TDog('Шарик', 5);
  p.gettingOlder;
  writeln(p.Name, ': ', p.Age, ' лет');
  pets[1] := p;
  for var i:=0 to pets.Count-1 do
    pets[i].Say;
end.
```

```
Шарик: 6 лет
Мурка: Мяу!
Шарик: Гав!
```

# Задание

**«B»:** Добавьте класс **TMammal** (млекопитающее) – наследник класса **TPet** и предок для классов **TCat** и **TDog**. Он должен иметь метод **Run** (бежать), который выводит сообщение вида «Вася побежал».

**Пример**:

```
pets[0] := new TCat('Мурзик', 3);
pets[1] := new TDog('Шарик', 5);
for var i:=0 to pets.Count-1 do begin
  pets[i].Say;
  if pets[i] is TMammal then
    TMammal(pets[i]).Run;
end;
```

```
Мурзик: Мяу!
Мурзик побежал...
Шарик: Гав!
Шарик побежал...
```

# Задание

**«С»:** Добавьте класс **TReptilia** (рептилии) – наследник класса **TPet** и предок для новых классов **TTurtle** (черепаха) и **TSnake** (змея). Он должен иметь метод **Crawl** (ползти), который выводит сообщение вида «Вася пополз…».

**Пример:**

```
pets[0] := new TCat('Мурзик', 3);
pets[1] := new TTurtle('Зак', 32);
pets[2] := new TDog('Шарик', 5);
pets[3] := new TSnake('Чаки', 2);
for var i:=0 to pets.Count-1
  do begin
  pets[i].Say;
  if pets[i] is TMammal then
    TMammal(pets[i]).Run;
  if pets[i] is TReptilia then
    TReptilia(pets[i]).Crawl;
end;
```

```
Мурзик: Мяу!
Мурзик побежал...
Зак: ...
Зак пополз...
Шарик: Гав!
Шарик побежал...
Чаки: ш-ш-ш-ш...
Чаки пополз...
```

# Задание

**«А»:** Собрать полную программу и построить таблицу истинности последовательного соединения элементов «ИЛИ» и «НЕ».

**Пример:**

```
 A | B | not(A+B)
------------------
 0 | 0 | 1
 0 | 1 | 0
 1 | 0 | 0
 1 | 1 | 0
```

# Задание

**«B»:** Добавить в иерархию классов элементы «И-НЕ» (**TNAnd**) и «ИЛИ-НЕ» (**TNOr**), которые представляют собой последовательные соединения элементов «И» и «ИЛИ» с элементом «НЕ». Построить их таблицы истинности.

**Пример**:

```
 A | B | A nand B
-------------------
 0 | 0 | 1
 0 | 1 | 1
 1 | 0 | 1
 1 | 1 | 0
 A | B | A nor B
-------------------
 0 | 0 | 1
 0 | 1 | 0
 1 | 0 | 0
 1 | 1 | 0
```

# Задание

**«С»:** Добавить в иерархию классов элемент «исключающее ИЛИ» (**TXor**) и «импликация» (**TImp**). Построить их таблицы истинности.

**Пример**:

```
 A | B | A xor B
-------------------
 0 | 0 | 0
 0 | 1 | 1
 1 | 0 | 1
 1 | 1 | 0

 A | B | A -> B
-------------------
 0 | 0 | 1
 0 | 1 | 1
 1 | 0 | 0
 1 | 1 | 1
```

# Задание

**«D»:** Добавить в иерархию классов элемент «триггер» (**TTrigger**). Построить его таблицу истинности при начальных значениях выхода Q, равных 0 и 1.

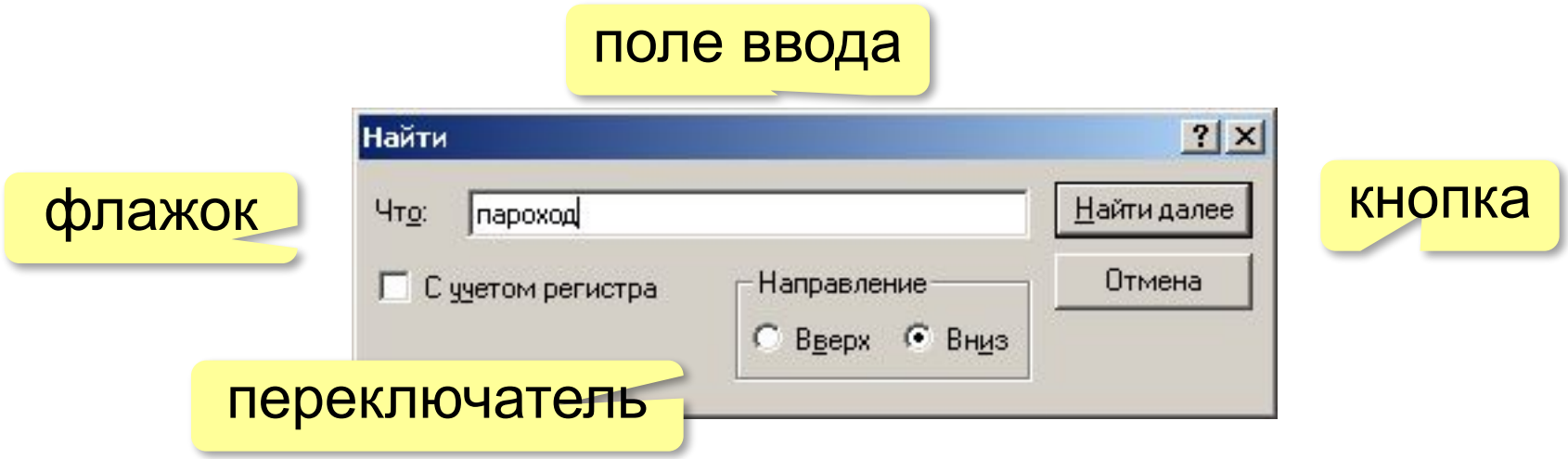
**Пример:**

```
При Q = 0:
  A | B | Q
-------------------
  0 | 0 | 0
  0 | 1 | 0
  1 | 0 | 1
  1 | 1 | 1
При Q = 1:
  A | B | Q
-------------------
  0 | 0 | 1
  0 | 1 | 0
  1 | 0 | 1
  1 | 1 | 1
```

# Объектно-ориентированное программирование

## § 51. Программы с графическим интерфейсом

# Интерфейс: объекты и сообщения

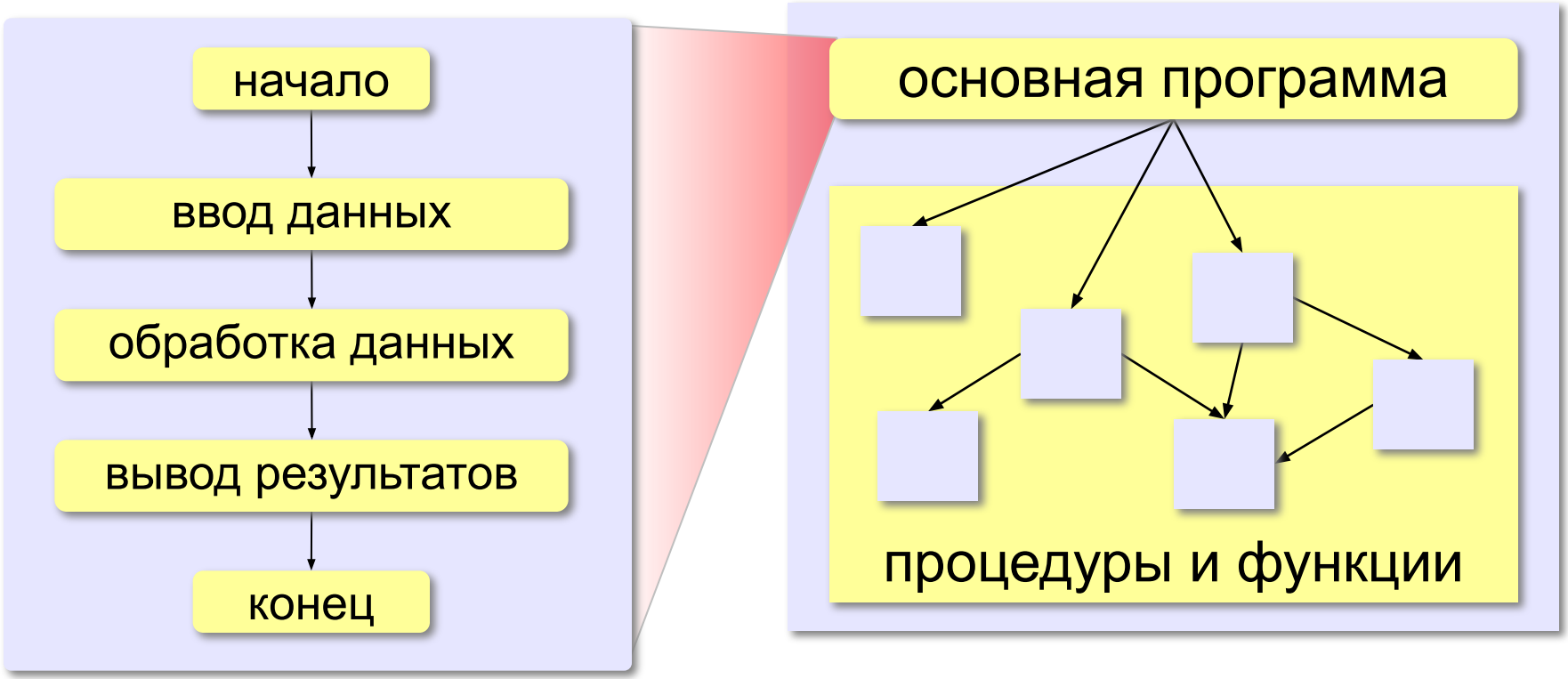



Все элементы окон – объекты, которые обмениваются данными, посылая друг другу сообщения.

**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!

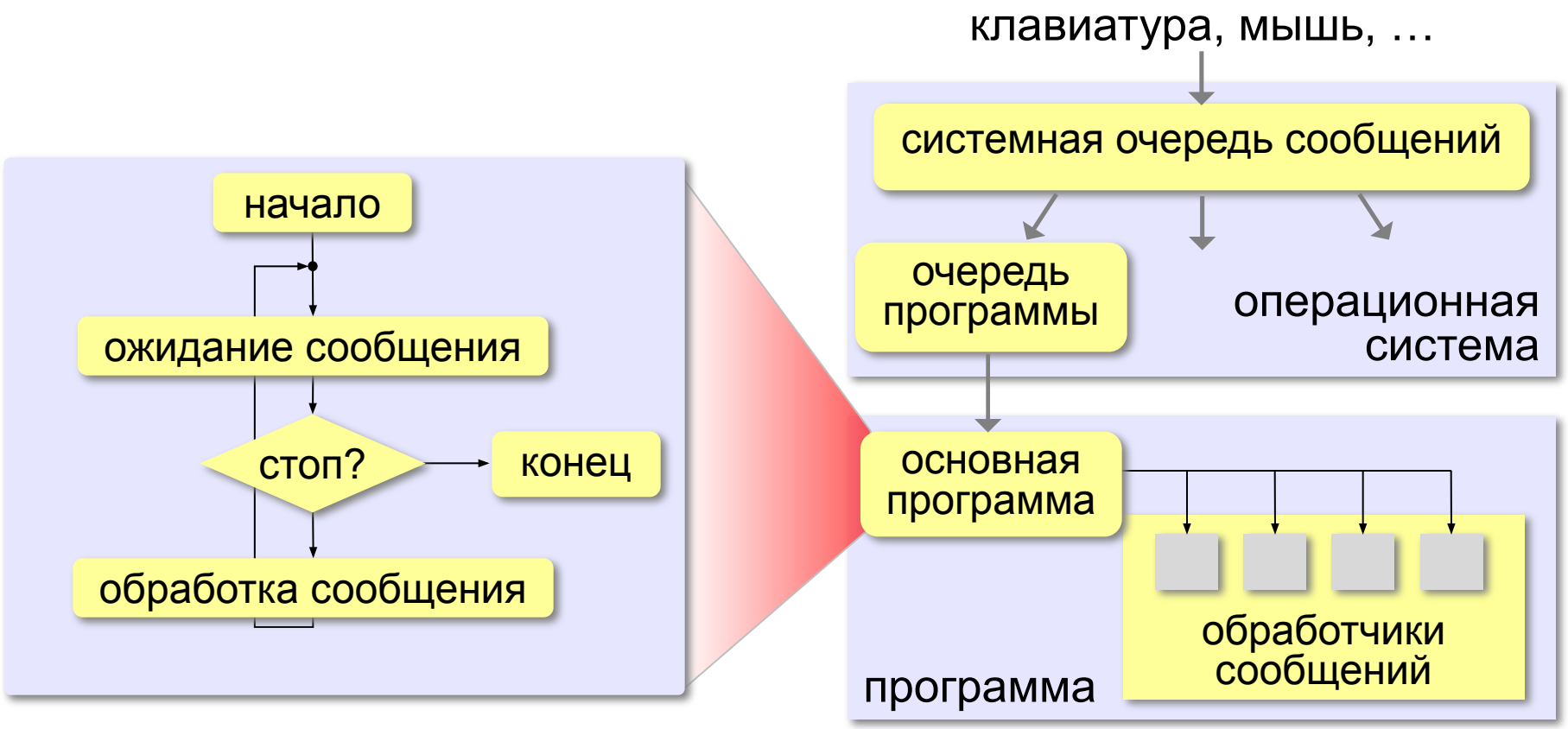
# Программы, управляемые событиями

**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

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

**Этапы разработки:**

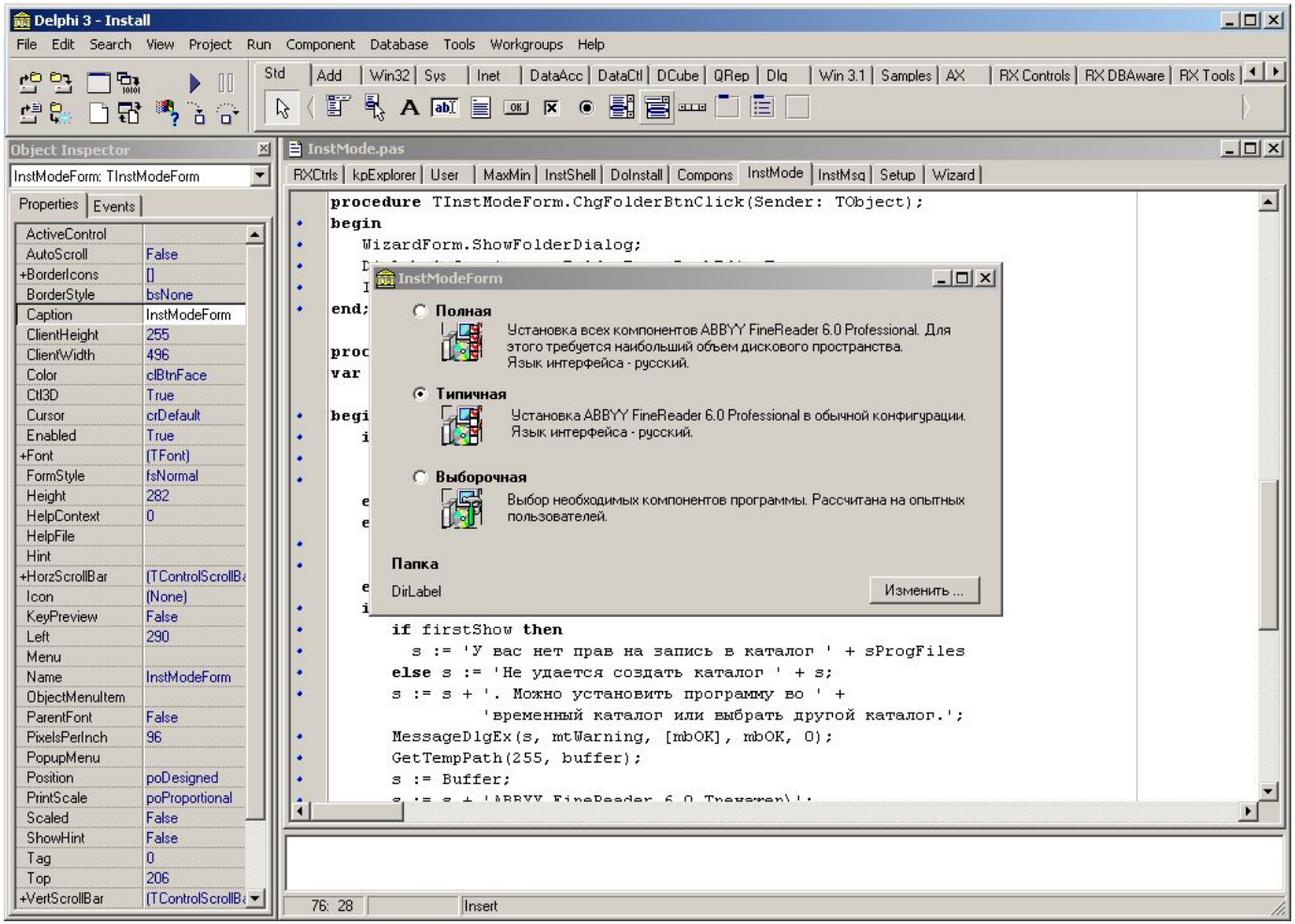
- создание **формы**
- минимальный код добавляется автоматически
- расстановка **элементов интерфейса** с помощью мыши и настройка их свойств
- создание **обработчиков** событий
- написание **алгоритмов** обработки данных

выполняются при возникновении событий

# RAD-среды: Delphi

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

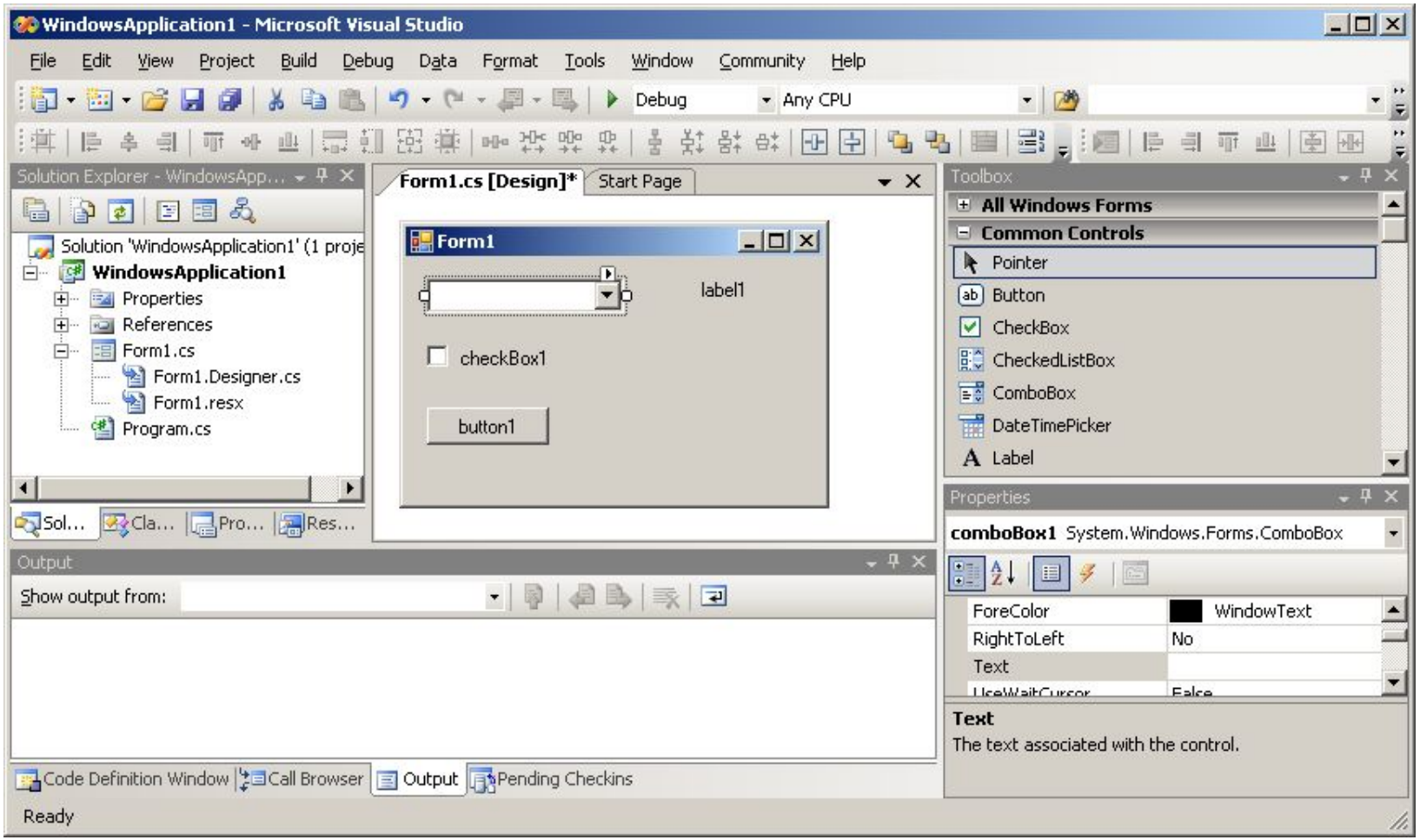
1995: *Borland*, сейчас: *Embarcadero Technologies*



# RAD-среды: MS Visual Studio

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

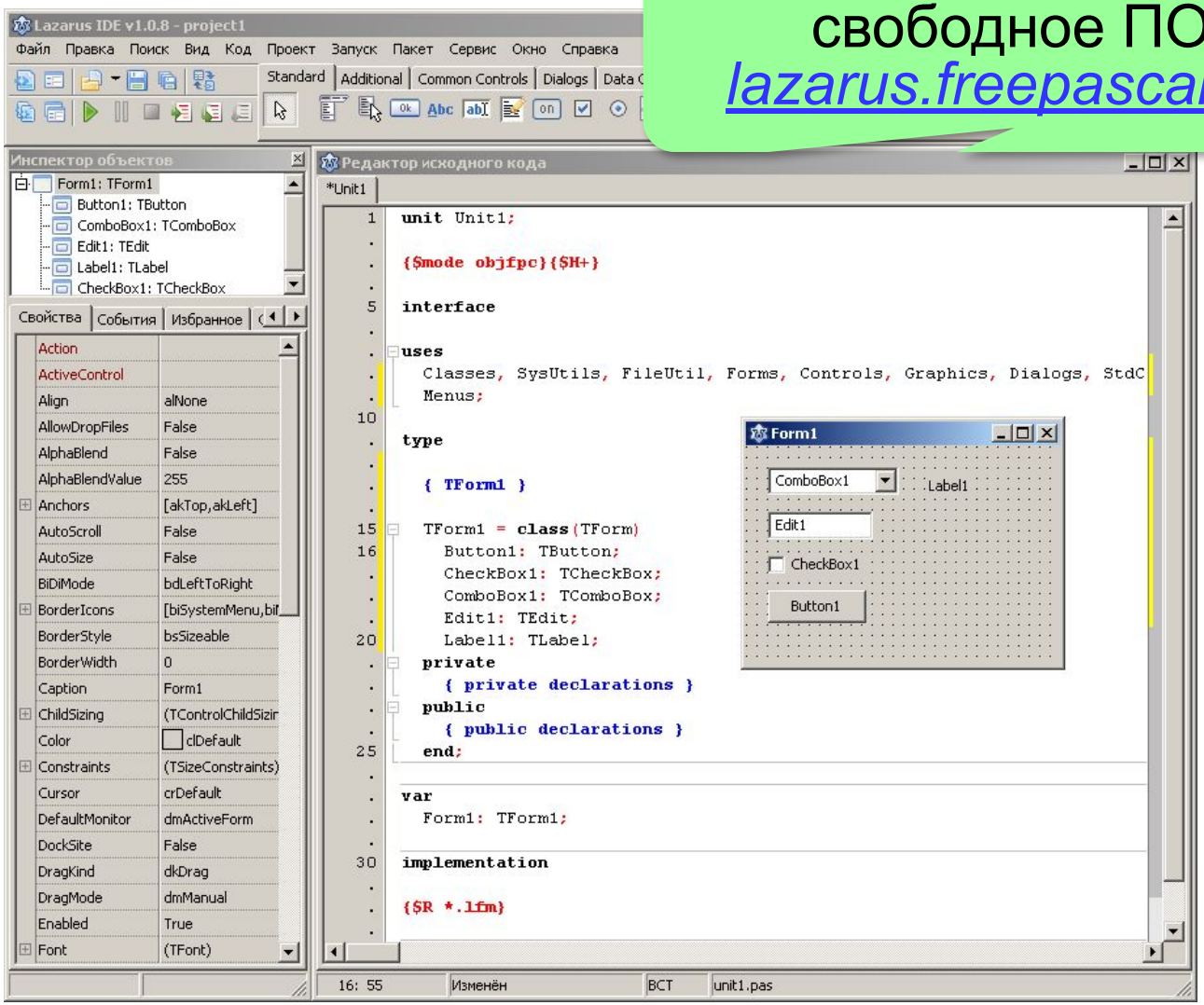
с 1995 по н.в.: *Microsoft*



# RAD-среды: Lazarus

## Языки: *FreePascal, Delphi*

свободное ПО:
lazarus.freepascal.org

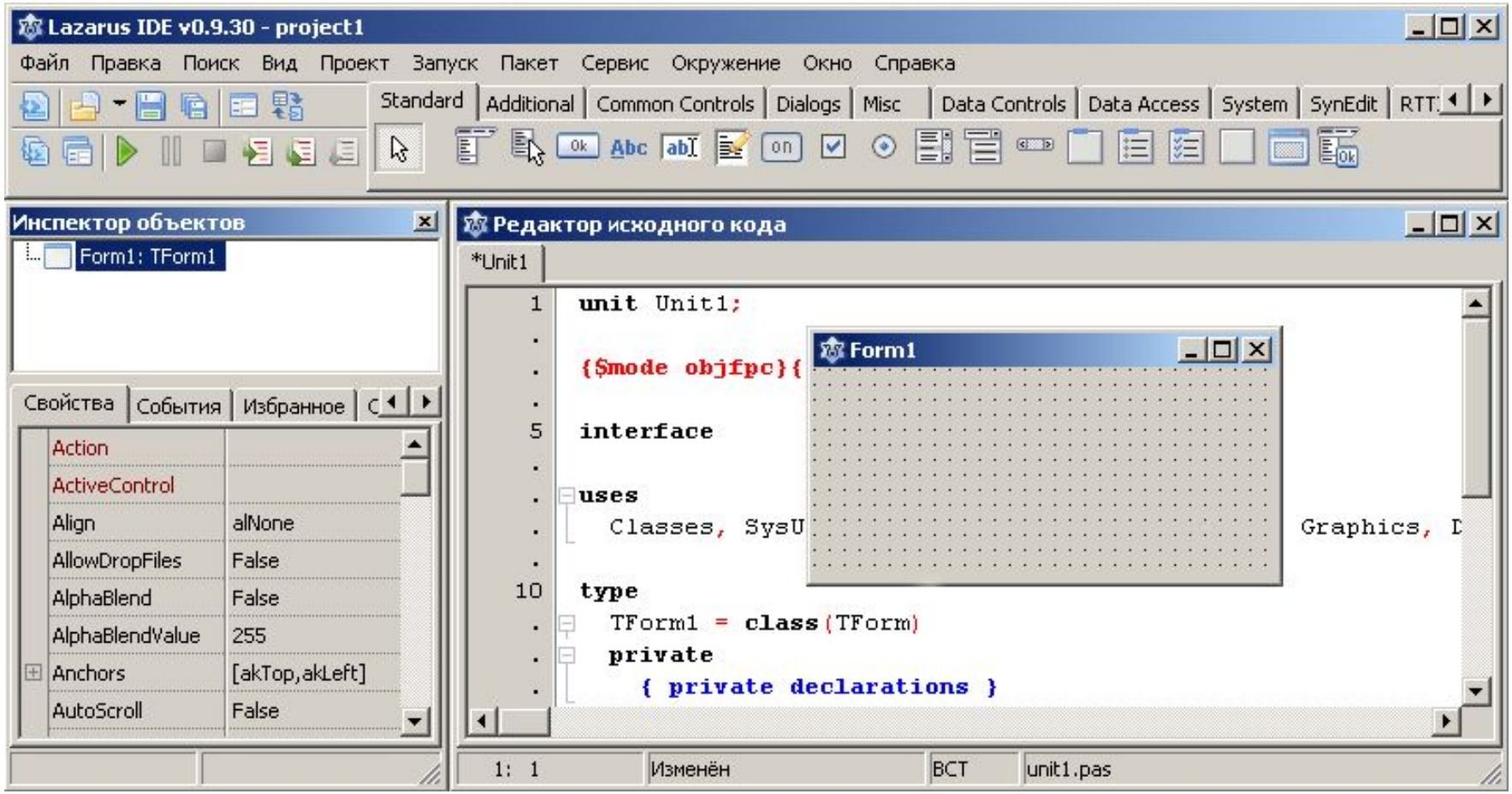


# Объектно-ориентированное программирование

## § 52. Программирование в RAD-средах

# Lazarus (lalazarus.freepascal.org)

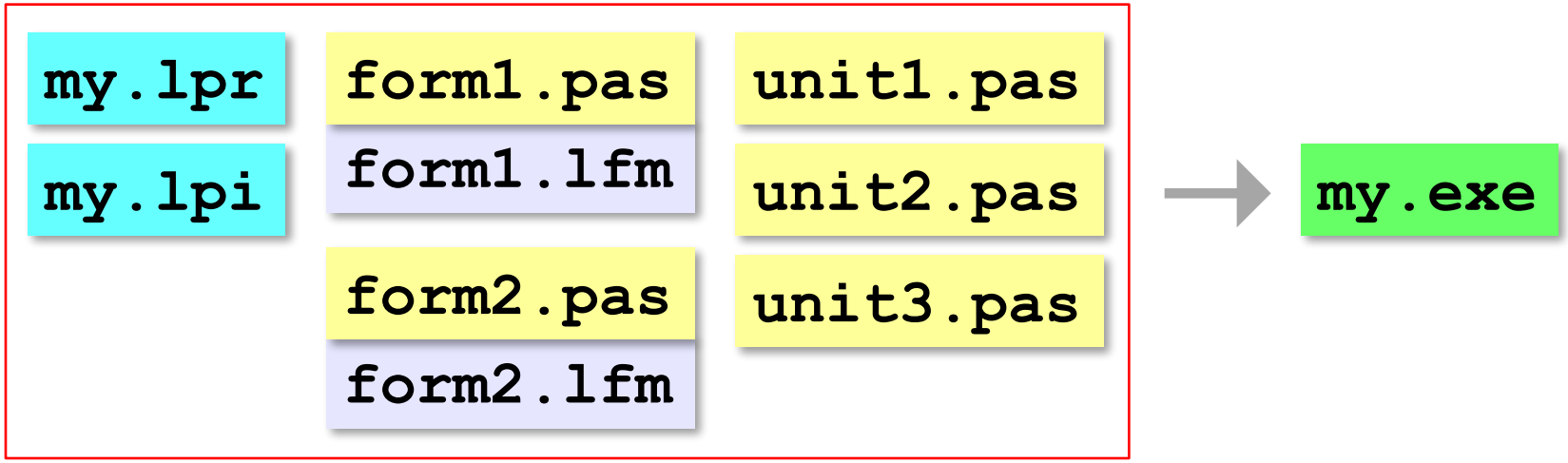
Свободное ПО для *Windows*, *Mac OS X*, *Linux*



# Проект *Lazarus*

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

- **проект** (`.lpr`, *Lazarus Project*) – основная программа
- **настройки** проекта (`*.lpi`, *Lazarus Project Information*)
- **модули**, из которых состоит программа (`*.pas`);
- **формы** (`*.lfm`, *Lazarus Form*) – описания внешнего вида и свойств окон и их элементов.

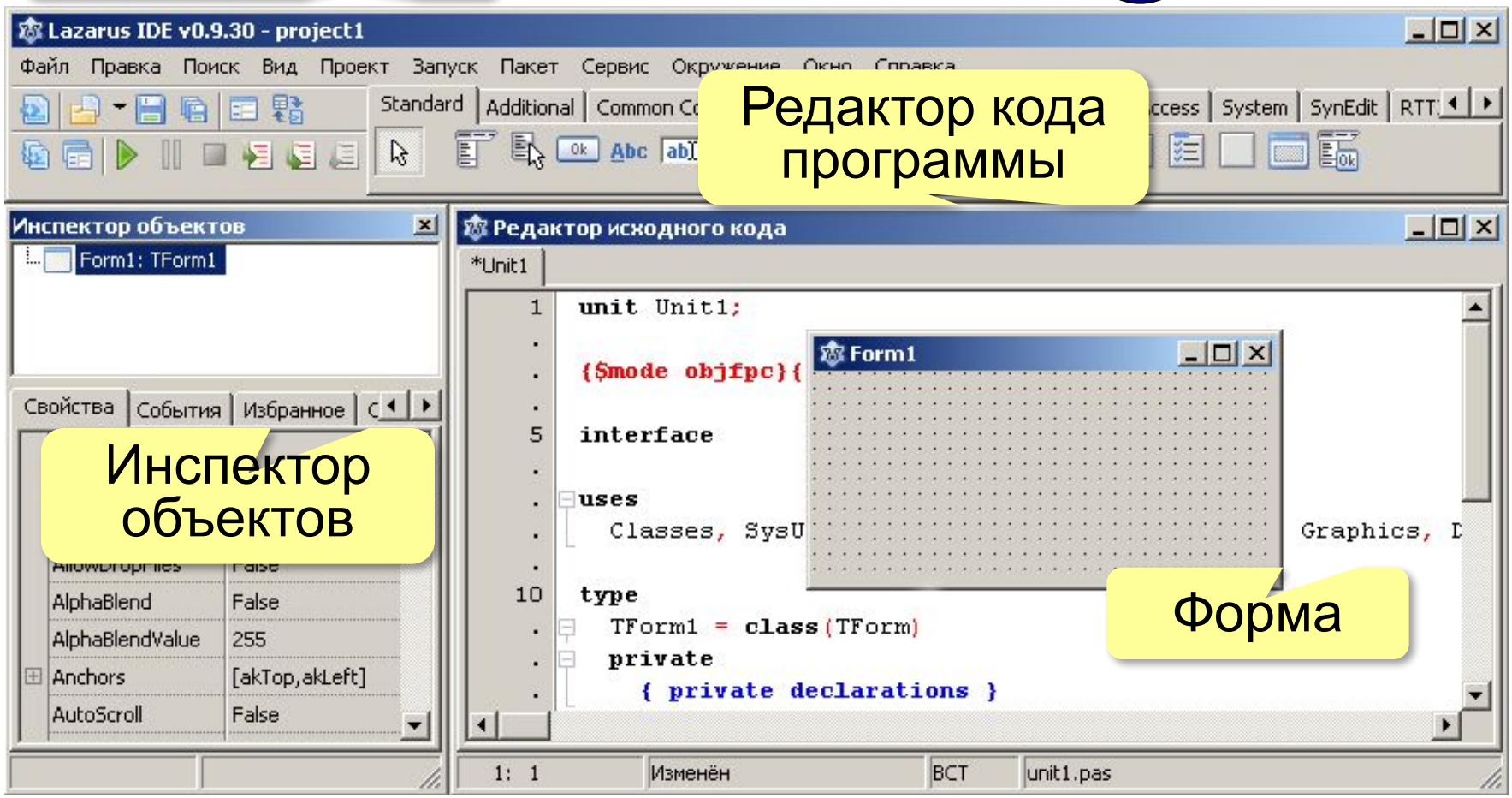


# Простейший проект

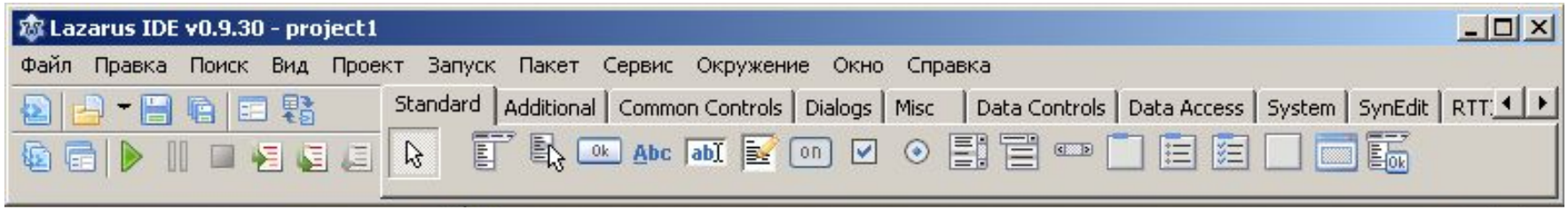
*Файл – Создать – Проект – Приложение*

главное окно

F9 – запуск!



# Главное окно




кнопки быстрого вызова команд

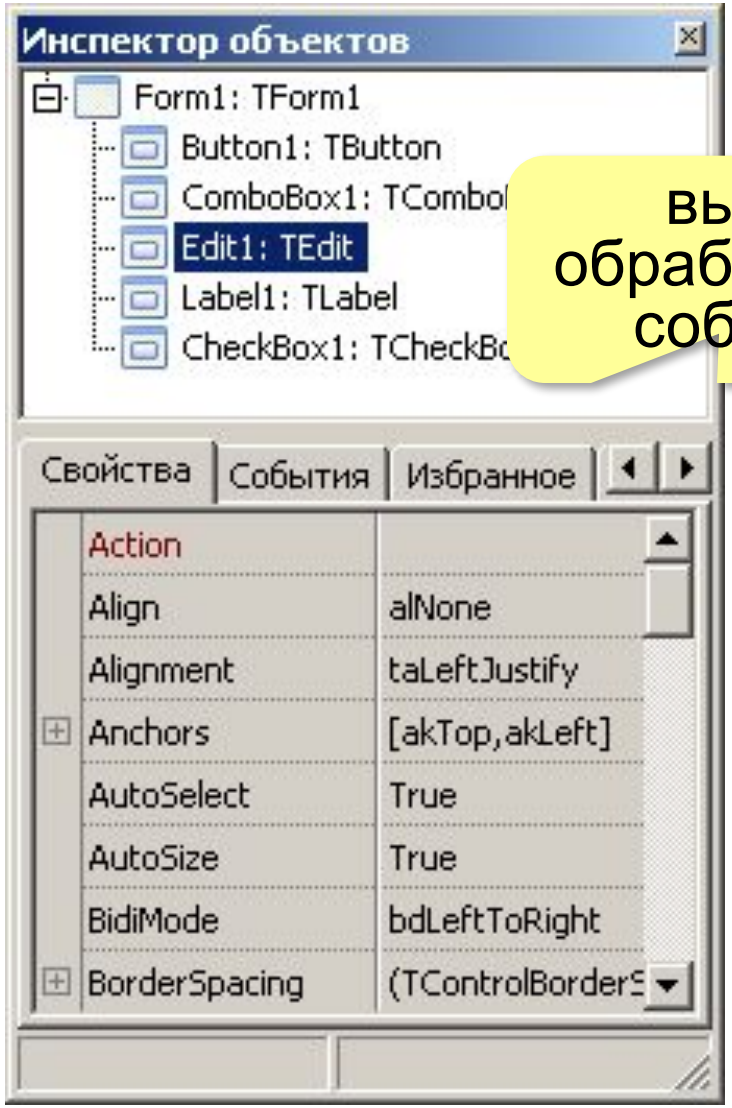
палитра **компонентов** – готовых объектов

# Инспектор объектов

дерево объектов

свойства выделенного объекта

выбор обработчиков событий



# Модуль формы

```
unit Unit1;
interface
uses
  Classes, SysUtils, FileUtil, Forms, Controls,
  Graphics, Dialogs;
type
  TForm1 = class(TForm)
  private
    { private declarations }
  public
    { public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.lfm}
end.
```

модули библиотеки *Lazarus*

класс формы

объявление формы

подключает описание формы и компонентов

# Основная программа (проект)

***Ctrl+F12*** – список модулей, выбрать **`*.lpr`**

```
program project1;
uses
  Interfaces, Forms, Unit1;
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

модули библиотеки

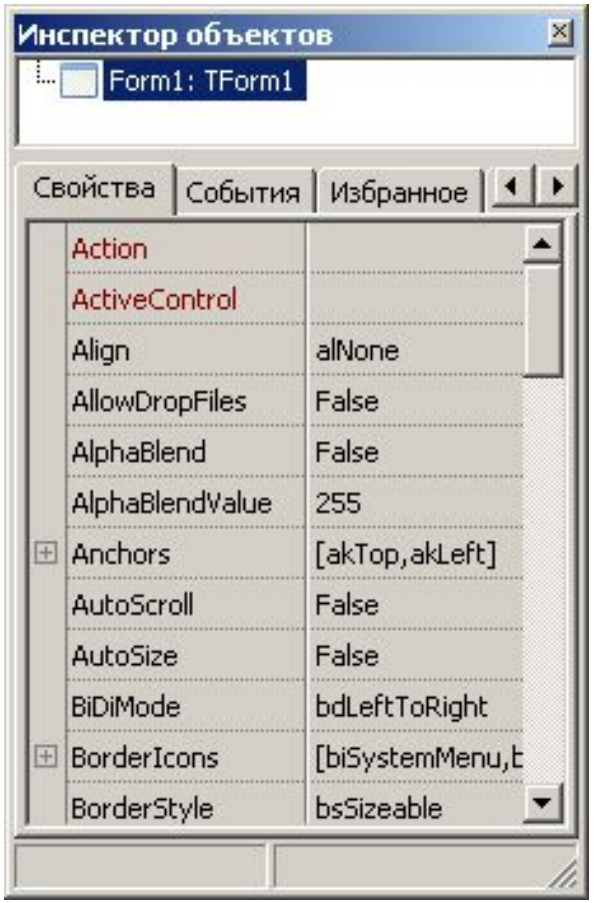
модуль формы

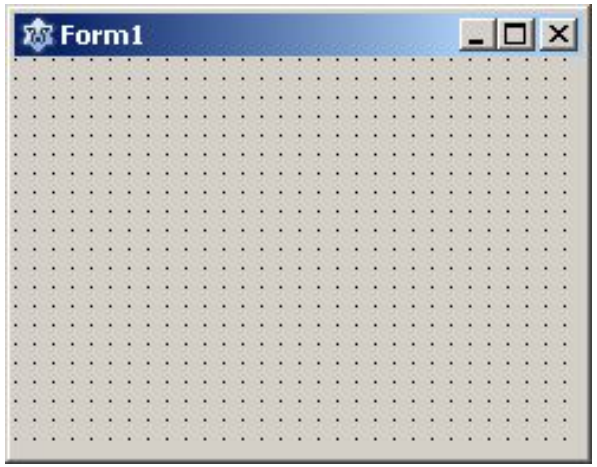
начальные установки

создание формы

запуск цикла обработки сообщений

# Свойства формы

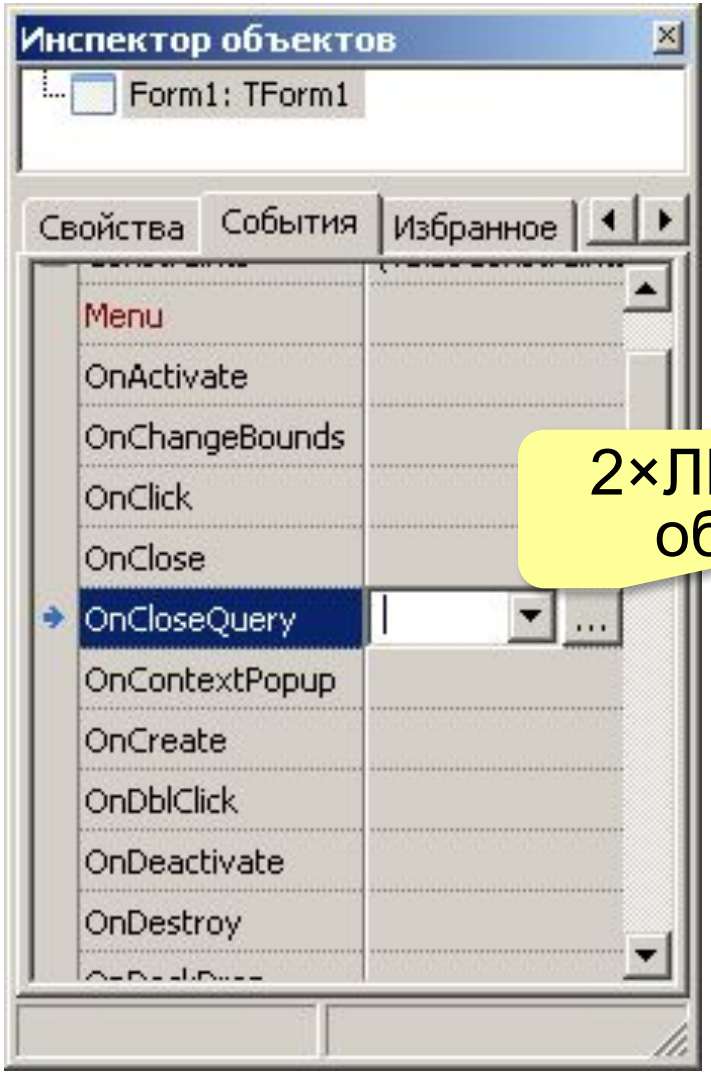


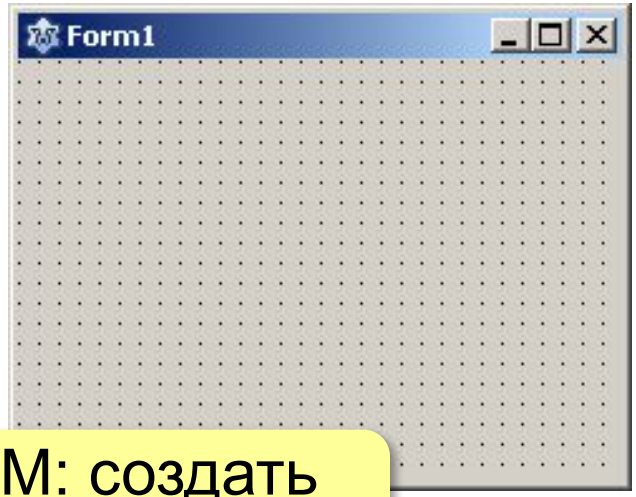




**Left** – левая граница (x-координата)
**Top** – верхняя граница (y-координата)
**Width** – ширина
**Height** – высота
**Name** – имя формы
**Caption** – текст в заголовке окна
**Color** – цвет рабочей области
**Font** – шрифт надписей
**Visible** – видимость (да/нет)

**MainForm** (**TMainForm**)

# Обработчик событий




2×ЛКМ: создать обработчик

**`OnCloseQuery`**: запрос разрешения на закрытие формы

# Обработчик события

метод класса TForm1

название обработчика

общий предок всех объектов

```
procedure TForm1.FormCloseQuery(
    Sender: TObject;
    var CanClose: boolean);
begin

end;
```

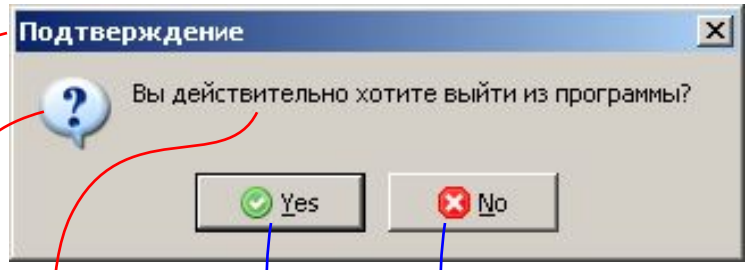
кто послал сообщение

результат (можно/нельзя)

! Автоматически добавлен в описание класса формы!

# Диалог с вопросом

## Функция MessageDlg:




```
procedure TForm1.FormCloseQuery(
                  Sender: TObject;
                     var CanClose: boolean);
var res: TModalResult;
begin
  res:= MessageDlg('Подтверждение',
  'Вы действительно хотите выйти из программы?',
                mtConfirmation, [mbYes,mbNo], 0);
  CanClose := (res = mrYes)
end;
```

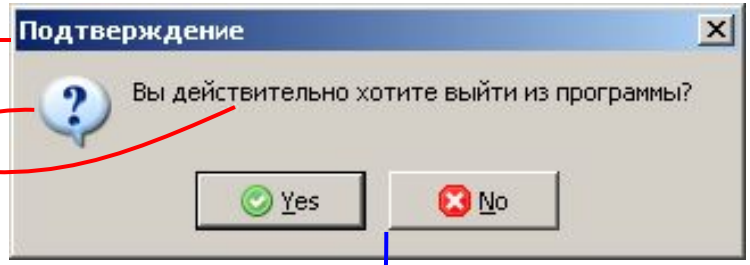
тип: результат диалога

нет справки

«да», если нажали Yes

# Параметры MessageDlg

- заголовок окна
- сообщение пользователю
- тип запроса
  - **mtError** ошибка
  - **mtWarning** предупреждение
  - **mtInformation** информация
  - **mtConfirmation** подтверждение
- набор (множество) кнопок:
  - **mbYes** «да»
  - **mbNo** «нет»
  - **mbOK** «OK»
  - **mbCancel** «Отмена»
- номер раздела справки



# Объектно-ориентированное программирование

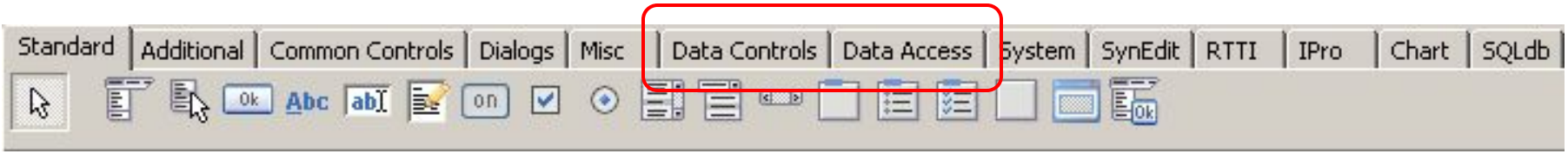
## § 53. Использование компонентов

# Панель компонентов

Стандартные

Диалоги

Системные



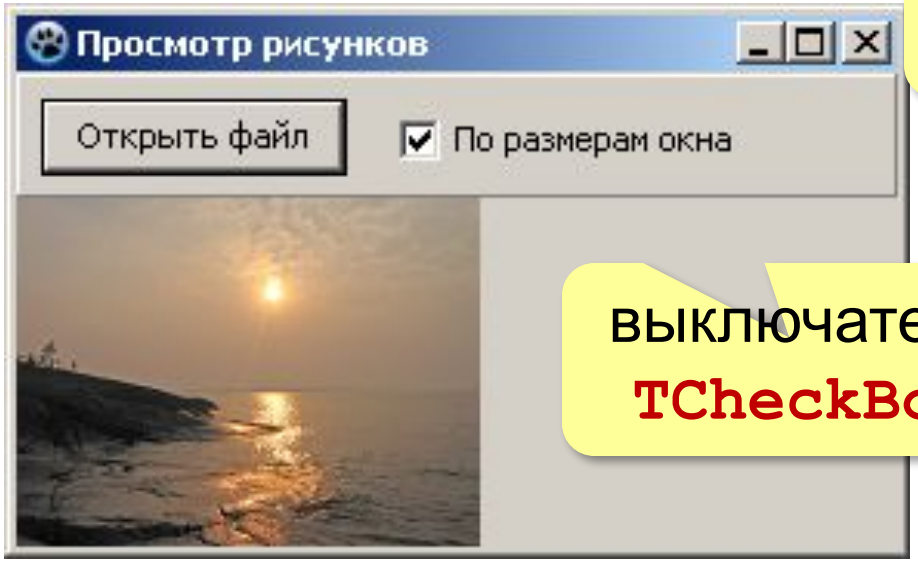

Дополнительные

Базы данных

# Просмотр рисунков

кнопка **TButton**

рисунок **TImage**

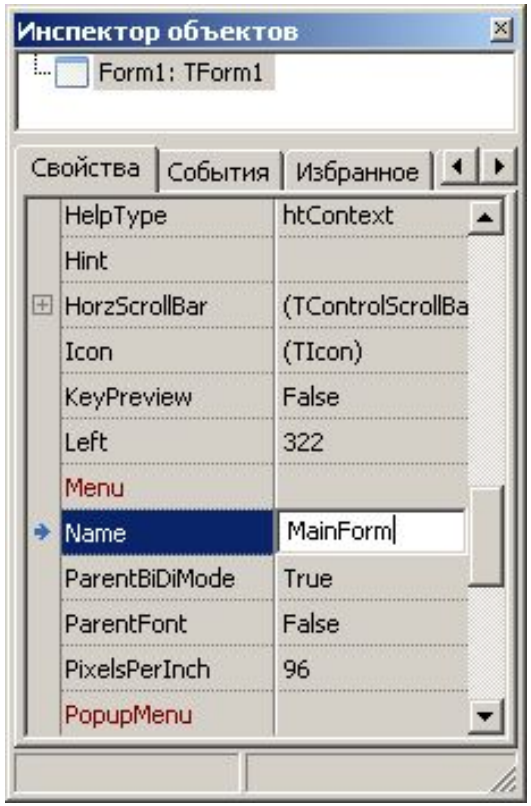


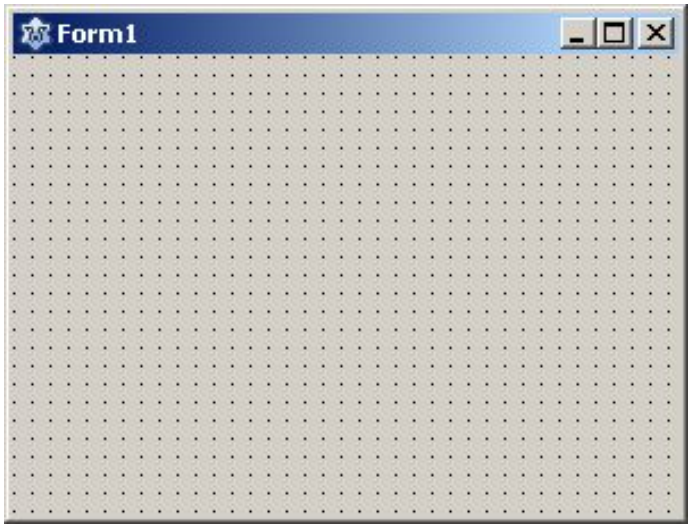

панель **TPanel**

выключатель **TCheckBox**

# Настройка формы

*Файл – Создать – Проект – Приложение*

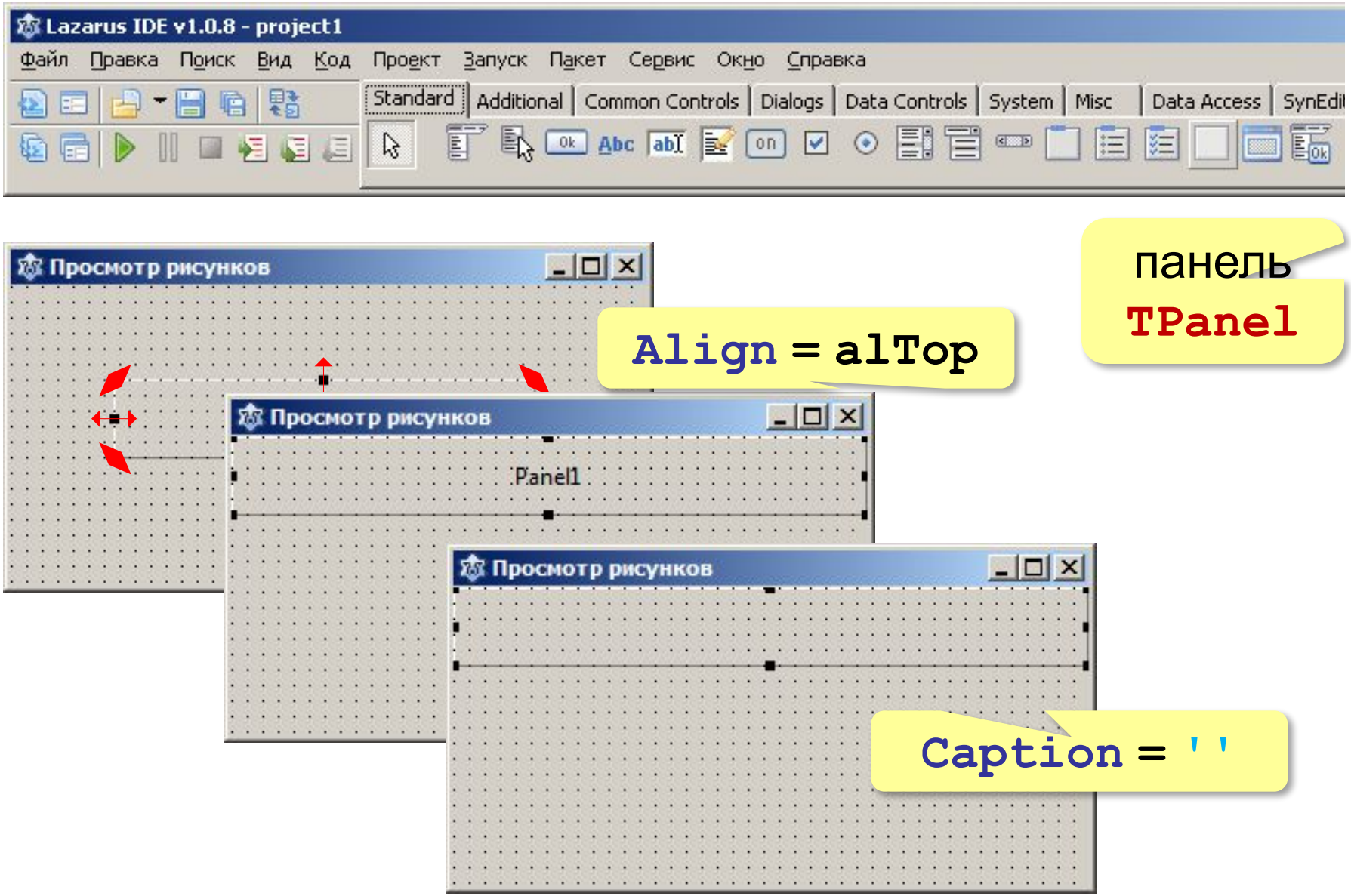


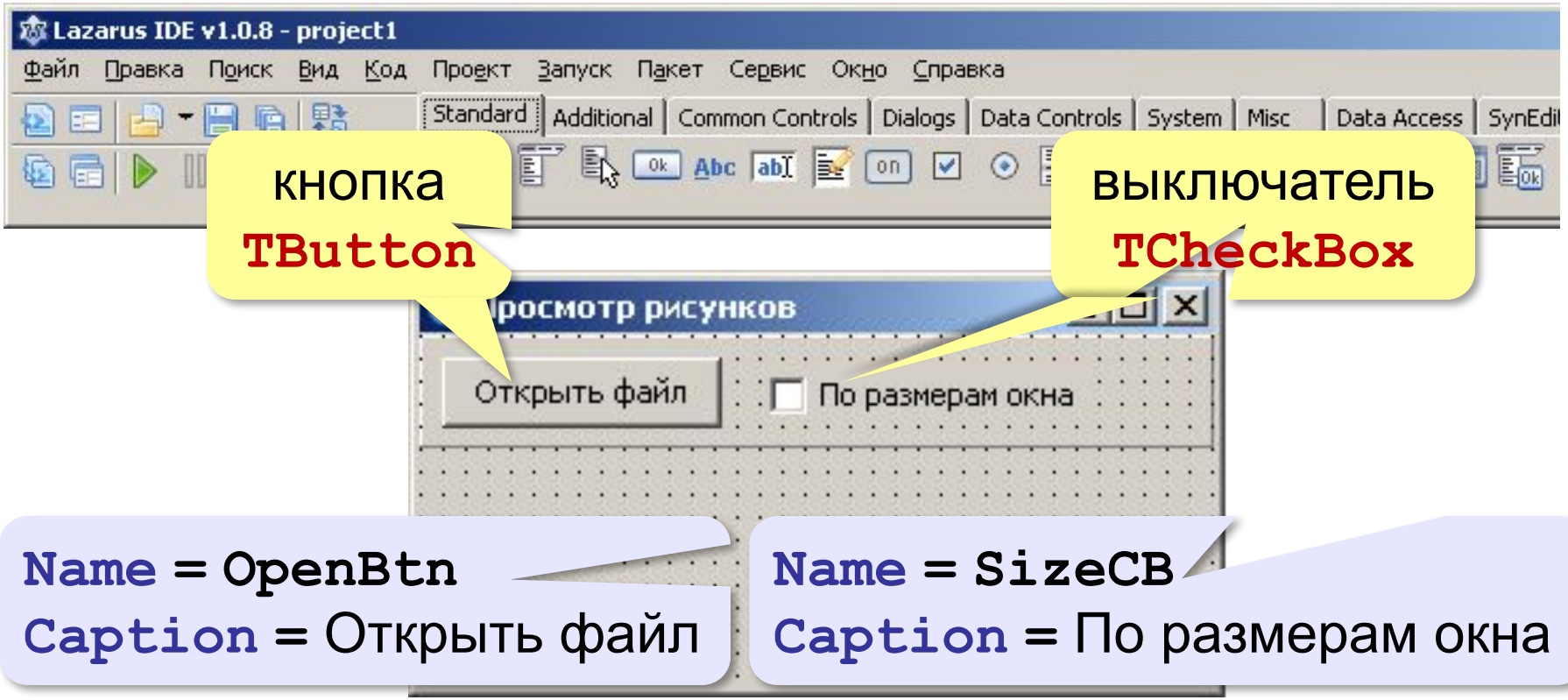

**Name** → MainForm

**Caption** → Просмотр рисунков

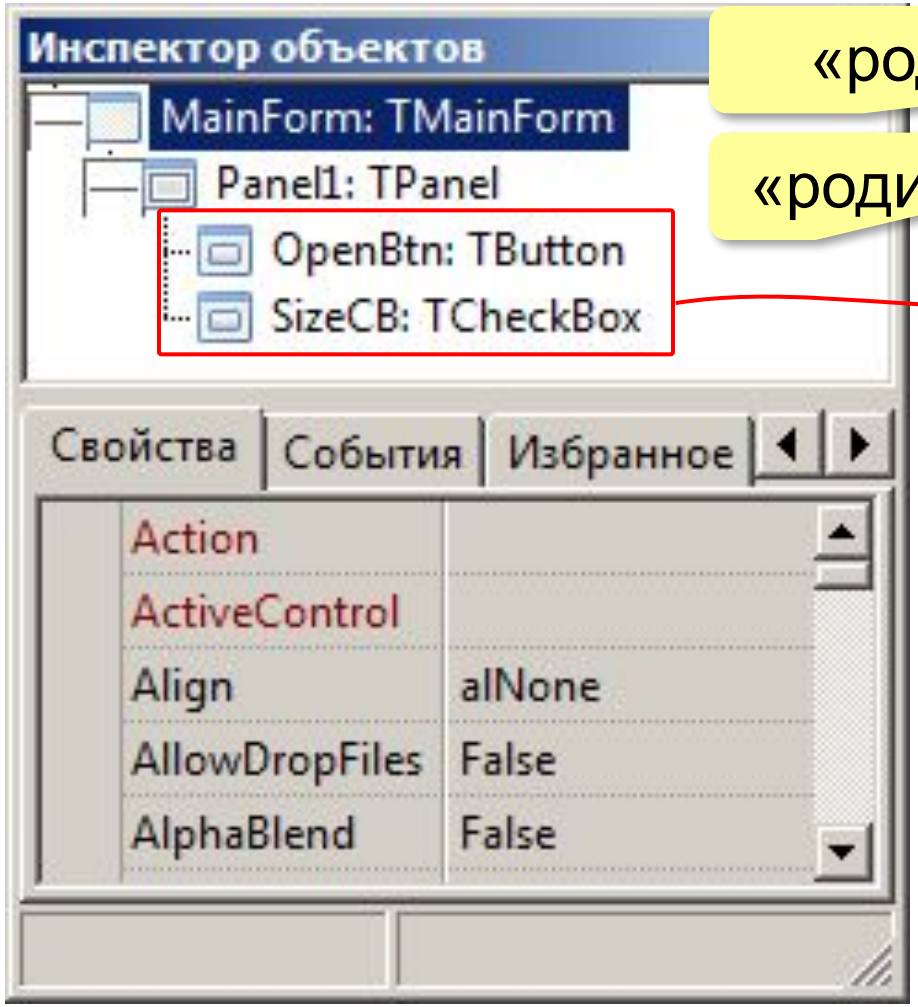
# Верхняя панель



# Кнопка и выключатель



# Инспектор объектов



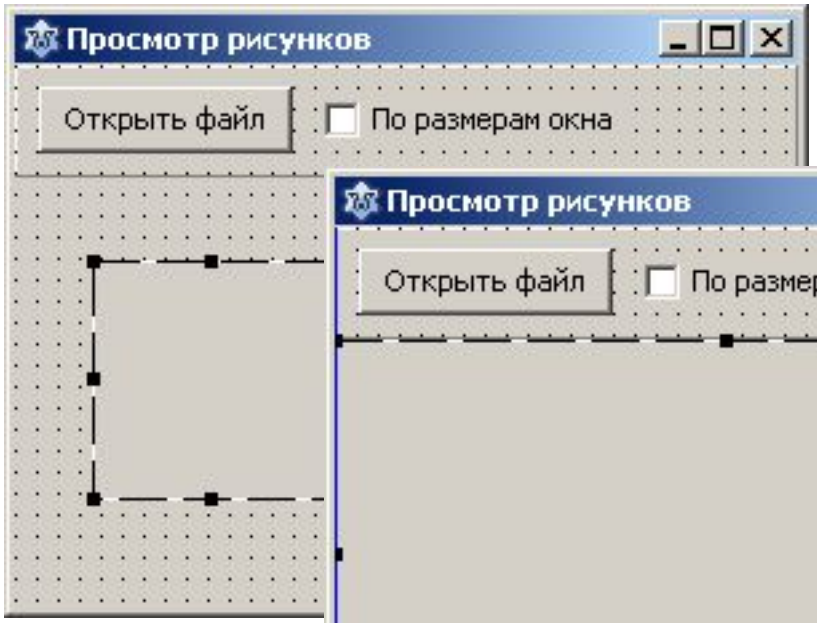

«родитель» для **Panel1**

«родитель» для …

**Родительский объект:**

- отвечает за перерисовку дочерних объектов
- все дочерние объекты перемещаются вместе с ним

# Компонент TImage

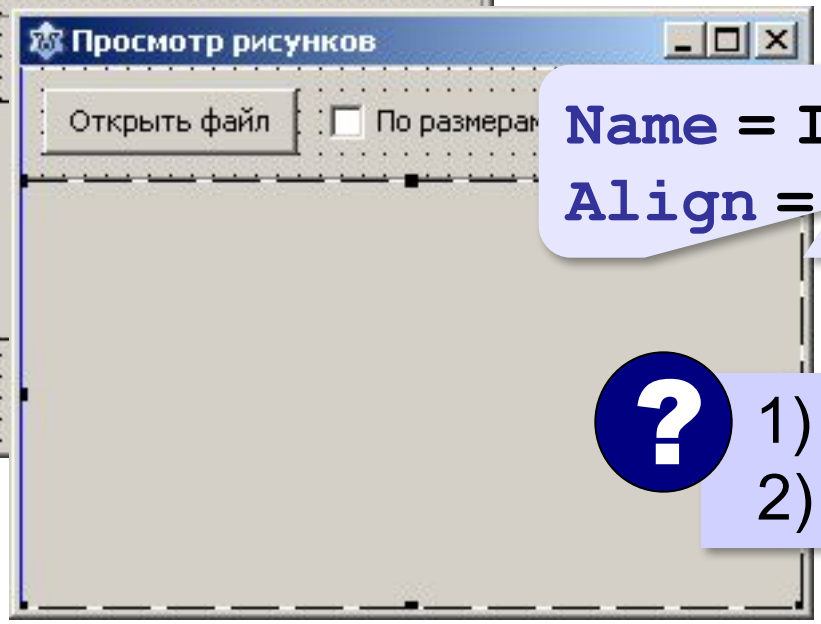
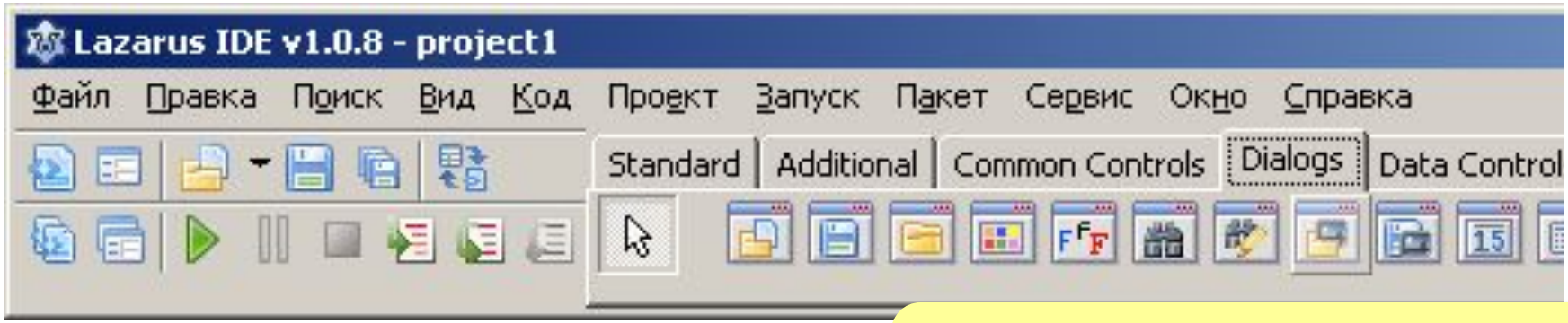




рисунок **TImage**

**Name** = **Image**
**Align** = **alClient**

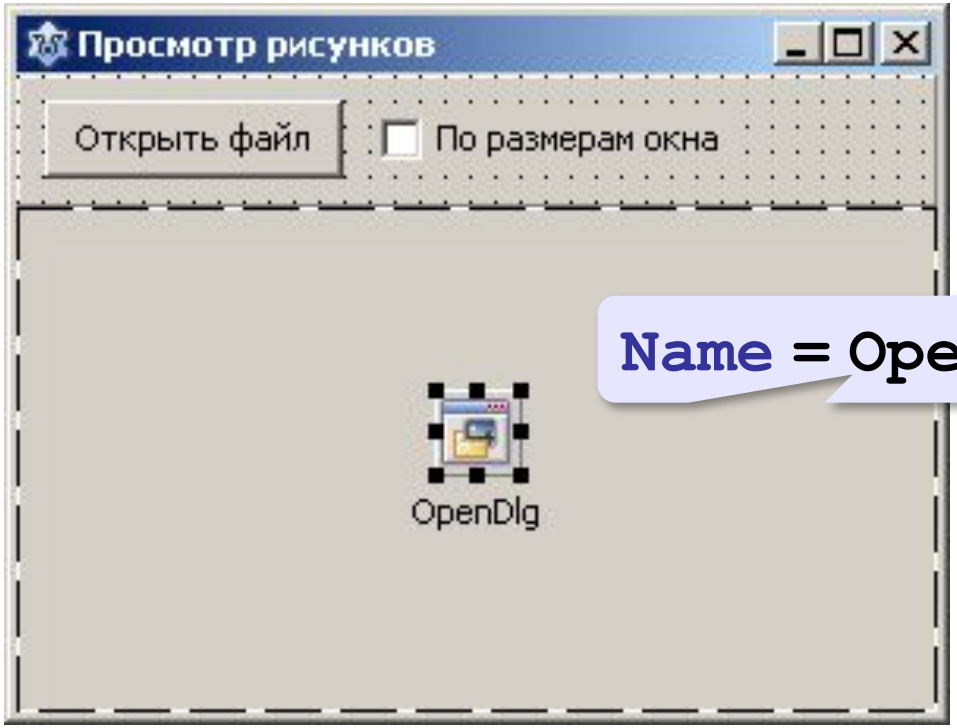
? 1) Загрузка файла?
2) Масштабирование?

# Выбор файла



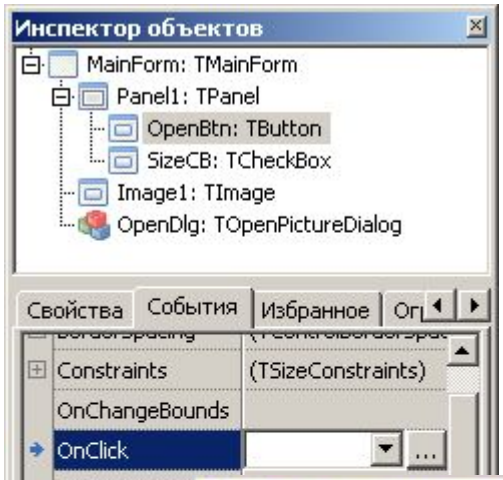

выбор рисунка

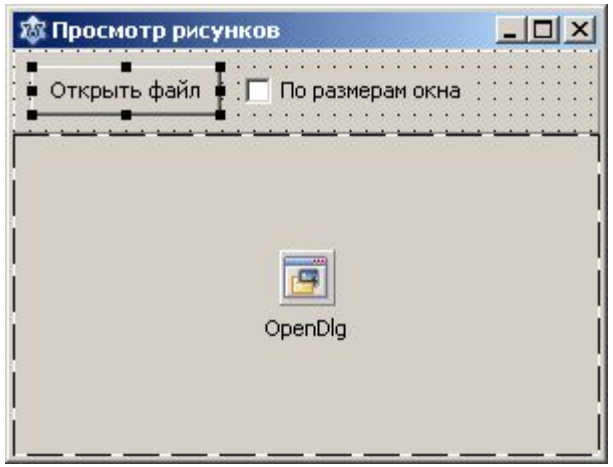
**TOpenPictureDialog**

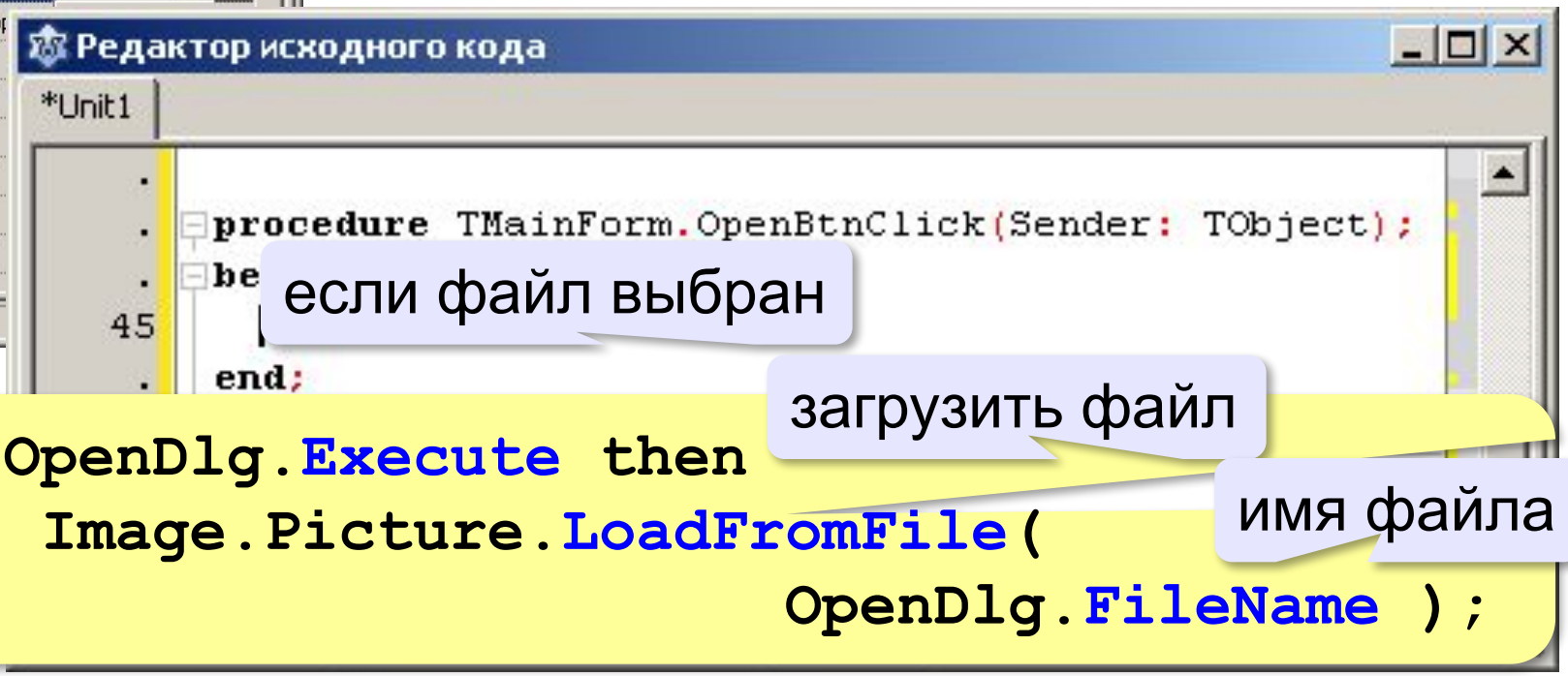



**Name** = **OpenDlg**

# Выбор файла



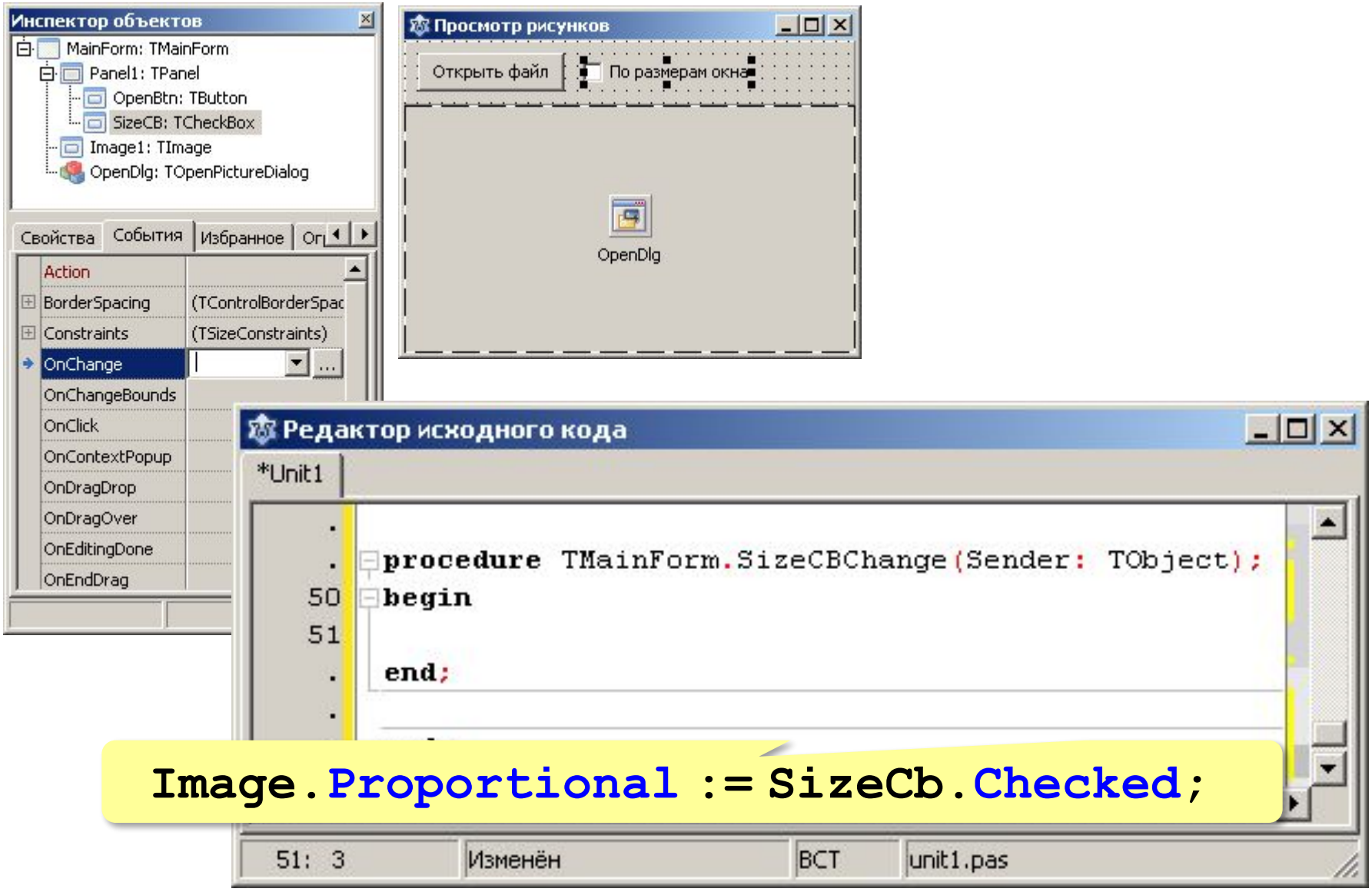

если файл выбран

загрузить файл

имя файла

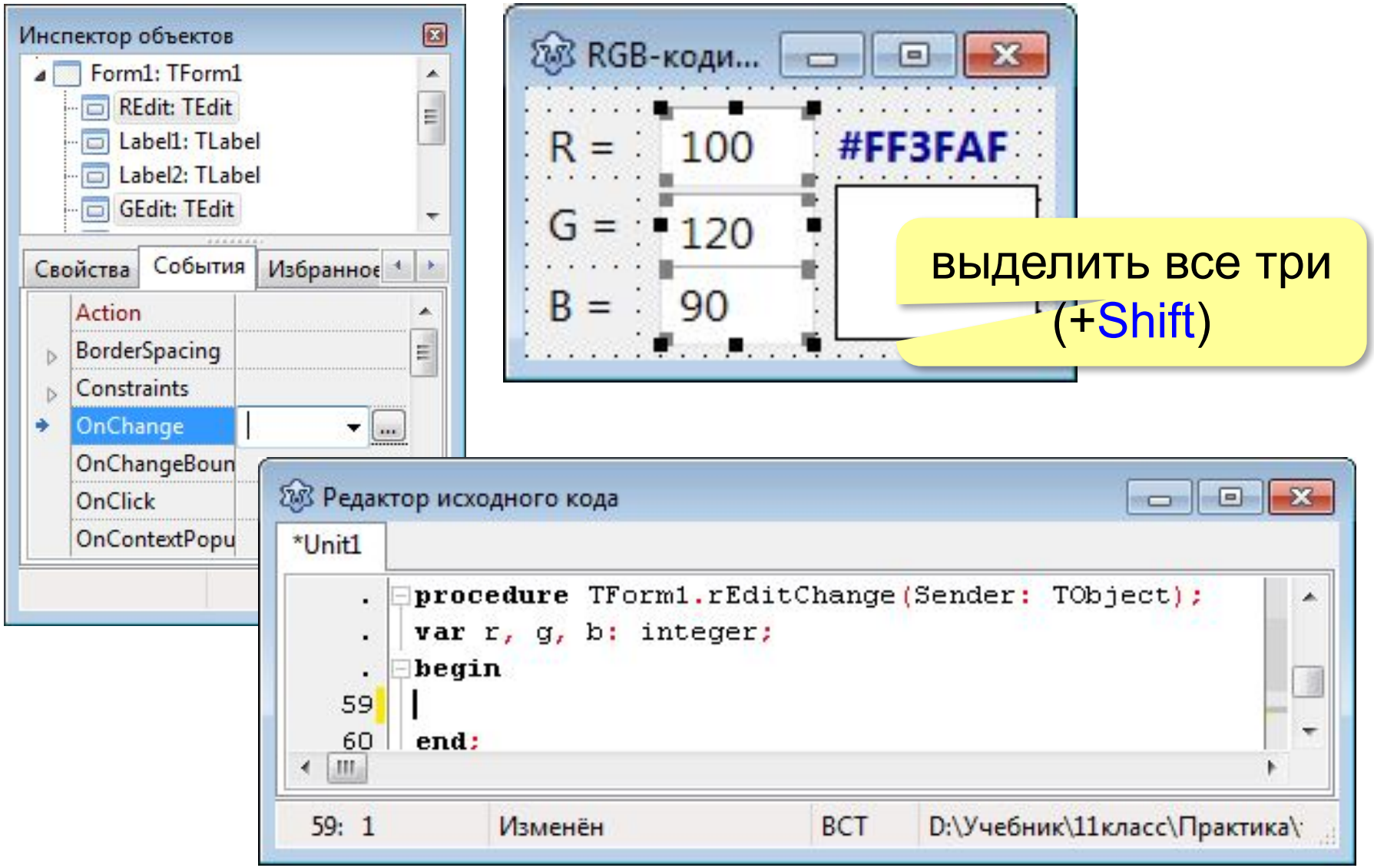
```
if OpenDlg.Execute then
    Image.Picture.LoadFromFile(
                        OpenDlg.FileName );
```

# Масштабирование



# Ввод и вывод данных

# Обновление компонентов вывода



# Обновление компонентов вывода

```
procedure TForm1.rEditChange(
                                Sender: TObject);
var r, g, b: integer;
begin
  r:=StrToInt(rEdit.Text);
  g:=StrToInt(gEdit.Text);
  b:=StrToInt(bEdit.Text);
  rgbShape.Brush.Color:= RGBToColor(r,g,b);
  rgbLabel.Caption:= '#'+IntToHex(r,2)
     +IntToHex(g,2)+IntToHex(b,2)
end;
```

из строки в число

построить цвет

в шестнадцатеричную систему

# Вызов при запуске

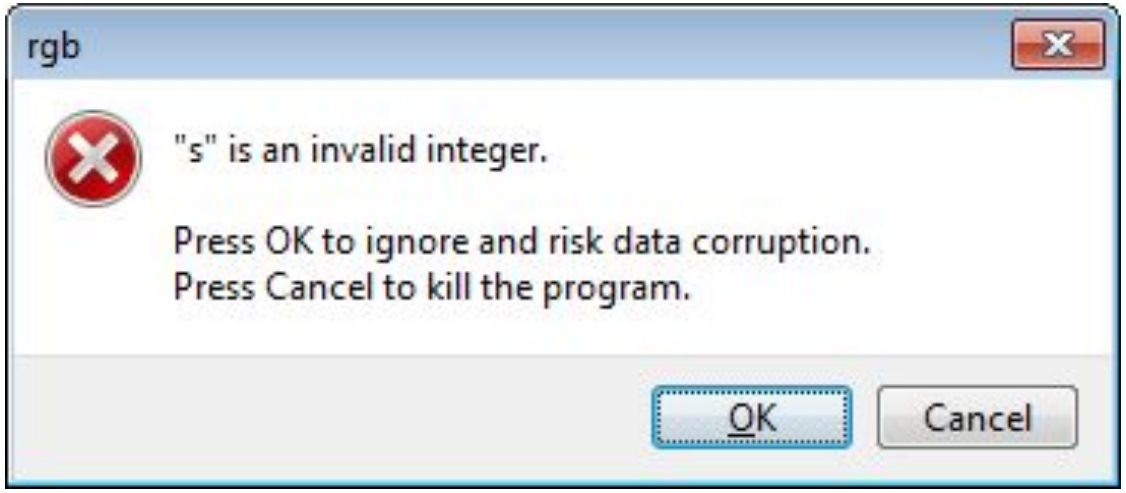
```
procedure TForm1.FormCreate(
                        Sender: TObject);
begin
   rEditChange(rEdit)
end;
```

вызов обработчика

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

# Обработка ошибок

? Если вместо числа ввести букву?




! Программа не должна «вылетать»!

# Обработка ошибок

попытаться выполнить

```
try
   { «опасные» команды }
except
   { обработка ошибки }
end;
```

если **исключение** (аварийная ситуация)

? Какие у нас опасные операции?

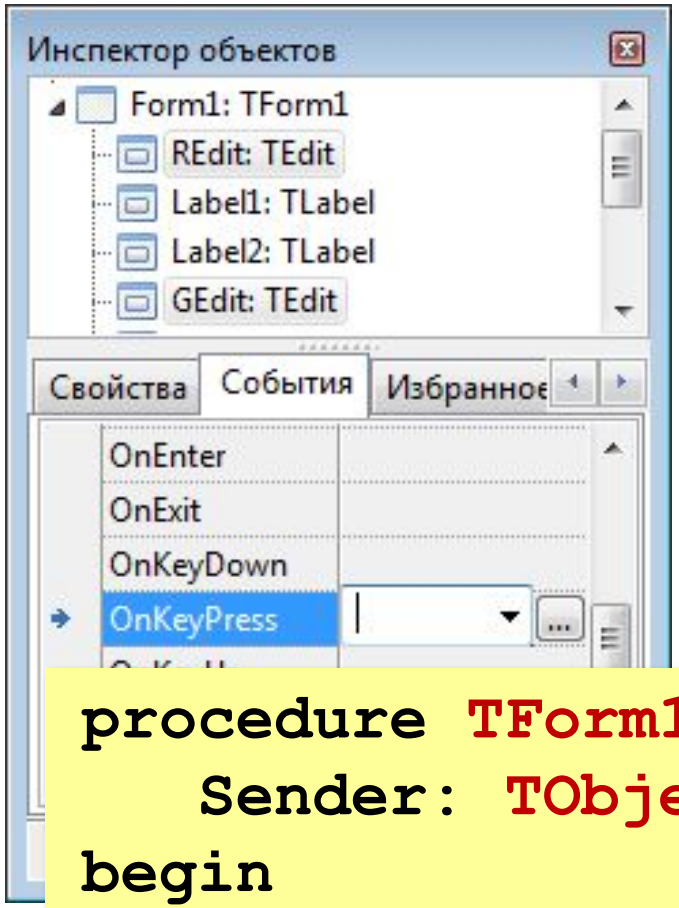
# Обработка ошибок

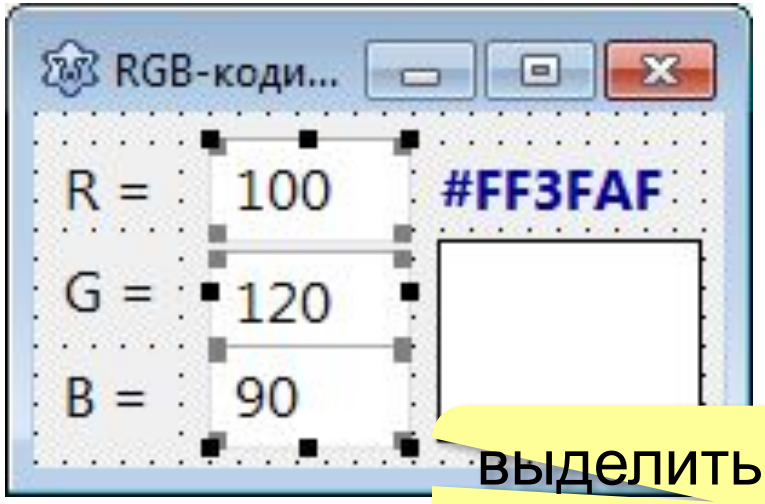
```
try
   r:= StrToInt(rEdit.Text);
   g:= StrToInt(gEdit.Text);
   b:= StrToInt(bEdit.Text);
   rgbShape.Brush.Color:= RGBToColor(r,g,b);
   rgbLabel.Caption:= '#' + IntToHex(r,2)
      + IntToHex(g,2) + IntToHex(b,2)
except
   rgbLabel.Caption:='?'
end;
```

если ошибка, записать '?'

? Что делать, если ошибка?

# Блокирование неверных символов




выделить все три (+Shift)

```
procedure TForm1.rEditKeyPress(
    Sender: TObject; var Key: char);
begin
  if not (Key in ['0'..'9',#8]) then
     Key:=#0
end;
```

Backspace

множество разрешённых символов

# Задание

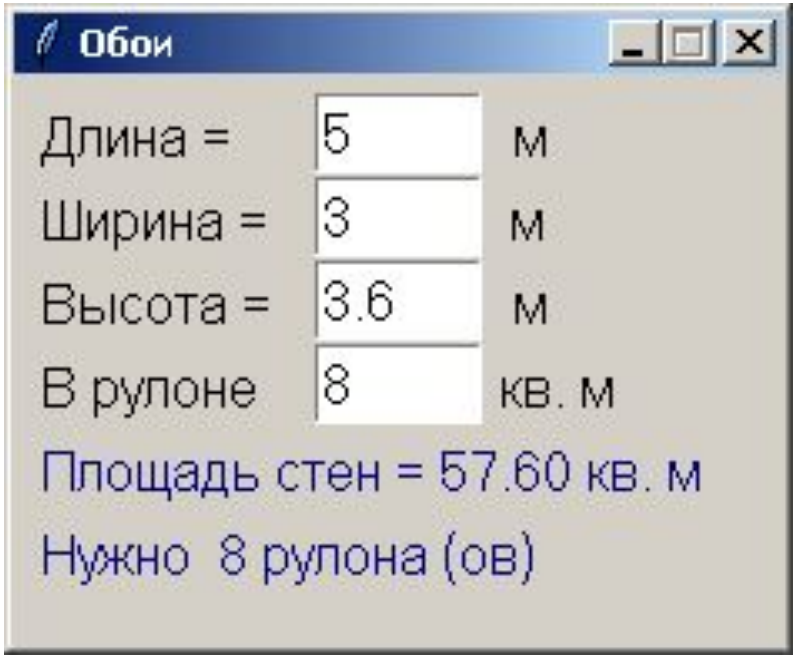
**«А»:** Постройте программу, которая вычисляет площадь комнаты.

**Требования**:

1) размер окна нельзя менять
2) при попытке закрыть окно выдаётся запрос на подтверждение
3) площадь пересчитывается сразу же, как только изменяются значения длины или ширины комнаты
4) если длина или ширина отрицательны или не числа, вместо площади выводится знак вопроса

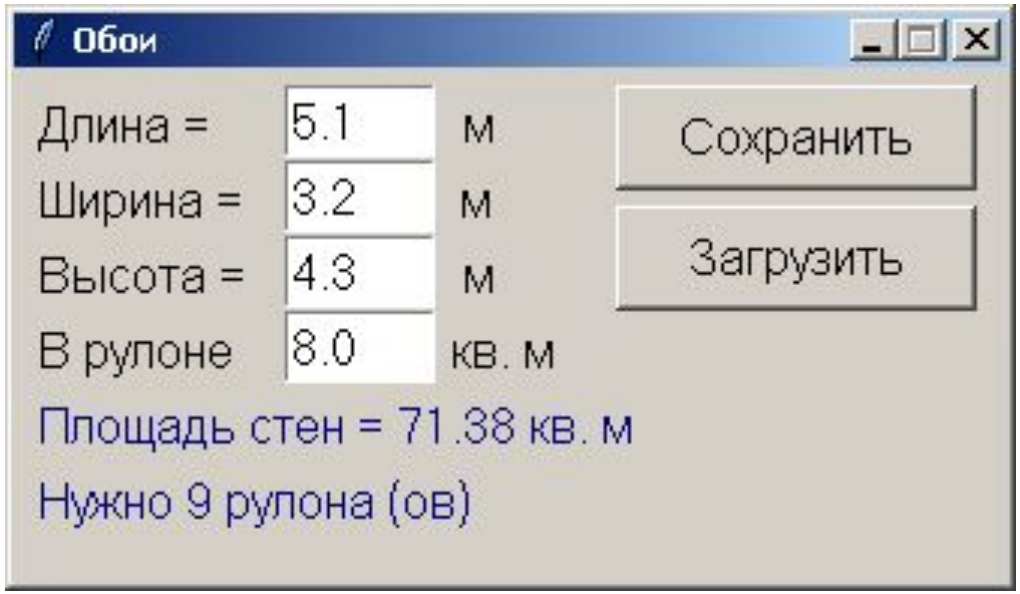

# Задание

**«B»:** Постройте программу, которая вычисляет площадь стен комнаты и определяет, сколько рулонов обоев нужно на оклейку всех стен. Количество рулонов – целое число. Остальные требования такие же, как в варианта «A».



# Задание

**«С»:** Доработайте программу так, чтобы по щелчку по кнопке «Сохранить» все данные сохранялись в файле с расширением **.dat** (имя файла можно выбрать), а по щелчку по кнопке «Загрузить» данные загружались из файла (имя файла также выбирается).



# Объектно-ориентированное программирование

## § 54. Совершенствование компонентов

# Что требуется?

*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 **TEdit**!

```
type
  TIntEdit = class(TEdit)
  ...
end;
```

# Поле для ввода целых чисел

- переопределить обработчик **KeyPress** (защита от ввода неверных символов)
- свойство **Value** (значение)
  методы **GetValue** (чтение) и **SetValue** (запись)

```
type
  TIntEdit = class(TEdit)
  private
    function GetValue: integer;
    procedure SetValue(Val: integer);
  protected
    procedure KeyPress(var Key: char);
                                override;
  public
    property Value: integer read GetValue
                           write SetValue
  end;
```

# Поле для ввода целых чисел

**Чтение** целого числа:

```
function TIntEdit.GetValue: integer;
begin
   try     Result:=StrToInt(Text);
   except  Result:=0
   end
end;
```

из строки в целое

**Запись** целого числа:

```
procedure TIntEdit.SetValue(Val: integer);
begin
   Text:=IntToStr(Val)
end;
```

из целого в строку

# Поле для ввода целых чисел

## Нажатие на клавишу:

```
procedure TIntEdit.KeyPress(var Key: char);
begin
  if not (Key in ['0'..'9', #8]) then
            Key:=#0;
  inherited
end;
```

вызвать аналогичный метод базового класса

# Поле для ввода целых чисел

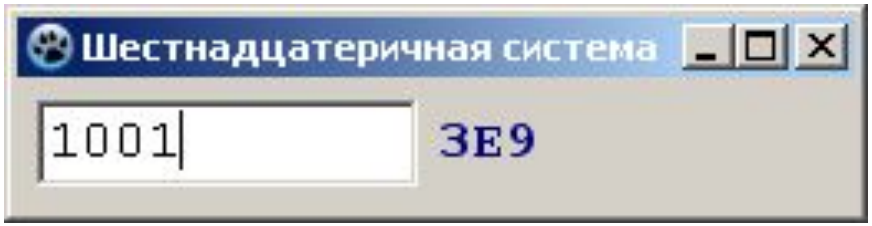
**Модуль** (*Файл – Создать модуль*):

```
unit int_edit;
{$mode objfpc}
interface
uses
  Classes, SysUtils, StdCtrls ;
type
  TIntEdit = class(TEdit)
  ...
  end;
implementation
  { методы класса TIntEdit }
end.
```

подключение **TEdit**

# Поле для ввода целых чисел

## Использование:




поле ввода **decEdit**
**TIntEdit**

метка **hexLabel**
**TLabel**

```
unit Unit1;
{$mode objfpc}
interface
uses
  Classes, ..., ExtCtrls, int_edit ;
```

подключение модуля

# Поле для ввода целых чисел

Компонента **TIntEdit** нет в палитре!

**Добавление вручную**:

```
TForm1 = class(TForm)
  ...
  decEdit: TIntEdit;
end;
```

это только указатель!

Когда создавать объект?

# Поле для ввода целых чисел

**Создание**:

```
procedure TForm1.FormCreate(
                    Sender: TObject);
begin
  decEdit:= TIntEdit.Create(Self);
  decEdit.Text:='100';
  decEdit.Left:=6;
  decEdit.Top:=6;
  decEdit.Width:=115;
  decEdit.Parent:=Self
end;
```

создать объект

владелец – форма (удаляет)

«родитель» – форма (перерисовывает)

# Поле для ввода целых чисел

## Обработчик сообщения:

```
TForm1 = class(TForm)
    ...
  procedure decEditChange(
                    Sender: TObject);
public
  decEdit: TIntEdit;
end;
```

добавить вручную

добавить вручную

```
procedure TForm1.decEditChange(
                          Sender: TObject);
begin
  hexLabel.Caption:=
                  IntToHex(decEdit.Value,1)
end;
```

# Поле для ввода целых чисел

## Подключение обработчика сообщения:

```
procedure TForm1.FormCreate(
                    Sender: TObject);
begin
  decEdit:= TIntEdit.Create(Self);
  decEdit.Text:='100';
  decEdit.Left:=6;
  decEdit.Top:=6;
  decEdit.Width:=115;
  decEdit.Parent:=Self;
  decEdit.OnChange:=@decEditChange;
  decEditChange(Sender);
end;
```

адрес процедуры

вызов при старте

# Добавление компонента в палитру

```
unit int_edit;
{$mode objfpc}
interface
…
implementation
…
procedure Register;
begin
  RegisterComponents('Samples', [TIntEdit]);
end;

end.
```

процедура регистрации

страница в палитре компонентов

названия классов (множество)

# Добавление компонента в палитру

**Порядок установки** в среде *Lazarus*:

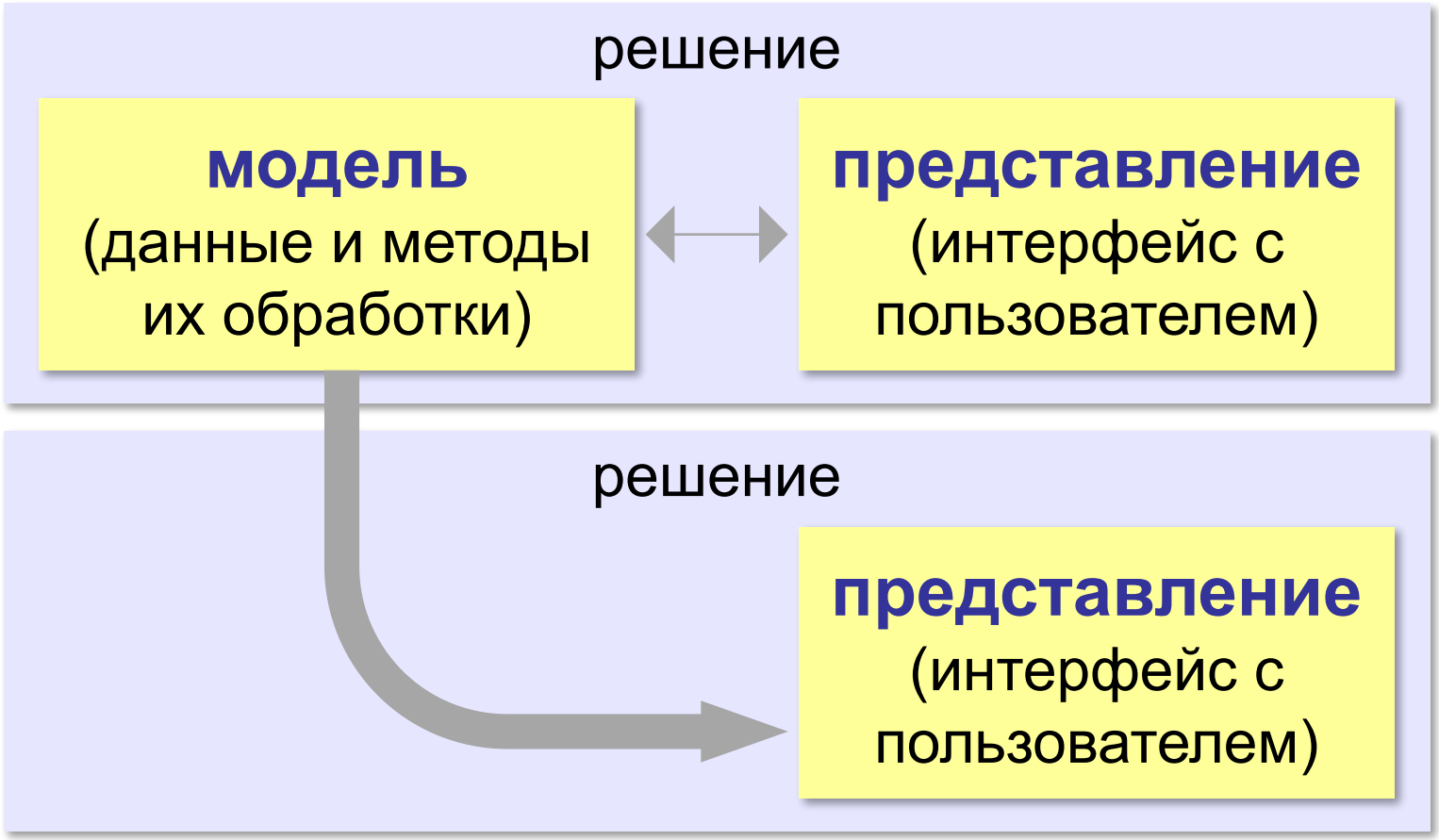
1. *Пакет* → *Новый пакет* → дать имя пакету и сохранить
2. *Добавить* → *Добавить файлы* → *Добавить файлы к пакету* → выбрать файл **`int_edit.pas`**
3. *Добавить* → *Новая зависимость* → *LCLBase*
4. Компилировать
5. *Использовать* → *Установить*
   Согласиться на пересборку IDE.

# Объектно-ориентированное программирование

## § 55. Модель и представление

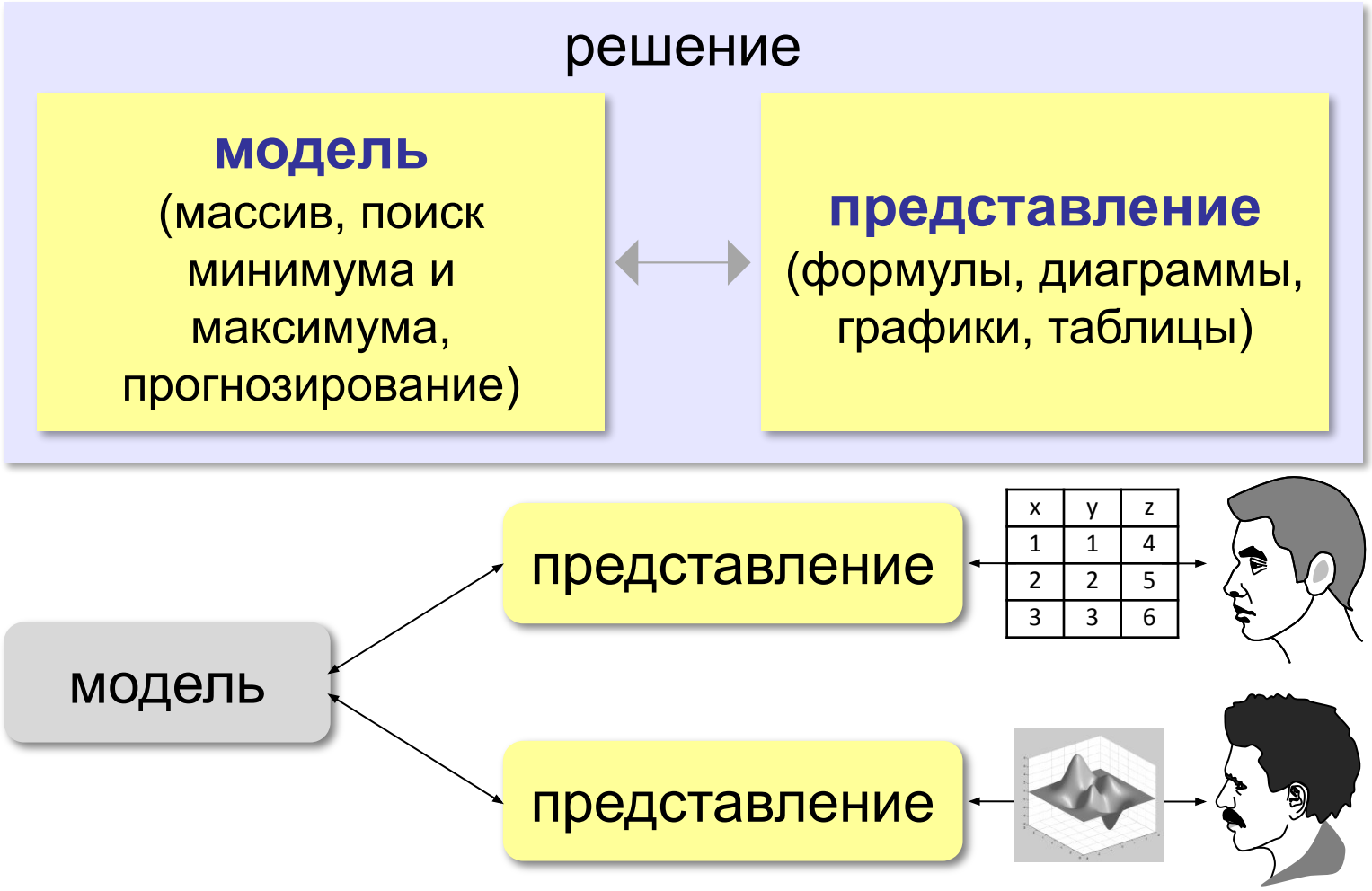
# Еще одна декомпозиция

*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 `+ - * /`

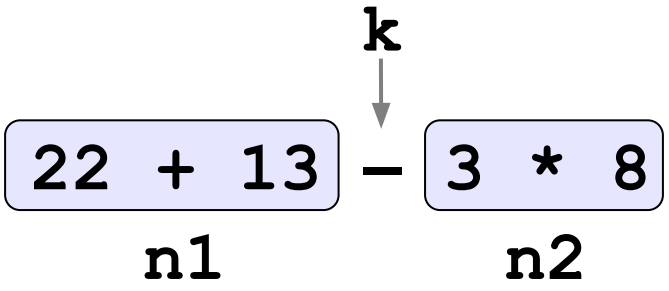
**Модель**:

- 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
если k = 0 то
  результат:= строка в число
иначе
  n1:= значение левой части
  n2:= значение правой части
  результат:= операция(n1, n2)
все
```

# Модель

```
function Calc ( s: string ): integer;
var k, n1, n2: integer;
begin
  k:= LastOp ( s );
  if k = 0 then Calc:= StrToInt(s)
  else begin
    n1:= Calc(Copy(s, 1, k-1));
    n2:= Calc(Copy(s, k+1, Length(s)-k));
    case s[k] of
      '+': Calc:= n1+n2;
      '-': Calc:= n1-n2;
      '*': Calc:= n1*n2;
      '/': Calc:= n1 div n2
    end
  end
end;
```

# Модель – в модуль

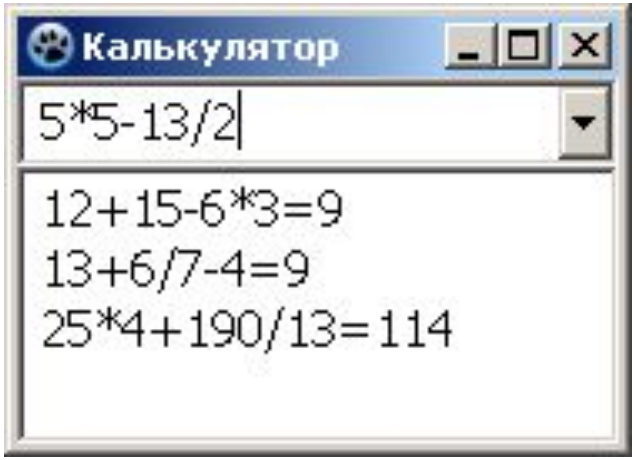
```
unit Model;
interface
  function Calc(s: string): integer;
implementation
uses SysUtils;
  function Priority(op: char): integer;
  ...
  function LastOpt(s: string): integer;
  ...
  function Calc(s: string): integer;
  ...
end.
```

для функции StrToInt

# Представление

выпадающий список **TComboBox**

**Name** = **Input**
**Align** = **alTop**




многострочное поле **TMemo**

**Name** = **Answers**
**Align** = **alClient**
**ReadOnly** = **True**

```
если нажата клавиша Enter то
  x:= значение выражения
  добавить результат в конец поля вывода
  если выражения нет в списке то
    добавить его в список
  все
все
```

# Перехват нажатия на клавишу Enter

OnKeyPress для элемента Input:

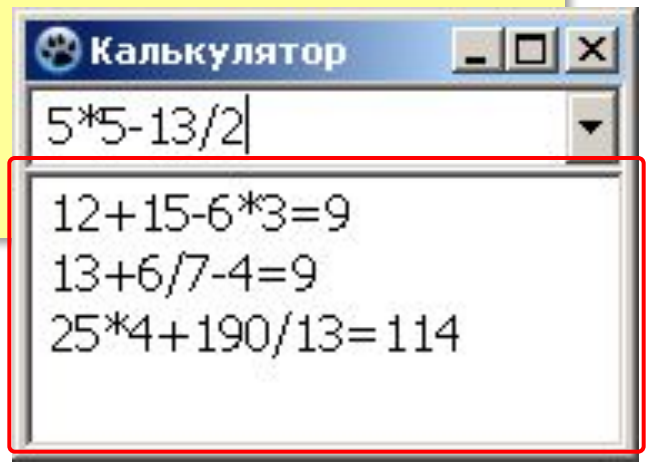
```
procedure TForm1.InputKeyPress(
            Sender: TObject; var Key: char);
begin
  if Key = #13 then begin
    ...
  end
end;
```

код клавиши Enter

# Обработка и вывод данных

**Вычисления** (обращение к модели):

```
uses Model;
...
x:=Calc(Input.Text);
```




**Добавление строки в TMemo**:

добавить строку

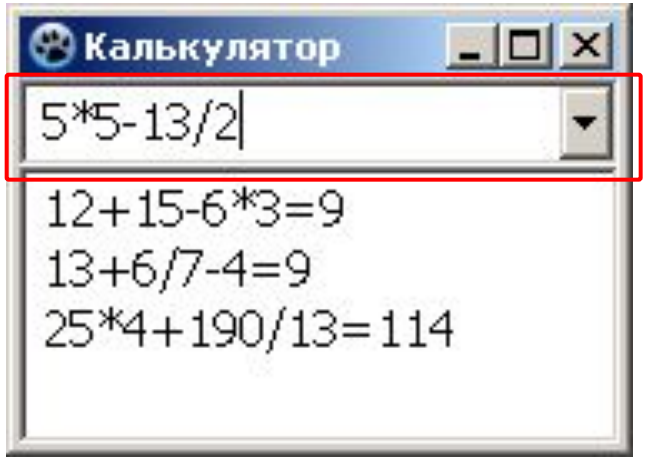
```
Answers.Lines.Add(Input.Text + '='
                  + IntToStr(x));
```

массив строк в TMemo

число в строку

# Обработка и вывод данных

## Добавление строки в TComboBox:




массив строк в TComboBox

найти индекс строки

```
i:= Input.Items .IndexOf(Input.Text);
if i < 0 then
        Input.Items.Insert(0, Input.Text)
```

добавить строку

позиция списка

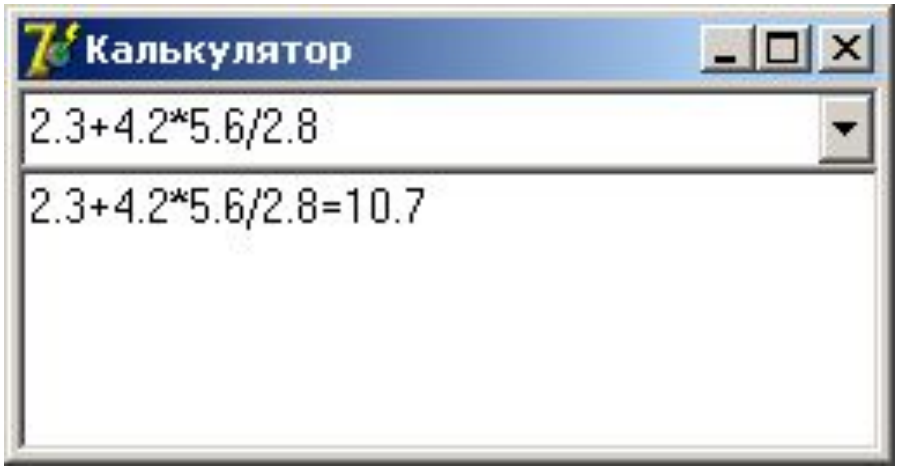
# Перехват нажатия на клавишу Enter

OnKeyPress для элемента Input:

```
procedure TForm1.InputKeyPress(
            Sender: TObject; var Key: char);
var x, i: integer;
begin
  if Key = #13 then begin
     x:=Calc(Input.Text);
     Answers.Lines.Add(Input.Text + '='
                              + IntToStr(x));
     i:=Input.Items.IndexOf(Input.Text);
     if i<0 then
        Input.Items.Insert(0,Input.Text)
  end
end;
```

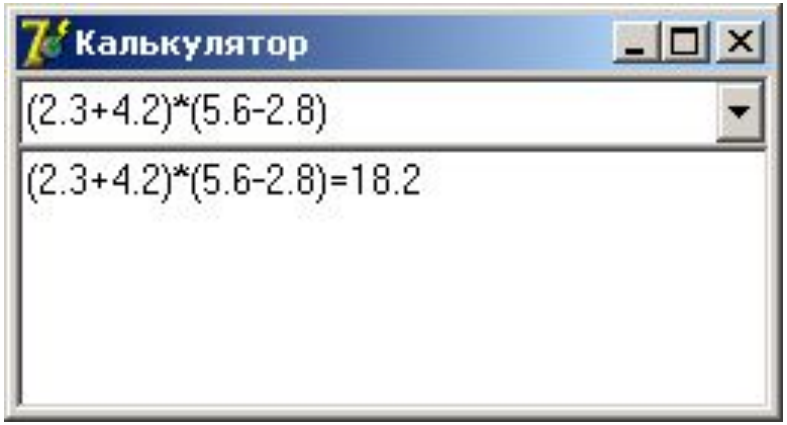
# Задание

**«А»:** Измените программу так, чтобы она могла вычислять значения выражений с вещественными числами.



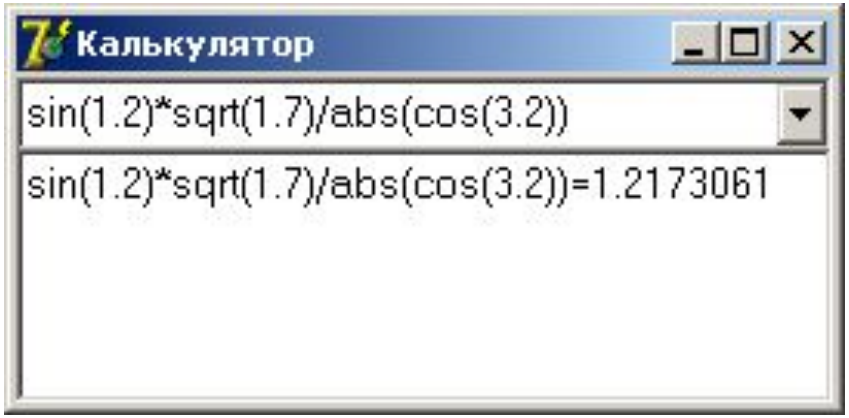
# Задание

**«B»:** Измените программу так, чтобы она могла вычислять значения выражений со скобками.



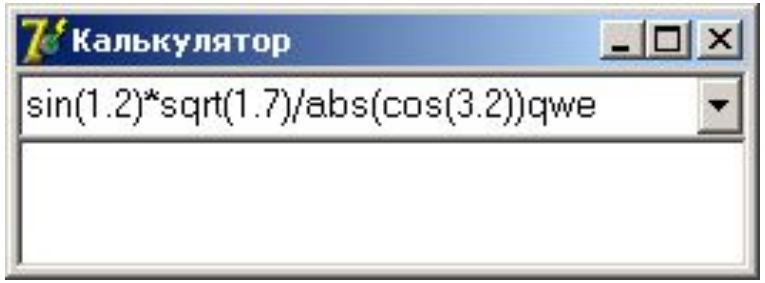
# Задание

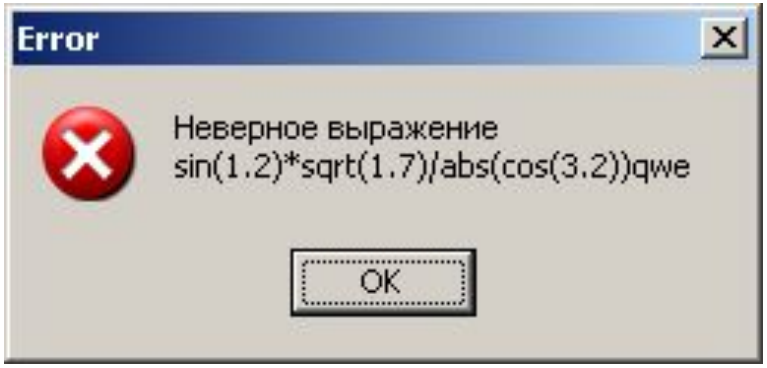
**«C»:** Измените программу так, чтобы она могла вычислять значения выражений, содержащих вызовы функций **`abs`**, **`sin`**, **`cos`**, **`sqrt`**.



# Задание

**«D»:** Измените программу так, чтобы при вводе неверного выражения выводилось сообщение об ошибке. :




Все результаты вычислений и сообщения об ошибках записываются в файл **results.txt**. Процедуру записи оформите как метод класса формы.

```
...
sin(1.2)*sqrt(1.7)=1.215230290196084
Неверное выражение sin(1.2)*sqrt(1.7)qwe
```

# Калькулятор


! Самостоятельно!

# Конец фильма

**ПОЛЯКОВ Константин Юрьевич**
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru

**ЕРЕМИН Евгений Александрович**
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru

# Источники иллюстраций

1. www.picstopin.com
2. maugav.info
3. yoursourceisopen.com
4. ru.wikipedia.org
5. иллюстрации художников издательства «Бином»
6. авторские материалы