

---

# ##### ##### # ##### Object Pascal ### #####

Michalis Kamburelis

## #####

|  |    |
|--|----|
| 1. #####: ### #### ## #####? .....                           | 2  |
| 2. ##### .....   | 4  |
| 2.1. ##### "Hello world!" .....                              | 4  |
| 2.2. #####, #####, ##### ##### .....                         | 5  |
| 2.3. ##### ##### (if) .....                                  | 7  |
| 2.4. ##### ##, ##### ##### # #####<br>(#####) ##### .....    | 8  |
| 2.5. ##### ##### ##### ## ##### (#####<br>case) .....        | 10 |
| 2.6. ##### # ##### ##, ##### # #####<br>##### .....          | 11 |
| 2.7. ##### (for, while, repeat, for .. in) .....             | 13 |
| 2.8. ##### ##### # ##### .....                               | 16 |
| 2.9. ##### # ##### ## .....                                  | 17 |
| 3. ##### (Unit-#) .....                                      | 19 |
| 3.1. ##### ##### ##### unit-### .....                        | 20 |
| 3.2. ##### ##### # ##### ##### unit-# .....                  | 21 |
| 3.3. ##### ##### ##### unit-# ##### .....                    | 24 |
| 4. ##### .....   | 26 |
| 4.1. ##### .....   | 26 |
| 4.2. ##### (Inheritance), ##### (is), # ##### ##### (as) ... | 26 |
| 4.3. ##### .....   | 29 |
| 4.4. ##### .....   | 33 |
| 4.5. ##### ##### .....                                       | 34 |
| 4.6. ##### ## ##### .....                                    | 35 |
| 5. ##### ##### .....   | 35 |
| 5.1. ##### ##### ##### ##### .....                           | 35 |
| 5.2. ##### ##### ##### .....                                 | 35 |
| 5.3. ##### # ##### ##### .....                               | 36 |
| 5.4. Free notification .....                                 | 40 |





## 2. #####

### 2.1. ##### "Hello world!"

```
## ##### ##### # ##### # ##### # ##### # #####  
##### "Hello world". ## ##### ## ##### #####:
```

---

```
{$mode objfpc}{$H+}{$J-} // Эту строку необходимо использовать во всех  
современных программах
```

```
program MyProgram; // Сохраните этот файл под названием myprogram.lpr  
begin  
  WriteLn('Hello world!');  
end.
```

---

### — #####, ##### # #####.

- ##### # ##### # ##### # ##### FPC, ##### # ##### # #####  
##### myprogram.lpr # ##### # ##### # ##### # ##### fpc  
myprogram.lpr .
- ##### ##### Lazarus, ## ##### # ##### # ##### (#  
##### #: Project → New Project → Simple Program). ##### ##  
## myprogram # ##### # ##### # ##### # #####. #####  
##### # ##### Run → Compile # #####.
- ##### # ##### # ##### # #####, ## # ##, # ##### # #####  
##### # ##### # ##### # ##### # ##### # #####  
##### # #####, ##### # ##### # #####.

```
##### # ##### # ##### # ##### # ##### # ##### # #####, #####  
## ##### # #####-##### # #####, ## # ##### # #####. ## # #####  
##### # # ##-##### # ##### "#####", ##### # ##### # ##### # ##### GUI  
##### # Lazarus (Project → New Project → Application). #####! — #####  
##### GUI # #####, # ##### # #####, ##### # #####  
##### # #####. Lazarus # Free Pascal Compiler # ##### # #####  
##### # ##### # #####, GUI, ## # #####, # ##### # ##### # #####  
##### # ##### (XML, json, #####...), ##### # ##### # #####,  
## # ##### # ##### # ##### # #####. ##### # ##### # #####  
Castle Game Engine, # ##### # ##### # ##### # ##### :)
```

## 2.2. #####, #####, #####

---

```
{ $mode objfpc } { $H+ } { $J- }
```

```
program MyProgram;
```

```
procedure MyProcedure(const A: Integer);
```

```
begin
```

```
  WriteLn('A + 10 составляет: ', A + 10);
```

```
end;
```

```
function MyFunction(const S: string): string;
```

```
begin
```

```
  Result := S + 'строки управляются автоматически';
```

```
end;
```

```
var
```

```
  X: Single;
```

```
begin
```

```
  WriteLn(MyFunction('примечание: '));
```

```
  MyProcedure(5);
```

```
  // деление с помощью оператора "/" всегда даёт результат с плавающей  
  запятой
```

```
  // для целочисленного деления необходимо использовать "div"
```

```
  X := 15 / 5;
```

```
  WriteLn('X составляет: ', X); // отобразить в научном формате вида
```

```
  3.0000000E+000
```

```
  WriteLn('X составляет: ', X:1:2); // отобразить 2 знака после запятой
```

```
end.
```

---

```
#####  
##### "#####" ##### Result # #####  
#####. ## ##### # #####  
##### # # #####
```

---

```
function MyFunction(const S: string): string;
```

```
begin
```

```
  Result := S + ' Добавим что-нибудь';
```

```
  Result := Result + ' и ещё что-нибудь!';
```

```
  Result := Result + ' И ещё немножко!';
```

```
end;
```

---

```
##### ##### # ##### Object Pascal ###
#####
```

---

```
##### (MyFunction # #####) ###
#####, ##### ##### #####. ##
##### # ## #####, ##### # #####
"#####", ##### # #####, ##### # #####
##### Result, # #####, #####
##### #####
```

```
#####, ##### #####. ## ##
##### # #####, ##### #####, ##### #
##### ##### ( ) (##### # ##### # #####
##### #####) - ##### ##### # #####
#####. #####:
```

---

```
function ReadIntegersUntilZero: string;
var
  I: Integer;
begin
  Readln(I);
  Result := IntToStr(I);
  if I <> 0 then
    Result := Result + ' ' + ReadIntegersUntilZero();
end;
```

---

```
##### Exit ##### ##### # #####
## ##, ## ## ##### end;. ## Exit #####
## #####, ## #####, ##### Result.
## ## ##### Exit(X), #####
##### # ##### — ##
##### return X # #-#####
```

---

```
function AddName(const ExistingNames, NewName: string): string;
begin
  if ExistingNames = '' then
    Exit(NewName);
  Result := ExistingNames + ', ' + NewName;
end;
```

---

```
#####, ## ##### # #####
"#####" (#####). ##### # #####
#####. ##### # ##### # #####
```

---

```
##### Object Pascal ###  
#####
```

---

```
##### (#####, #####), # ## #####  
#####. #####:
```

---

```
var
```

```
  Count: Integer;  
  MyCount: Integer;
```

```
function CountMe: Integer;
```

```
begin
```

```
  Inc(Count);  
  Result := Count;
```

```
end;
```

```
begin
```

```
  Count := 10;  
  CountMe; // результат функции будет отброшен, однако функция  
            выполняется, Count станет равен 11.  
  MyCount := CountMe; // запоминаем результат выполнения функции будет,  
  Count теперь 12.
```

```
end.
```

---

## 2.3. ##### (if)

```
##### if .. then ### if .. then .. else #####  
###, #####. # ##### ## C-##### #  
##### ##. #####.
```

---

```
var
```

```
  A: Integer;  
  B: boolean;
```

```
begin
```

```
  if A > 0 then  
    DoSomething;
```

```
  if A > 0 then  
    begin  
      DoSomething;  
      AndDoSomethingMore;
```

```
  end;
```

```
  if A > 10 then  
    DoSomething  
  else
```

```
##### ##### # ##### Object Pascal ###
#####
```

---

```
DoSomethingElse;
```

```
// идентично предыдущему примеру
```

```
B := A > 10;
```

```
if B then
```

```
    DoSomething
```

```
else
```

```
    DoSomethingElse;
```

```
end;
```

---

```
##### else ##### ##### # ##### ##### if. #####
##### ##### ##### ## ##### ##### ##### ##:
```

---

```
if A <> 0 then
```

```
    if B <> 0 then
```

```
        AIsNonzeroAndBToo
```

```
    else
```

```
        AIsNonzeroButBIsZero;
```

---

```
#####, ##### if ##### begin ... end;
#####, ## #####, ##### ##
#####, ##### ## ##### ## ##### ## #
#####. #####, ##### #
if ##### else - ##### A ## ##### B - #,
#####, ##### ##### ##### ## ##### ##
## ## #####.
```

---

```
if A <> 0 then
```

```
begin
```

```
    if B <> 0 then
```

```
        AIsNonzeroAndBToo
```

```
    else
```

```
        AIsNonzeroButBIsZero;
```

```
end;
```

---

## 2.4. #####, ##### # ##### (#####) #####

```
##### and, or, not,
xor. ## ##### # ##### ##### ##
```

---



---

#####. #####, ## ##### xor, #####  
# ##### "#####". ## #####  
##### (boolean), # ##### ##  
## ## boolean. ## ##### # #####, # #####,  
#### ## ##### (integer, byte ##), # ##### ##  
#####.

##### (#####) - #####  
#####: =, <>, >, <, <=, >=, #####. #####  
#####, ## # ##### #-##### #, #####  
##### "#####" A = B (# # #, ##  
##### A == B). ##### #  
##### :=.

##### (#####) #####, ##  
#####. #####, #####  
##### # #####  
#####.

#####:

---

```
var
  A, B: Integer;
begin
  if A = 0 and B <> 0 then ... // так делать НЕЛЬЗЯ
```

---

##### and # ##### (0  
and B) - #####. #####  
##### "#####" # ##### A =  
(0 and B). # # ##### type mismatch, # #####  
##### A = (0 and B) # #####  
##### 0.

#####:

---

```
var
  A, B: Integer;
begin
  if (A = 0) and (B <> 0) then ...
```

```
##### Object Pascal ###  
#####
```

```
##### "##### (short-circuit evaluation)" - #####  
#####, ##### ## #####, #####-####  
### ##### #####. #####:
```

```
if MyFunction(X) and MyOtherFunction(Y) then...
```

- ##### MyFunction(X) #####.
- # ##### MyFunction(X) ##### false, ## ## ##  
##### - ##### ## ## #####, ##  
##### false and что-нибудь ## ##### false. #####  
##### MyOtherFunction(Y) #####.
- ##### or. # #####, ## ## ##  
#####, ## ##### true ##### ## ## ##  
##### true, ##### ## ## ## ## ## ##  
#####.
- ### ##### ##### ##### #####:

```
if (A <> nil) and A.IsValid then...
```

```
# ##### ## ##### # ##### # ##### # A  
##### nil. ##### nil ## #####, ##### #  
##### ## "#####" #####. ## #####  
##### (null pointer).
```

## 2.5. ##### (##### case)

```
### # ##### ## ##### #####  
### #####, ##### #####  
case .. of .. end.
```

```
case SomeValue of  
  0: DoSomething;  
  1: DoSomethingElse;  
  2: begin  
      IfItsTwoThenDoThis;  
      AndAlsoDoThis;  
  end;
```



```
#####  ##### # ##### Object Pascal ###  
#####
```



```
#####  #####  ##### #####  ##### #  
#####  ##### # #####  #####  
{$scopedenums on}. # #####  #####  #####  
#####  ##### # ### #####  #####  
#####: TAnimalKind.akDuck. # #####  #####  
##### # ##### ak, # #####  #####  
#####  ##### Duck, Cat, Dog. #####  #####  
#####  ####, ### #####  ##### # C#.
```

```
#####  #####  #####, ### ## ###  #####  
# #####. ## # ####, ### #####  
#####, #####  Ord(MyAnimalKind), #####  
##### # #####. #####  #####  
### #####  TAnimalKind(MyInteger) # #####  ##### #  
#####  ##### ##. # #####  #####  
### #####, ### MyInteger #####  ##### ## 0 ##  
Ord(High(TAnimalKind))).
```

```
#####  #####  #####  #####  ##### # #####  #####  
#####  #####:
```

---

**type**

```
TArrayOfTenStrings = array [0..9] of string;  
TArrayOfTenStrings1Based = array [1..10] of string;  
  
TMyNumber = 0..9;  
TAlsoArrayOfTenStrings = array [TMyNumber] of string;  
  
TAnimalKind = (akDuck, akCat, akDog);  
TAnimalNames = array [TAnimalKind] of string;
```

---

```
### #####  #####  ##### (### ## set-#, ### ##  
#####  #####):
```

---

**type**

```
TAnimalKind = (akDuck, akCat, akDog);  
TAnimals = set of TAnimalKind;
```

**var**

```
A: TAnimals;
```

**begin**

```
A := [];  
A := [akDuck, akCat];
```

```
A := A + [akDog];
A := A * [akCat, akDog];
Include(A, akDuck);
Exclude(A, akDuck);
end;
```

---

## 2.7. ##### (for, while, repeat, for .. in)

---

```
[$mode objfpc}{$H+}{$J-}
{$R+} // включаем проверку на диапазон величин, очень полезно для отладки
var
  MyArray: array [0..9] of Integer;
  I: Integer;
begin
  // инициализация
  for I := 0 to 9 do
    MyArray[I] := I * I;

  // отображение
  for I := 0 to 9 do
    WriteLn('Квадрат составляет ', MyArray[I]);

  // делает то же самое, что и предыдущий вариант
  for I := Low(MyArray) to High(MyArray) do
    WriteLn('Квадрат составляет ', MyArray[I]);

  // делает то же самое
  I := 0;
  while I < 10 do
  begin
    WriteLn('Квадрат составляет ', MyArray[I]);
    I := I + 1; // это идентично "I += 1" или "Inc(I)"
  end;

  // делает то же самое
  I := 0;
  repeat
    WriteLn('Квадрат составляет ', MyArray[I]);
    Inc(I);
  until I = 10;

  // делает то же самое
  // обратите внимание, тут переменная I перечисляет значения элементов
  массива, а не его индексы
```

---



```
##### ##### # ##### Object Pascal ###  
#####
```

---

```
var  
  Animals: TAnimals;  
  AK: TAnimalKind;  
begin  
  Animals := [akDog, akCat];  
  for AK in Animals do ...
```

---

```
# # ### ## ##### ## ##### #####, ##### generic-#,  
#####, TObjectList ### TFPGObjectList.
```

---

```
{$mode objfpc}{$H+}{$J-}
```

```
uses  
  SysUtils, FGL;
```

```
type  
  TMyClass = class  
    I, Square: Integer;  
  end;  
  TMyClassList = specialize TFPGObjectList<TMyClass>;
```

```
var  
  List: TMyClassList;  
  C: TMyClass;  
  I: Integer;  
begin  
  List := TMyClassList.Create(true); // значение true означает, что  
  List владеет всеми дочерними объектами  
  try  
    for I := 0 to 9 do  
      begin  
        C := TMyClass.Create;  
        C.I := I;  
        C.Square := I * I;  
        List.Add(C);  
      end;  
  
      for C in List do  
        WriteLn('Квадрат ', C.I, ' составляет ', C.Square);  
      finally  
        FreeAndNil(List);  
      end;  
  end.
```

---

```
##### # # ##### Object Pascal ##  
#####
```

---

```
## ## # #####, #####  
#####. ## # #####  
##### :)
```

## 2.8. ##### # #####

```
## ##### # ##### Write ##  
WriteLn. ## ##### # #####  
#####.
```

```
## "#####" # #####, ## #####  
#####, ##### #####  
##### # ##### ## ##  
#####. ##### ##,  
##### ##### ##  
#####.
```

---

```
WriteLn('Hello world!');  
WriteLn('Можно вывести целое число: ', 3 * 4);  
WriteLn('Отформатировать его: ', 666:10);  
WriteLn('А также вывести число с плавающей запятой: ', Pi:1:4);
```

---

```
#####  
LineEnding (## ##### FPC RTL). (Castle Game Engine #####  
##### NL). # ##### ## HTML # #####  
##### (\) ## #####  
##### # #####, #####:
```

---

```
WriteLn('Первая строка.\nВторая строка.');
```

---

```
## ##### # ##### ## ##### # #####  
#####. #####:
```

---

```
WriteLn('Первая строка.' + LineEnding + 'Вторая строка.');
```

---

```
## ##:
```

---

```
WriteLn('Первая строка.');
```

```
WriteLn('Вторая строка.');
```

---



```
##### # ##### # ##### Object Pascal ###
#####
```

```
#### #, ## # Write/WriteLn ##### #
# #####. ## ##### {Sapptype
CONSOLE} (## ## {Sapptype GUI}) # ##### # (Unix) # ## #
#####. # # ##### Write/WriteLn
## GUI ##### # ##### (#, # Windows).
```

```
# Castle Game Engine ## ##### WriteLn, ##### #
##### # WriteLnLog ### WriteLnWarning ###
##### #. ## #
#####: ## Unix-##### # #
# #####. ## Windows GUI ##### ## #. # Android #####
##### # Android logging facility (#####), #####
##### # adb logcat. ##### WriteLn
#### # #, ###
##### (##### #) # ##### #
#####, ## #. ####, ## #
##### ## #, ##### #
##### # #####.
```

## 2.9. ##### # ##### #

```
## # (#####
###, ##### # #) ####
#####.
```

- ##### # IntToStr # FloatToStr. # ##### (concatenation) ##### #: 'Моё целое число ' + IntToStr(MyInt) + ', а значение числа пи составляет ' + FloatToStr(Pi).
- # #####. ## #####. ##### #
 ### XxxToStr # ## ##### (#####, FormatFloat), ## #
 ##### #.
- # #####. ## ##### # #####. #####
 ##### (#####, #####) # ## #
 ##### # StrToInt, StrToFloat # #####
 (#####, StrToIntDef).

##### `Object Pascal` ###  
#####

---

# #####. ##### `XxxToStr` # #####  
#####.

- ##### `Format` ##### # ##### `Format('%d %f %s', [MyInt, MyFloat, MyString])`. ### ##### `sprintf` # C-#####  
#####. ### ##### # ##### placeholder-# #####  
#####. ### placeholder-# #####  
#####, ##### # #####, ##### `%.4f` ### #####  
##### # 4 #####

# #####. ##### ## ##### #  
#####. ##### #, ## #, #,

# #####. ## "#####"  
#####. ##### #  
##### #  
##### (### #####  
##### `array of const`). #####  
##### `Format`, ## # # # #  
##, ## #.

# #####. ##### #, #####-#####  
#####. ##### # placeholder-# #  
`EConvertError`, #, # #, # segmentation  
fault (##### `SIGSEGV`).

- # ##### # ##### `WriteStr(TargetString, ...)` ##  
##### `Write(...)`, # ##### - #####  
##### # `TargetString`.

# #####. ## # `Write`, #  
## # "#####", ##  
##### `Pi:1:4`.

# #####. ##### #  
"#####", #. #. ## #  
##### #, #, #  
##### `MyStringFormatter(...)` ## #  
#####, ## `Pi:1:4`. ##### #, # ##-# #, # #  
##### # # # # #, #

### 3. ##### (Unit-#)

Unit-# #####  
##### (#####  
#####), ##### unit-### #  
#####. ##### # #####. #####  
##### interface, ##### unit-## #  
##### # #.#, # implementation, #####, #####  
#####. Unit MyUnit ##### myunit.pas (#####  
##### .pas).

```
.....  
{$mode objfpc}{$H+}{$J-}  
unit MyUnit;  
interface  
  
procedure MyProcedure(const A: Integer);  
function MyFunction(const S: string): string;  
  
implementation  
  
procedure MyProcedure(const A: Integer);  
begin  
  WriteLn('A + 10 составляет: ', A + 10);  
end;  
  
function MyFunction(const S: string): string;  
begin  
  Result := S + 'строки управляются автоматически';  
end;  
  
end.
```

#####  
myprogram.lpr ( lpr = Lazarus program file; # Delphi ##### .dpr ).  
##### # ##### # ##### # ##### # ##### # #####  
##### .pas ##### # #####. #####  
unit-## ##### .pp. ##### # # #####  
##### .pas # unit-## # .lpr # FPC/Lazarus #####.  
  
##### unit # ##### uses :

```
.....  
{$mode objfpc}{$H+}{$J-}
```

```
program MyProgram;  
  
uses MyUnit;  
  
begin  
  WriteLn(MyFunction('Примечание: '));  
  MyProcedure(5);  
end.
```

---

Unit ##### # initialization # finalization. # ##  
##### #, ##### # ##  
#####, #####.

---

```
{ $mode objfpc } { $H+ } { $J- }  
unit initialization_finalization;  
interface  
  
implementation  
  
initialization  
  WriteLn('Hello world!');  
finalization  
  WriteLn('Goodbye world!');  
end.
```

### 3.1. ##### unit-###

```
## ##### #, ## # unit-# ##### # ##  
##### unit-#. ##### unit ##### # ##### interface ## # ##  
implementation. ##### # ##  
(#####, ####...), ##### # ## unit-#.  
## ##### # - ##### unit  
# ##### implementation, ## # ##  
##### implementation ##### unit-#.
```

---

```
{ $mode objfpc } { $H+ } { $J- }  
unit AnotherUnit;  
interface  
  
uses  
  Classes;
```

```
##### ##### # ##### Object Pascal ###
#####
```

```
{ Тип класса "TComponent" определён в unit-e Classes.
По этому необходимо использовать uses Classes, как видно выше. }
procedure DoSomethingWithComponent(var C: TComponent);
```

```
implementation
```

```
uses
```

```
    SysUtils;
```

```
procedure DoSomethingWithComponent(var C: TComponent);
```

```
begin
```

```
    { Процедура FreeAndNil определена в unit-e SysUtils.
Поскольку мы лишь ссылаемся на её имя в разделе implementation,
вполне допустимо использовать SysUtils в секции "implementation". }
    FreeAndNil(C);
```

```
end;
```

```
end.
```

```
##### ##### ##### ##### (cyclic reference) #
##### interface. #. #. ### unit-# ## ##### ##### #
##### interface. ##### ##### ##### # ###, ##
### ##, ##### "#####" ##### interface unit-#, ##### #
"#####" ### unit-#, ##### # uses # ##### interface. # ##### #
##### #, ### ##### #
##### # ##### #
##### #####. # ##### #
##### Makefile ### ##### #, # #
##### #, ##### #
#####.
```

```
##### ##### ##### ##### unit-### ##### #
## ## "#####" ##### # implementation. ##### # A
##### B # interface, # ##### unit B ##### A # implementation.
```

### 3.2. ##### # ##### unit-#

```
##### unit-# ##### ##### #
##### # # # # # #, ##### #
##### # # # # # #. # ##### # # # # # # # #
unit # ##### uses "##### # # # # # #", #. #.
```



```
##### ##### # ##### Object Pascal ###  
#####
```

---

```
// НЕВЕРНЫЙ пример
```

```
interface
```

```
uses Graphics;
```

```
procedure ShowColor(const Color: TColor);
```

```
implementation
```

```
uses GoogleMapsEngine;
```

```
procedure ShowColor(const Color: TColor);
```

```
begin
```

```
    // WriteLn(ColorToString(Color));
```

```
end;
```

```
end.
```

---

```
Unit Graphics (## ##### ##### Lazarus LCL) ##### ## TColor.  
## ##### ##### ## ##### # ##### unit-#, ##### ## ##, ##  
##### # ##### Interface ##### ShowColor ## #####. #####  
# ##, ## unit GoogleMapsEngine ##### ##### ## TColor,  
##### ##### # ##### implementation, #####  
## ##### TColor # ##### implementation. ##.  
##### ##### ## ##### ##:
```

---

```
{$mode objfpc}{$H+}{$J-}
```

```
unit UnitUsingColors;
```

```
// НЕВЕРНЫЙ пример
```

```
// демонстрирующий, как предыдущий пример "видит" компилятор
```

```
interface
```

```
uses Graphics;
```

```
procedure ShowColor(const Color: Graphics.TColor);
```

```
implementation
```

```
uses GoogleMapsEngine;
```

```
##### ##### # ##### Object Pascal ###
#####
```

---

```
procedure ShowColor(const Color: GoogleMapsEngine.TColor);
begin
  // WriteLn(ColorToString(Color));
end;

end.
```

---

```
# ##### ##### ##### #####: ##### ##### #####
implementation, ##### ##### ##### TColor ## unit-# Graphics.###
##### ##### #####, ##### GoogleMapsEngine # ##### interface ##
unit-# Graphics. #####, ### ##### # ##### #####
unit-# UnitUsingColors, ### ### ##### ##### ## #####.
```

---

```
{$mode objfpc}{$H+}{$J-}
```

```
unit UnitUsingColors;
```

```
interface
```

```
uses Graphics;
```

```
procedure ShowColor(const Color: TColor);
```

```
implementation
```

```
uses GoogleMapsEngine;
```

```
procedure ShowColor(const Color: Graphics.TColor);
```

```
begin
```

```
  // WriteLn(ColorToString(Color));
```

```
end;
```

```
end.
```

---

### 3.3. ##### unit-# #####

```
##### ##### ##### ##### ##### unit-# #
##### ## ##### unit. #.#. # ##### ##### unit-#
##### ##### ##### ##### # #####.
```

```
# ##### ##### ## ##### #-## ##### #####
##### # ##### unit-#. # ##### #####
"#####" #####-#### unit ### #####.
```



```
##### # ##### # ##### Object Pascal ###
#####
```

---

```
### ##### # ##### # ##### # ##### # #####
##### # ##### unit-#.
```

---

```
{$mode objfpc}{$H+}{$J-}
```

```
unit MyUnit;
```

```
interface
```

```
uses Graphics;
```

```
type
```

```
{ Используем TColor из unit-a Graphics для определения TMyColor. }
```

```
TMyColor = TColor;
```

```
{ Как вариант, можно переопределить его под тем же именем.
```

```
В таком варианте необходимо будет явно указать наименование unit-a,
иначе получится несогласованное определение "TColor = TColor". }
```

```
TColor = Graphics.TColor;
```

```
const
```

```
{ С константами это тоже работает. }
```

```
clYellow = Graphics.clYellow;
```

```
clBlue = Graphics.clBlue;
```

```
implementation
```

```
end.
```

---

```
##### ######, ### ##### # # ##### # ###### ###### ######,
##### # ######. # ##### ###### ###### ###### ######
##### ###### # ###### # ###### unit-# (##. ##### 7.2, «#####
—### ## ######, ### ## ###### ## ######, ### ## ######
#####»), ## ##### ## ###### ## ###### #####.
```

```
##### ###### ###### ###### ###### ###### "#####-
#####", ##### ## ###### ###### ###### ## ###### unit-#,
##### ###### ## ###### # ###### ###### ###### ######.
```

```
##### ###### ## ## # ###### ###### ###### ###### ######
##### (## ###### unit-#) ######, ##. ##### 4.3, «#####».
```

## 4. #####

### 4.1. #####

# ##### (classes). ## #####  
"#####", ##### # ##:

- ##### (field) (##### "#####"),
- ##### (method) (##### "#####"),
- ##### (property) (##### #, ##### #, ##### # (getter) # (setter) #####-#####; #. ???).
- ##### #, # ##### #, #. ##### 8.2, «##### # #».

---

#### type

```
TMyClass = class
  MyInt: Integer; // это "поле"
  property MyIntProperty: Integer read MyInt write MyInt; // это
"свойство"
  procedure MyMethod; // это "метод"
end;
```

```
procedure TMyClass.MyMethod;
begin
  WriteLn(MyInt + 10);
end;
```

---

### 4.2. ##### (Inheritance), ##### (is), # ##### ##### (as)

##### # #####-  
#####.

---

```
[$mode objfpc]{$H+}{$J-}
program MyProgram;
```

**uses**

SysUtils;

**type**

TMyClass = **class**

MyInt: **Integer**;

**procedure** MyVirtualMethod; **virtual**;

**end**;

TMyClassDescendant = **class**(TMyClass)

**procedure** MyVirtualMethod; **override**;

**end**;

**procedure** TMyClass.MyVirtualMethod;

**begin**

WriteLn('TMyClass отображает MyInt + 10: ', MyInt + 10);

**end**;

**procedure** TMyClassDescendant.MyVirtualMethod;

**begin**

WriteLn('TMyClassDescendant отображает MyInt + 20: ', MyInt + 20);

**end**;

**var**

C: TMyClass;

**begin**

C := TMyClass.Create;

**try**

C.MyVirtualMethod;

**finally**

FreeAndNil(C);

**end**;

C := TMyClassDescendant.Create;

**try**

C.MyVirtualMethod;

**finally**

FreeAndNil(C);

**end**;

**end**.

---

```
## ##### ##### ## ##### ##### # ## ##### ##  
##### ##### ##### ##### ##### virtual.#####  
## ##### ##### ##### ##### # ##### #####
```

```
##### # ##### # ##### Object Pascal ###  
#####
```

---

```
##### override, ##### #####. ##### #####  
##### reintroduce, #####, #####  
## ## #####, ## ##### ## #####.
```

```
##### ## ##### ## #####  
### ##### is. ###  
##### # #####  
as.
```

---

```
{ $mode objfpc } { $H+ } { $J- }
```

```
program is_as;
```

```
uses
```

```
  SysUtils;
```

```
type
```

```
  TMyClass = class
```

```
    procedure MyMethod;
```

```
  end;
```

```
  TMyClassDescendant = class(TMyClass)
```

```
    procedure MyMethodInDescendant;
```

```
  end;
```

```
procedure TMyClass.MyMethod;
```

```
begin
```

```
  WriteLn('Это MyMethod')
```

```
end;
```

```
procedure TMyClassDescendant.MyMethodInDescendant;
```

```
begin
```

```
  WriteLn('Это MyMethodInDescendant')
```

```
end;
```

```
var
```

```
  Descendant: TMyClassDescendant;
```

```
  C: TMyClass;
```

```
begin
```

```
  Descendant := TMyClassDescendant.Create;
```

```
  try
```

```
    Descendant.MyMethod;
```

```
    Descendant.MyMethodInDescendant;
```

```
  { производные классы сохраняют все функции родительского класса }
```

```
##### # # ##### Object Pascal ###
#####
```

---

```

    TMyClass, по этому можно таким образом создавать ссылку }
C := Descendant;
C.MyMethod;

{ так не работает, поскольку в TMyClass не определён этот метод }
//C.MyMethodInDescendant;
{ правильно записать следующим образом: }
if C is TMyClassDescendant then
    (C as TMyClassDescendant).MyMethodInDescendant;

finally
    FreeAndNil(Descendant);
end;
end.

```

---

```
##### # # # X as TMyClass, #####
##### # # # ##### # ##### ##### TMyClass(X). #####
### ##### #####-### #####, ## ##### # #####
##### # ##### # X ## ##### ##### TMyClass. ## #####
##### TMyClass(X) ##### # ##### #####, #####
##### #####, ## X ##### ##### ##### TMyClass,
#####, ##### ## ##### # ##### # ##### is:
```

---

```

if A is TMyClass then
    (A as TMyClass).CallSomeMethodOfMyClass;
// вариант ниже - работает незначительно быстрее
if A is TMyClass then
    TMyClass(A).CallSomeMethodOfMyClass;

```

---

### 4.3. #####

```
##### (Properties) ##### "#####" (###. #####. syntax
sugar - #####, #####, #####)
## ##### # #####, ## #####
##### ## #####) ##### # #####:
```

1. ##### #-## ##### # ## ## (##### # #
#####), ## ##### # #####
(getter) # ##### (setter). #####
##### - ##### # #####
##, ##### # #####

```
##### # ##### # ##### Object Pascal ###
#####
```

2. ##### #-## ##### ##### ## ####, ## ##### ##### -  
#####, ##### ## ##### ##### ##### ## #####.

**type**

```
TWebPage = class
```

```
private
```

```
FURL: string;
```

```
FColor: TColor;
```

```
function SetColor(const Value: TColor);
```

```
public
```

```
{ Значение URL невозможно установить напрямую.
```

```
Следует вызвать метод вроде Load('http://www.freepascal.org/'),  
для загрузки страницы и установки значения этого свойства. }
```

```
property URL: string read FURL;
```

```
procedure Load(const AnURL: string);
```

```
property Color: TColor read FColor write SetColor;
```

```
end;
```

```
procedure TWebPage.Load(const AnURL: string);
```

```
begin
```

```
FURL := AnURL;
```

```
NetworkingComponent.LoadWebPage(AnURL);
```

```
end;
```

```
function TWebPage.SetColor(const Value: TColor);
```

```
begin
```

```
if FColor <> Value then
```

```
begin
```

```
FColor := Value;
```

```
{ Например, требовать обновления класса, каждый раз,  
когда изменяется значение его цвета:
```

```
Repaint;
```

```
{ Ещё пример: обеспечить чтобы нечто изменялось синхронно  
с установкой цвета, например }
```

```
RenderingComponent.Color := Value;
```

```
end;
```

```
end;
```

```
##### #####, ## ##### ####, ##### ##### ##### ## #####  
## #####, ##### ##### ##### #####/##### ##### (#####  
##### ##### private # ##### ##### ##### ##### ##### property #  
##### ##### f (## field - ####) ##### #####  
## #####. # ##### ####, ##### Color #####
```

---

setter-##### SetColor . #####, ### ##### ##### ##### Color  
##### ##### ## private ##### FColor . ##### ##### ## #####  
####, #####, #####, ### ##### ##### ##### getter ##  
setter - ## # ##### ##### #####, ## # # ##### #####  
#####.

### #####:

1. ##### ## ## ##### #####, # ## (# ##### ##### ##### ##, ## #  
##### getter ).
2. ##### ## ## ##### #####, # ## (# ##### ##### ##### ##, ##  
##### setter ).

##### ##### ## ##, ##### ## # ##### ##### ##### #  
##### ##### # ##### ##### # ##### ## #####. #####, #####  
##### ##### Integer ##### ## ##### ##### Integer , ##  
##### ##### (#####), ##### ##### Integer .

# ##### ##### ## ##, ##### "getter" # "setter" - ##### ##### # ##  
##### ##### ## ##### (##### ##### #####).  
## # ##### ##### ## ##### ##### ## ##, #####  
##### ##-##### ##### ##:

- ##### getter ## ##### ##### ##### ##### (#####,  
## # ##### ##### ##### ## # / # #####). ##  
##### ##### ##### (## ##### ##-  
##### :)) - ##### ##### ##### ## ##  
##### #####, ##### ##### ##  
#####.

##### #####, ## ##### ##### ##### ##### getter ##### ##  
##### #####, #####, ##### # ##### ##-  
##### ## ##### ##### ## ## ##### ##  
#####, ## ## ## ##### ##### ##### "getter".

- ##### setter ##### ##### ##### ##### ## ##, #####  
getter ##### ## # #####. ## ##### #####  
##### "setter", # # #####, ##### ## #####  
exception. ##### ## ##### ##  
##### ##. ##### ## # ##, ##### ##

##### Object Pascal ###  
#####

---

```
##### MyClass.MyProperty := 123; #####  
#####, ### MyClass.MyProperty = 123.
```

- #####, ##### ##### ### ##### ##### ##### ##### #####  
##### ##### ##### ## ##### ##### ##### #####. #####  
#####, ##### ##### - ##### ## ##### ##### ## #####, ##  
##### ##### ##### ##### # ##### ##### #####.  
##### ##### ##### ## ##### ##### #####. #####  
##### ##-## #####, ##### #####, # ## #####.
- #####, # ##### ##### ##### ##### ##### ##### # #####  
private, ##### ##### ##### ##### - ##### ##### # #####  
##### # #####.
- #####, ##### ##### #####, ##### ##### #####  
#####, ## ## ##### ##. #####, ##### ##### #####  
##### ## ## ## ##### :)



```
##### ## ## ##### ##### ##### ## #####, ## #####  
unit-#. ## ##### ##### #####: ## ##### #####  
## ##### #####, ## ##### # ## #####  
##### ##### getter # setter.
```

### ##### (Serialization) #####

```
#####, ##### ##### Published ##### ##### ## #####  
(serialization) (##### ##### ##### # ##### (streaming  
components)) # #####. ##### "Serialization" ##### ## ##### "Series" - "###",  
#.#. ##### ##### ##### # ##### ##### #####  
#####, ##### ## #####, ##### # ##### ## # #####.
```

```
#####, ##### ##### # ##### # #####, ##### Lazarus  
##### ## ##### ##### ##### ##/# ##### xxx.lfm. #  
Delphi ##### ##### ##### .dfm. #####  
##### # # ##### #, # ##### ##### ##  
ReadComponentFromTextStream ## unit-# LResources. #####  
##### ##### #####, ##### unit FpJsonRtti  
##### ##### # ##### JSON.
```

```
# Castle Game Engine ##### unit CastleComponentSerialize  
(##### ## ##### FpJsonRtti) ## #####  
#####, ##### ## user-interface # transformation.
```



```
##### # ##### # ##### Object Pascal ###
#####
```

---

```
### ##### ##### ##### #####, ##### #####
##### ### ##### ##### #####:
```

- ##### ##### "## #####" # ##### ##### #####  
default. #####, ## ## ##### #####  
##### # ##### ##### ## ##### - ## ## #####  
#####. ##### default ## ## #####  
#####: "##### ##### constructor-#, ## #####  
##### ## #####".
- ##### ## ## ##### - # ##### ##### ##### stored.

#### 4.4. #####

```
# ##### ##### # #####. ## ##### "#####" #  
##### try ... except ... end, ##### #####  
"##### # #####" try ... finally ... end.
```

---

```
{$mode objfpc}{$H+}{$J-}
```

```
program MyProgram;  
  
uses  
  SysUtils;  
  
type  
  TMyClass = class  
    procedure MyMethod;  
  end;  
  
procedure TMyClass.MyMethod;  
begin  
  if Random > 0.5 then  
    raise Exception.Create('Вызываем exception!');  
end;  
  
var  
  C: TMyClass;  
begin  
  Randomize;  
  C := TMyClass.Create;  
  try  
    C.MyMethod;
```

##### **Object Pascal** ###  
#####

---

**finally**  
    FreeAndNil(C)  
**end;**  
**end.**

##### **finally** #####  
##### **Exit** (#####)  
##### **Break** ##### **Continue** (#####).

### 4.5. #####

##### **public** #####  
##### **private** #####  
##### **protected** #####.

**public**

##### **private** #####

**private**

#####

**protected**

#####

##### **private** # **protected**, #####  
##### **unit-** #####  
##### **private** # **protected**.#####  
##### **strict private** ## **strict protected** ##  
##### **8.1**,  
«##### **private** # **strict private**».

## ##### **public**.#####  
##### **TPersistent**, #####  
##### **TComponent**, ##### **TPersistent**. ##  
##### **published**, #####

```
##### Object Pascal ###
#####
```

---

```
##### public, ##### (stream) #####.
```

```
#####, ## ##### published - ## ##### (stream), # #####, #####, #####. ### #####, ## ##### -## ##### ## #####, ## ##### public.
```

#### 4.6. #####

```
#### ##### class ##### TObject.
```

### 5. #####

#### 5.1. #####

```
#####, ## ##### FPC -gl -gh, ##### (#####) ##. https://castle-engine.io/manual\_optimization.php#section\_memory).
```

```
##### (raised exceptions). ## ##### (# #####, ## #####) # #####), #####. #####.
```

#### 5.2. #####

```
##### FreeAndNil(A) ## unit-# SysUtils ##### nil, # ##### — ##### (destructor), # A # nil. #####. #####.
```

```
#####:
```

```
if A <> nil then
begin
```



```
##### # ##### # ##### Object Pascal ###
#####
```

---

```
##### ##### nil, # #####
##### ##, ##### FreeAndNil(A).
```

```
#####:
```

---

```
uses SysUtils;
```

```
type
```

```
  TGun = class
  end;
```

```
  TPlayer = class
    Gun1, Gun2: TGun;
    constructor Create;
    destructor Destroy; override;
  end;
```

```
constructor TPlayer.Create;
```

```
begin
```

```
  inherited;
  Gun1 := TGun.Create;
  Gun2 := TGun.Create;
```

```
end;
```

```
destructor TPlayer.Destroy;
```

```
begin
```

```
  FreeAndNil(Gun1);
  FreeAndNil(Gun2);
  inherited;
```

```
end;
```

---

```
##### ##### ##### ##### ## ##### #####
##### ##, ##### ##### ##### TComponent,
##### ##### "owner" (#####). ##### #
#### owner(#####) ##### ## owner-##. #####
##### # ##### ## #####, ##### ##, #.#.
## ##### ##### # #####, ##### ## #####.
##### ##### ##### ##### #####:
```

---

```
uses SysUtils, Classes;
```

```
type
```

```
  TGun = class(TComponent)
```

```
##### ##### # ##### Object Pascal ###
#####
```

---

```
end;
```

```
TPlayer = class(TComponent)
  Gun1, Gun2: TGun;
  constructor Create(AOwner: TComponent); override;
end;
```

```
constructor TPlayer.Create(AOwner: TComponent);
begin
  inherited;
  Gun1 := TGun.Create(Self);
  Gun2 := TGun.Create(Self);
end;
```

```
##### #####, ### ##### override #####
constructor ## TComponent. ##, # #####, #####, ## #####
##### ##### constructor-. #####, ##-### ## ##### —#####
##### constructor # ##### reintroduce. ##### #####
##### #####, ## ## #####, #####, streaming,
## ##### ##### ##### constructor, ## #####
#####, ## ## ##### ##### ## #####
#####.
```

```
##### #####, ### ##### nil # ##### owner-.
##### ##### ## ##### ## ##### owner-##### # ##### ## #####
##### #####. ## #####, #####
##### ## ##### TComponent, ##### ## #####
##### ## #####. ##### #####
##### #####: ManualGun := TGun.Create(nil);.
```

```
## ## ##### #####
##### —##### OwnsObjects (##### ## ##### true)
#####-#####, #####, ## TFPGObjectList ## TObjectList. ##. #####
##### #####:
```

```
uses SysUtils, Classes, FGL;
```

```
type
```

```
TGun = class
end;
```

```
TGunList = specialize TFPGObjectList<TGun>;
```

---

```
TPlayer = class
  Guns: TGunList;
  Gun1, Gun2: TGun;
  constructor Create;
  destructor Destroy; override;
end;
```

```
constructor TPlayer.Create;
begin
  inherited;
  // Вообще говоря, параметр OwnsObjects и так true по умолчанию
  Guns := TGunList.Create(true);
  Gun1 := TGun.Create;
  Guns.Add(Gun1);
  Gun2 := TGun.Create;
  Guns.Add(Gun2);
end;
```

```
destructor TPlayer.Destroy;
begin
  { Здесь достаточно освободить сам список.
  Он сам автоматически освободит всё содержимое. }
  FreeAndNil(Guns);

  { Таким образом нет нужды освобождать Gun1, Gun2 отдельно. Правда,
  хорошей
  практикой будет теперь установить значение "nil" соответствующим
  значениям
  ссылок на них, поскольку мы знаем, что они освобождены.
  В этом простом классе с простым destructor-ом, очевидно,
  что к ним не произойдёт доступа, однако в случае сложных destructor-ов
  это может оказаться полезно.

  Альтернативно, можно избежать объявления Gun1 и Gun2 отдельно
  и использовать напрямую Guns[0] и Guns[1] в коде.
  Можно также создать метод Gun1, который возвращает ссылку на
  Guns[0]. }
  Gun1 := nil;
  Gun2 := nil;
  inherited;
end;
```

---

```
#####, ### ##### owner-## #####-##### ##### (### #####)
#####) #, # ##### ##### # #####
```





```
if Obj1 <> nil then  
  WriteLn(Obj1.ClassName);
```

---

```
##### nil ## #####  
#####  
##### Obj1 <> nil, # ##### # Obj1,  
#####
```

```
## ## ##### # ## #####  
##-##### # ##  
#####
```

2. ##### # **Obj2** —#### #####. ## ##### # **nil**, #####  
#####. ##### # ##-**nil** #####  
##### —#### #####  
##### access violation, # ##### -## #####

# #####:

- ##### - ##### #  
#####  
##### **TCar** #### **wheel**, #####  
##### **Twheel**, ## ##### # **wheel** #####,  
##### **car**, # ## **car** ##### # **wheel** #####  
##### destructor. ## ## #####. # #####  
##### # ##### # ##, ## #  
## #####

- # ##### # ##### **Obj1**, #####  
##### **Obj2** ##### **nil**. # #####  
#####

- ##### # ##### "free notification" (#####  
#####). ##### # #####  
# ##### # **nil**.

```
##### # #####  
##### # #####
```

```
##### ##### # ##### Object Pascal ###
#####
```

---

```
### ##### ### ##### ##### ##### TComponent.
##### ### ##### # ##### FreeNotification ,
RemoveFreeNotification, # override Notification.
```

```
##### ##### ##### ##### ##### ##### #
constructor-## / destructor-## # setter-##. ##### ##### #
#####, ## ##### ##### #####, ##### #
##### # ##### #.
```

---

#### type

```
TControl = class(TComponent)
end;
```

```
TContainer = class(TComponent)
```

```
private
```

```
FSomeSpecialControl: TControl;
```

```
procedure SetSomeSpecialControl(const Value: TControl);
```

```
protected
```

```
procedure Notification(AComponent: TComponent; Operation:
TOperation); override;
```

```
public
```

```
destructor Destroy; override;
```

```
property SomeSpecialControl: TControl
```

```
read FSomeSpecialControl write SetSomeSpecialControl;
```

```
end;
```

#### implementation

```
procedure TContainer.Notification(AComponent: TComponent; Operation:
TOperation);
```

```
begin
```

```
inherited;
```

```
if (Operation = opRemove) and (AComponent = FSomeSpecialControl) then
```

```
{ установить значение nil для SetSomeSpecialControl чтобы всё
аккуратно подчистить }
```

```
SomeSpecialControl := nil;
```

```
end;
```

```
procedure TContainer.SetSomeSpecialControl(const Value: TControl);
```

```
begin
```

```
if FSomeSpecialControl <> Value then
```

```
begin
```

```
if FSomeSpecialControl <> nil then
```

```
##### # ##### # ##### Object Pascal ###  
#####
```

---

```
    FSomeSpecialControl.RemoveFreeNotification(Self);  
    FSomeSpecialControl := Value;  
    if FSomeSpecialControl <> nil then  
        FSomeSpecialControl.FreeNotification(Self);  
    end;  
end;  
  
destructor TContainer.Destroy;  
begin  
    { Установить значение nil для SetSomeSpecialControl, чтобы запустить  
    notification про освобождение памяти }  
    SomeSpecialControl := nil;  
    inherited;  
end;
```

---

## 6. Run-time library

### 6.1. #####/##### # ##### # #####

```
### ##### / ##### # ##### # ##### # ##### # ##### # #####  
TStream. # ##### # ##### # ##### # ##### # ##### # ##### # #####  
### TFileStream, TMemoryStream, TStringStream # #.#.
```

---

```
{ $mode objfpc } { $H+ } { $J- }  
uses  
    SysUtils, Classes;  
  
var  
    S: TStream;  
    InputInt, OutputInt: Integer;  
begin  
    InputInt := 666;  
  
    S := TFileStream.Create('my_binary_file.data', fmCreate);  
    try  
        S.WriteBuffer(InputInt, SizeOf(InputInt));  
    finally  
        FreeAndNil(S);  
    end;  
  
    S := TFileStream.Create('my_binary_file.data', fmOpenRead);  
    try  
        S.ReadBuffer(OutputInt, SizeOf(OutputInt));  
    end;
```

```
##### # ##### # ##### Object Pascal ###
#####
```

---

```
finally
```

```
  FreeAndNil(S);
```

```
end;
```

```
  WriteLn('Из файла прочитано целое число: ', OutputInt);
```

```
end.
```

---

```
# Castle Game Engine ##### Download ### #####
#####, ##### # ##### URL. ### #####
##### # HTTP # HTTPS, Android assets # #####
#####. ##### # #####-##### #
##### (##### # data) #####
##### ApplicationData ### ##### URL castle-data:/xxx URL.
#####:
```

---

```
EnableNetwork := true;
```

```
S := Download('https://castle-engine.io/latest.zip');
```

---

```
S := Download('file:///home/michalis/my_binary_file.data');
```

---

```
S := Download(ApplicationData('gui/my_image.png'));
```

---

```
##### # ##### # ##### # ##### # ##### # #####
TStreamReader ## CastleClassUtils. ## ##### API,
### ##### ## TStream. URL ##### # ##### # #####
TStreamReader, ### ##### ## TStream #####.
```

---

```
Text := TStreamReader.Create(ApplicationData('my_data.txt'));
```

```
while not Text.Eof do
```

```
  WriteLnLog('NextLine', Text.ReadLine);
```

---

## 6.2. #####

```
### ##### # ##### # ##### # ##### # #####
##### generic ##### ## unit-# FGL. ##### TFPGList
### ##### # ##### (##### record-# ## ##### object-#), #
TFPGObjectList ### ##### # Castle Game Engine:
##### # ##### TGenericStructList ## CastleGenericLists
```



```
begin  
  if Source is TMyClass then  
  begin  
    SourceMyClass := TMyClass(Source);  
    MyInt := SourceMyClass.MyInt;  
    // Xxx := SourceMyClass.Xxx; // добавить необходимые поля здесь  
  end else  
    { Вызываем inherited ТОЛЬКО если не получается вручную обработать  
Source }  
    inherited Assign(Source);  
end;  
  
procedure TMyClassDescendant.Assign(Source: TPersistent);  
var  
  SourceMyClassDescendant: TMyClassDescendant;  
begin  
  if Source is TMyClassDescendant then  
  begin  
    SourceMyClassDescendant := TMyClassDescendant(Source);  
    MyString := SourceMyClassDescendant.MyString;  
    // Xxx := SourceMyClassDescendant.Xxx; // добавить необходимые поля  
здесь  
  end;  
  
  { ВСЕГДА вызываем inherited, чтобы TMyClass.Assign сама обработала  
все оставшиеся поля. }  
  inherited Assign(Source);  
end;  
  
var  
  C1, C2: TMyClass;  
  CD1, CD2: TMyClassDescendant;  
begin  
  // тестируем TMyClass.Assign  
  C1 := TMyClass.Create;  
  C2 := TMyClass.Create;  
  try  
    C1.MyInt := 666;  
    C2.Assign(C1);  
    WriteLn('C2 состояние: ', C2.MyInt);  
  finally  
    FreeAndNil(C1);  
    FreeAndNil(C2);  
end;
```

##### Object Pascal ###  
#####

```
// тестируем TMyClassDescendant.Assign
CD1 := TMyClassDescendant.Create;
CD2 := TMyClassDescendant.Create;
try
  CD1.MyInt := 44;
  CD1.MyString := 'что-нибудь';
  CD2.Assign(CD1);
  WriteLn('CD2 состояние: ', CD2.MyInt, ' ', CD2.MyString);
finally
  FreeAndNil(CD1);
  FreeAndNil(CD2);
end;
end.
```

##### override AssignTo #  
##### override Assign # #####,  
#####

##### inherited Assign.Inherited #  
TPersistent.Assign #####,  
##### AssignTo # #####,  
#####). # #####,  
##### Assign, ##  
##### inherited ## TMyClass.Assign.  
##.



##### TPersistent #  
##### published, #  
##### (streaming) #  
## TPersistent #. # # # # # # # # # # # # # #  
##### published. # # # # # # # # # # # # # #  
#####  
#####  
## public. #. # # # # # # # # # # # # # # 4.5, «#####  
#####».

## 7. #####

### 7.1. #####

##### (### #####, #####, ##### # #.)  
##### (#####) ###-#####.

##### (##### #  
#####) #####, # #####, #####  
#####. ### #####, #####  
##### # #####  
##### # #####. #####, ##  
#####. # ##### (#####)  
##### #, ### #####, #####  
##### # #####.

#####:

---

```
function SumOfSquares(const N: Integer): Integer;
```

```
    function Square(const Value: Integer): Integer;  
    begin  
        Result := Value * Value;  
    end;
```

```
var  
    I: Integer;  
begin  
    Result := 0;  
    for I := 0 to N do  
        Result := Result + Square(I);  
    end;
```

---

```
# ##### Square ##### I:
```

---

```
function SumOfSquares(const N: Integer): Integer;
```

```
var  
    I: Integer;  
  
function Square: Integer;  
begin  
    Result := I * I;
```





```
##### ##### # ##### Object Pascal ###
#####
```

---

```
var
  I: Integer;
begin
  Result := 1;
  for I := 2 to 10 do
    Result := F(Result, I);
end;
```

```
var
  SomeFunction: TMyFunction;
begin
  SomeFunction := @Add;
  WriteLn('1 + 2 + 3 ... + 10 = ', ProcessTheList(SomeFunction));

  SomeFunction := @Multiply;
  WriteLn('1 * 2 * 3 ... * 10 = ', ProcessTheList(SomeFunction));
end.
```

- 
- ##### ## #####: ### ##### # ##### ##### of object .
- 

```
{mode objfpc}{$H+}{$J-}
```

```
uses
  SysUtils;
```

```
type
  TMyMethod = procedure (const A: Integer) of object;
```

```
TMyClass = class
  CurrentValue: Integer;
  procedure Add(const A: Integer);
  procedure Multiply(const A: Integer);
  procedure ProcessTheList(const M: TMyMethod);
end;
```

```
procedure TMyClass.Add(const A: Integer);
begin
  CurrentValue := CurrentValue + A;
end;
```

```
procedure TMyClass.Multiply(const A: Integer);
begin
  CurrentValue := CurrentValue * A;
end;
```

```
procedure TMyClass.ProcessTheList(const M: TMyMethod);
```

```
##### ##### # ##### Object Pascal ###  
#####
```

---

```
var  
  I: Integer;  
begin  
  CurrentValue := 1;  
  for I := 2 to 10 do  
    M(I);  
end;  
  
var  
  C: TMyClass;  
begin  
  C := TMyClass.Create;  
  try  
    C.ProcessTheList(@C.Add);  
    WriteLn('1 + 2 + 3 ... + 10 = ', C.CurrentValue);  
  
    C.ProcessTheList(@C.Multiply);  
    WriteLn('1 * 2 * 3 ... * 10 = ', C.CurrentValue);  
  finally  
    FreeAndNil(C);  
  end;  
end.
```

---

```
##### #####, ### ##### ##### ##### #####  
### #####. ### #####. #### ### ##### #####  
of object, ## ## ##### #####-##### ### #####, ###  
##### ##### 8.3, «Class method» # ##### #####.
```

---

```
type  
  TMyMethod = function (const A, B: Integer): Integer of object;  
  
  TMyClass = class  
    class function Add(const A, B: Integer): Integer  
    class function Multiply(const A, B: Integer): Integer  
  end;  
  
var  
  M: TMyMethod;  
begin  
  M := @TMyClass(nil).Add;  
  M := @TMyClass(nil).Multiply;  
end;
```

---

```
##### # ##### # ##### Object Pascal ###
#####
```

```
# #####, # ##### ##### ##### ##### #####
@TMyClass(nil).Add, # ## ##### @TMyClass.Add.
• ##### ##### ## ##### #####: ### ##### ## ##### #####
# is nested # #####, # ##### ##### ##### { $modeswitch
nestedprocvars} ### ##### #####. ##### ##. ##### 7.1,
«##### (#####) #####».
```

### 7.3. Generic-#

Generic-# - ### ##### ##### ##### #####. ##### #####-  
##### (##### - #####) ##### ##### #####. #####  
##### ##### ##### ##### ##### (#####,  
#####, #####, #####...): ##### ##### T, # ##### specialize  
(#####) ### ##### ##### ##### ## integer, ##### ##  
string, ##### ## TMyRecord # #.#.

Generic-# # ##### ##### generic-## # C++. ##  
#####, ## ## "#####" ##### ##### specialize, ##  
##### ## ##### (#####, ##### ## "#####",  
##### ## #####, # #####  
##### "#####" #####-#### ##### # generic ##### ##  
"#####"). ##### ## ##### #####  
(##### ## #####) # #####  
#####. ##### ##### (integer, float) # ##### record,  
class # #.#. ## specialize ##### generic-#.

```
.....
{ $mode objfpc } { $H+ } { $J- }
uses
  SysUtils;

type
  generic TMyCalculator<T> = class
    Value: T;
    procedure Add(const A: T);
  end;

procedure TMyCalculator.Add(const A: T);
begin
  Value := Value + A;
end;
```

```
##### ##### # ##### Object Pascal ###  
#####
```

---

**type**

```
TMyFloatCalculator = specialize TMyCalculator<Single>;  
TMyStringCalculator = specialize TMyCalculator<string>;
```

**var**

```
FloatCalc: TMyFloatCalculator;  
StringCalc: TMyStringCalculator;
```

**begin**

```
FloatCalc := TMyFloatCalculator.Create;
```

**try**

```
FloatCalc.Add(3.14);  
FloatCalc.Add(1);  
WriteLn('Сложение величин типа Float: ', FloatCalc.Value:1:2);
```

**finally**

```
FreeAndNil(FloatCalc);
```

**end;**

```
StringCalc := TMyStringCalculator.Create;
```

**try**

```
StringCalc.Add('что-нибудь');  
StringCalc.Add(' ещё');  
WriteLn('Сложение величин типа String: ', StringCalc.Value);
```

**finally**

```
FreeAndNil(StringCalc);
```

**end;**

**end.**

---

```
##### generic-## ## #####, ##### generic  
##### # #####:
```

---

```
[$mode objfpc]{$H+}{$J-}
```

```
uses SysUtils;
```

```
{ Примечание: этот пример требует FPC 3.1.1 и не скомпилируется в FPC  
3.0.0 или более ранних версиях. }
```

```
generic function Min<T>(const A, B: T): T;
```

**begin**

```
if A < B then
```

```
Result := A
```

```
else
```

```
Result := B;
```

**end;**

**begin**

```
##### Object Pascal ###  
#####
```

```
WriteLn('Min (1, 0): ', specialize Min<Integer>(1, 0));  
WriteLn('Min (3.14, 5): ', specialize Min<Single>(3.14, 5):1:2);  
WriteLn('Min (''a'', ''b''): ', specialize Min<string>('a', 'b'));  
end.
```

## 7.4. Overload

```
##### (# #####, #####) # #####  
##### ## #####, ## ## #####. ## #####  
##### #####, ##### ##  
### ##### # #####.
```

```
## ##### overload ##### FPC. ## ##, ## ##  
## ## # ##### (# #####) ##, # #####  
## ##### # ##### # #####.#####  
#####, ## ## ## # ##### Foo(Integer) # Foo(string),  
# ## ## ## ##### # ##### ## ## Foo(Float), ##  
##### Foo(Float) #####, ##### # #####  
##### ## ## # #####. #####  
##### ## ## ## # #####.#####  
##### ## ## overload.
```

## 7.5. #####

```
##### ## ##:
```

- ##### ## ##### (#####, ## ## ##  
## ## ##, ## ## ##),
- ##### ## ## ##,
- ## ## ##.

```
##### ## ##, ##### # #####. # ##, #####  
##### ## ## ##, ##### ## #####  
#####. ##### ## ##, ## ##, ## ##  
##### ## ## "#####" ##### ## ##. ## ##, ## ##  
"#####".
```

```
{mode objfpc}{$H+}{$J-}  
unit PreprocessorStuff;  
interface
```

```
##### ##### # ##### Object Pascal ###  
#####
```

---

```
{$ifdef FPC}
```

```
{ всё что идёт внутри данного условия ifdef определено только для FPC, а  
не других компиляторов (например, Delphi). }
```

```
procedure Foo;
```

```
{$endif}
```

```
{ Определить константу NewLine. Это пример того, как "нормальный"  
синтаксис Паскаля "поломан" директивами предобработки.
```

```
Если компилировать на Unix-системах (включая Linux, Android, Mac OS X),  
компилятор увидит следующее:
```

```
const NewLine = #10;
```

```
Если компилировать на Windows, компилятор увидит так:
```

```
const NewLine = #13#10;
```

```
Однако, на других операционных системах, код не скомпилируется,  
поскольку компилятор увидит следующее:
```

```
const NewLine = ;
```

```
Вообще, это *хорошо* что в данном случае возникает ошибка -- если  
возникнет
```

```
необходимость портировать программу на другую операционную систему,  
которая не является ни Unix, ни Windows, то компилятор "напомнит", что  
необходимо выбрать правильное значение NewLine для такой системы. }
```

```
const
```

```
NewLine =
```

```
{$ifdef UNIX} #10 {$endif}
```

```
{$ifdef MSWINDOWS} #13#10 {$endif} ;
```

```
{$define MY_SYMBOL}
```

```
{$ifdef MY_SYMBOL}
```

```
procedure Bar;
```

```
{$endif}
```

```
{$define CallingConventionMacro := unknown}
```

```
{$ifdef UNIX}
```

```
{$define CallingConventionMacro := cdecl}
```

```
{$endif}
```

```
{$ifdef MSWINDOWS}
```

```
{$define CallingConventionMacro := stdcall}
```





```
##### # # ##### Object Pascal ###  
#####
```

```
##### "uses" ##### (##### # GUI-  
##### GUI #####). #####,  
##### # ##### myunit.pas ##### #  
### #####, ### # ##### #####,  
##### include #####  
#####.
```

- #####-#####  
unit # #####, ###, #####, #####  
#####:

```
.....  
{$ifdef UNIX} {$I my_unix_implementation.inc} {$endif}  
{$ifdef MSWINDOWS} {$I my_windows_implementation.inc} {$endif}  
.....
```

```
##### ## #####, ### #####  
{$ifdef UNIX}, {$ifdef MSWINDOWS} #####  
##### (#####). ##### #  
#####. ### #####  
##### -fi ### FPC, #####  
#####  
##### include ##### {$I my  
platform_specific_implementation.inc}. ### #####  
#####.
```

## 7.6. Record

```
Record #####. ###  
#####, #####: # record-## #####,  
##### structure # C-#####.
```

```
##### {modeswitch advancedrecords}, #######  
##### # ##### record # #####  
#####, #####, ##### ## ##### #  
##### record.
```

```
.....  
{$mode objfpc}{$H+}{$J-}  
{$modeswitch advancedrecords}
```

type

TMyRecord = record

public

```
##### # ##### # ##### Object Pascal ###  
#####
```

---

```
    I, Square: Integer;  
    procedure WriteLnDescription;  
end;  
  
procedure TMyRecord.WriteLnDescription;  
begin  
    WriteLn('Квадрат числа ', I, ' равен ', Square);  
end;  
  
var  
    A: array [0..9] of TMyRecord;  
    R: TMyRecord;  
    I: Integer;  
begin  
    for I := 0 to 9 do  
        begin  
            A[I].I := I;  
            A[I].Square := I * I;  
        end;  
  
        for R in A do  
            R.WriteLnDescription;  
        end.  
end.
```

---

```
# #####, # ##### ##### ##### class, #  
## record, ##### ##### ##### #####  
#####, ## ##### # #####. #####, record-# ## ##  
##### ##### ##### ##### ##### ## #####  
##### #####:
```

- Record-## ## ##### constructor ## destructor. ## ##### ##  
#####. ## ##### # ##### (## ##### "#####  
# #####"), ##### ##### ##### #####, ##### ##  
string, ##### ##### #####. ##### #####  
#### ##### ## ## #####, ##### #####  
##### ##### ##### ##### #####.
- #####, ##### ## record-## ##### # #####  
##### ## #####.
- ##### ##### (##### # ##### #####) ## record-##  
##### # #####: ##### C layout ##  
#### ##### packed record. ## ##### # #####  
#####:





```
##### # ##### # ##### Object Pascal ###
#####
```

---

```
end;
```

## 7.9. #####

```
# ##### "#####" (overload) #####
##### # #####
#####. #####:
```

---

```
{$mode objfpc}{$H+}{$J-}
```

```
uses StrUtils;
```

```
operator* (const S: string; const A: Integer): string;
```

```
begin
```

```
  Result := DupeString(S, A);
```

```
end;
```

```
begin
```

```
  WriteLn('повтор' * 10);
```

```
end.
```

---

```
##### #, ### #
#####-##### #, #####
##### #. #####.
```

---

```
{$mode objfpc}{$H+}{$J-}
```

```
uses
```

```
  SysUtils;
```

```
type
```

```
  TMyClass = class
```

```
    MyInt: Integer;
```

```
  end;
```

```
operator* (const C1, C2: TMyClass): TMyClass;
```

```
begin
```

```
  Result := TMyClass.Create;
```

```
  Result.MyInt := C1.MyInt * C2.MyInt;
```

```
end;
```

```
var
```

```
  C1, C2: TMyClass;
```

```
begin
```

```
  C1 := TMyClass.Create;
```

```
##### ##### # ##### Object Pascal ###
#####
```

---

```
try
  C1.MyInt := 12;
  C2 := C1 * C1;
  try
    WriteLn('12 * 12 = ', C2.MyInt);
  finally
    FreeAndNil(C2);
  end;
finally
  FreeAndNil(C1);
end;
end.
```

---

```
##### ##### # ##### record-###. ##### ## ##, ## # #####
#####, ##### ## ##### ##### ## #####
### #####.
```

---

```
{ $mode objfpc } { $H+ } { $J- }
uses
  SysUtils;

type
  TMyRecord = record
    MyInt: Integer;
  end;

operator* (const C1, C2: TMyRecord): TMyRecord;
begin
  Result.MyInt := C1.MyInt * C2.MyInt;
end;

var
  R1, R2: TMyRecord;
begin
  R1.MyInt := 12;
  R2 := R1 * R1;
  WriteLn('12 * 12 = ', R2.MyInt);
end.
```

---

```
### ##### # record-### ##### ##### { $modeswitch
advancedrecords } # ##### # class operator
##### ##### record-#. ##### ##### ##### generic
##### ##### ## #####-##### (#####
```

---

```
#####  #####  # ##### Object Pascal ###  
#####
```

```
TFPGList, #####  #####  ## #####  #####  #####). #  
#####  #####  "#####" #####  #####  (## #####  
#####  #####  record) ## #####  #####  (#####  ##  ##  #####  ##,  ##  
#####  #####  # TFPGList), # ## #####  specialize #####  # ##  
specialize TFPGList<TMyRecord>.
```

---

```
{ $mode objfpc } { $H+ } { $J- }  
{ $modeswitch advancedrecords }
```

```
uses
```

```
  SysUtils, FGL;
```

```
type
```

```
  TMyRecord = record
```

```
    MyInt: Integer;
```

```
    class operator+ (const C1, C2: TMyRecord): TMyRecord;
```

```
    class operator= (const C1, C2: TMyRecord): boolean;
```

```
  end;
```

```
class operator TMyRecord.+ (const C1, C2: TMyRecord): TMyRecord;
```

```
begin
```

```
  Result.MyInt := C1.MyInt + C2.MyInt;
```

```
end;
```

```
class operator TMyRecord.= (const C1, C2: TMyRecord): boolean;
```

```
begin
```

```
  Result := C1.MyInt = C2.MyInt;
```

```
end;
```

```
type
```

```
  TMyRecordList = specialize TFPGList<TMyRecord>;
```

```
var
```

```
  R, ListItem: TMyRecord;
```

```
  L: TMyRecordList;
```

```
begin
```

```
  L := TMyRecordList.Create;
```

```
  try
```

```
    R.MyInt := 1;   L.Add(R);
```

```
    R.MyInt := 10;  L.Add(R);
```

```
    R.MyInt := 100; L.Add(R);
```

```
  R.MyInt := 0;
```

```
  for ListItem in L do
```

```
    R := ListItem + R;
```

```
WriteLn('1 + 10 + 100 = ', R.MyInt);  
finally  
  FreeAndNil(L);  
end;  
end.
```

---

## 8. #####

### 8.1. ##### private # strict private

```
##### private #####  
#####. #####  
### # ##### unit-# ##### # #####  
# private ##### # C++ ##### #, ### #  
##### # ##### unit-# "#####". #####
```

```
#####, ### unit-# #####  
#####, #####  
strict private. #####, ###  
##### # #####. #####.
```

```
##### protected (#####  
##### "#####" # ## unit-#) # strict  
protected (##### # #####  
#####).
```

### 8.2. ##### # #####

```
##### (const) ##  
### (type). #####  
##### # #####  
##### private (## "#####"), ##
```

```
##### # var :
```

---

```
type  
  TMyClass = class
```



```
##### ##### # ##### Object Pascal ###  
#####
```

---

```
private  
  type  
    TInternalClass = class  
      Velocity: Single;  
      procedure DoSomething;  
    end;  
  var  
    FInternalClass: TInternalClass;  
public  
  const  
    DefaultVelocity = 100.0;  
  constructor Create;  
  destructor Destroy; override;  
end;  
  
constructor TMyClass.Create;  
begin  
  inherited;  
  FInternalClass := TInternalClass.Create;  
  FInternalClass.Velocity := DefaultVelocity;  
  FInternalClass.DoSomething;  
end;  
  
destructor TMyClass.Destroy;  
begin  
  FreeAndNil(FInternalClass);  
  inherited;  
end;  
  
{ Обратите внимание на префикс "TMyClass.TInternalClass." }  
procedure TMyClass.TInternalClass.DoSomething;  
begin  
end;
```

---

### 8.3. Class method

```
##### ##, ##### ##### ## ##### ( TMyClass ), ##  
##### ## ### #####.
```

---

```
type  
  TEnemy = class  
    procedure Kill;  
    class procedure KillAll;  
end;
```

```
var  
  E: TEnemy;  
begin  
  E := TEnemy.Create;  
  try  
    E.Kill;  
  finally  
    FreeAndNil(E);  
  end;  
  TEnemy.KillAll;  
end;
```

---

##### , ### ## ##### — ##### ##  
##### , ##### , ##### # ##### 8.4,  
«##### ».

##### , ### constructor ##### , ### class method #####  
##### "#####" (MyInstance := TMyClass.Create(...  
);). ##### , ##### # #####  
##### —# ## ##### # ##### #.  
##### "#####" constructor-## , #####  
##### constructor (##### , #####  
#####) ##### , # ##### constructor  
(##### , #####).

## 8.4. #####

##### , ### ## class method-# ## constructor-# ## ,  
##### . #####  
#####: class of TMyClass.

---

```
type  
  TMyClass = class(TComponent)  
  end;  
  
  TMyClass1 = class(TMyClass)  
  end;  
  
  TMyClass2 = class(TMyClass)  
  end;
```

```
TMyClassRef = class of TMyClass;
```

```
var
```

```
  C: TMyClass;
```

```
  ClassRef: TMyClassRef;
```

```
begin
```

```
  // Можно сделать так:
```

```
  C := TMyClass.Create(nil); FreeAndNil(C);
```

```
  C := TMyClass1.Create(nil); FreeAndNil(C);
```

```
  C := TMyClass2.Create(nil); FreeAndNil(C);
```

```
  // А с помощью ссылки на класс можно сделать следующим образом:
```

```
  ClassRef := TMyClass;
```

```
  C := ClassRef.Create(nil); FreeAndNil(C);
```

```
  ClassRef := TMyClass1;
```

```
  C := ClassRef.Create(nil); FreeAndNil(C);
```

```
  ClassRef := TMyClass2;
```

```
  C := ClassRef.Create(nil); FreeAndNil(C);
```

```
end;
```

---

```
##### ## ##### ##### ##### # ##### class method-###.  
##### ### ##### ##### # ##### — ##### ##  
##### ##### ##### ##### ## # ##### #####  
#####.
```

---

```
type
```

```
  TMyClass = class(TComponent)
```

```
    class procedure DoSomething; virtual; abstract;
```

```
  end;
```

```
  TMyClass1 = class(TMyClass)
```

```
    class procedure DoSomething; override;
```

```
  end;
```

```
  TMyClass2 = class(TMyClass)
```

```
    class procedure DoSomething; override;
```

```
  end;
```

```
  TMyClassRef = class of TMyClass;
```

```
##### # ##### # ##### Object Pascal ###  
#####
```

---

**var**

```
C: TMyClass;  
ClassRef: TMyClassRef;
```

**begin**

```
ClassRef := TMyClass1;  
ClassRef.DoSomething;
```

```
ClassRef := TMyClass2;  
ClassRef.DoSomething;
```

```
{ А следующая строка приведёт к ошибке выполнения,  
  поскольку DoSomething является abstract в TMyClass. }
```

```
ClassRef := TMyClass;  
ClassRef.DoSomething;
```

**end;**

---

```
#### ##### ##### # ##### ##### ## ##### (##  
## #####-##### # ## #, ##### ##  
##### ## ##### ######), #####  
##### ClassType. #####, ##### ClassType #####  
TClass, ##### # ##### class of TObject. ## ##  
##### ## ## # #####, ##### # ##  
##### ######.
```

```
ClassType ##### ##### ###### ###### ######,  
##### ##### constructor-#. ##### # #####  
#####, ## Clone ##### ###### ###### ######  
##### # ##### ###### ######. #####  
##### # Assign (##. ##### 6.3, «##### TPersistent.Assign)  
## ##, ##### ###### ######, ##### # #####  
#####.
```

```
##### ###### ######, ## ###### ###### ######  
constructor ##### ###### ######. ######, ## ######  
# ###### ###### ###### TComponent, ##### # ##  
##### override ##### constructor-# TComponent.Create(AOwner:  
TComponent)).
```

---

**type**

```
TMyClass = class(TComponent)  
  procedure Assign(Source: TPersistent); override;
```

---

```
#####      ##### # ##### Object Pascal ###
#####
```

---

```
function Clone(AOwner: TComponent): TMyClass;
end;
```

```
TMyClassRef = class of TMyClass;
```

```
function TMyClass.Clone(AOwner: TComponent): TMyClass;
```

```
begin
```

```
  // Таким образом будет создан класс конкретного типа TMyClass:
```

```
  // Result := TMyClass.Create(AOwner);
```

```
  // А такой подход может создать класс как типа TMyClass, так и его
  наследников:
```

```
  Result := TMyClassRef(ClassType).Create(AOwner);
```

```
  Result.Assign(Self);
```

```
end;
```

---

## 8.5. Class helper

```
#####      ##### #####      #####      #####. #####      #####
## #####      # #####      #####      #####      MyInstance.MyMethod(...
) . # #####      #####      #####      #####      ### #####      #####
#####      #####      X, #####      #####      X.Action(...).
```

```
#####, #####      #####      #####      #####      #####      ###-####, ##
#####      #####      #####      ## #####      TMyClass, #####      ##      ##
#####      #####      ##      TMyClass. #####      ##      #####      #####      #####. #####,
##      #####      #####      ##, #####      #####      #####      #####      ##
#####      ##      #####      #####. #####      #####      #####      #####      ##
#####      ##### — #####      #####      Render # #####      TMy3DObject #####
#####      #####, #####, #####, #####      #####      TMy3DObject
#####      ##      #####      ##      #####      ##      #####      #####
"#####" #####      #####, #####      #      #####      ##      ##
#####      ##      #####      ##.
```

```
#####      #####      #####      #####      ## - #####      #####      #####      ##
#####      #####      #####      ##      TMy3DObject      ##      #####.
```

---

```
procedure Render(const Obj1: TMy3DObject; const Color: TColor);
```

```
var
```

```
  I: Integer;
```

```
begin
```

```
  for I := 0 to Obj1.ShapesCount - 1 do
```

```
    RenderMesh(Obj1.Shape[I].Mesh, Color);
```

---

```
##### # ##### # ##### Object Pascal ###
#####
```

```
end;
```

```
# ### #####. #####, ##### - ##
##### # #####. ##### ## ##### #
##### X.Action(...), # ### ## #####:
Render(X, ...). ##### X.Render(...), ### ##
#####, ##### Render ## unit-#, # ##### TMy3DObject .
```

```
### ##### # ##### class helper-#, ##### /
#####, ##### # ##### # ##### #
### # #####. ##### ## # "#####"
##### — ## "#####" ##### TMy3DObject .
```

```
type
```

```
TMy3DObjectHelper = class helper for TMy3DObject
  procedure Render(const Color: TColor);
end;
```

```
procedure TMy3DObjectHelper.Render(const Color: TColor);
```

```
var
```

```
  I: Integer;
```

```
begin
```

```
  { Обратите внимание, мы получаем доступ к ShapesCount, Shape без
  дополнительных указаний типа TMy3DObject.ShapesCount }
```

```
  for I := 0 to ShapesCount - 1 do
    RenderMesh(Shape[I].Mesh, Color);
```

```
end;
```



```
##### "type helper", #####
##### #
# ##### integer. #####
##### "record helper" #####... ##, ##
#####. ##### #: http://lists.freepascal.org/fpc-announce/2013-February/000587.html.
```

## 8.6. ##### constructor-#, destructor-#

```
### destructor-# ##### Destroy, # ## virtual
(##### ## #####)
# #####.
```



**uses**

SysUtils;

**type**

TGun = **class**

**end;**

TPlayer = **class**

Gun1, Gun2: TGun;

**constructor** Create;

**destructor** Destroy; **override;**

**end;**

**constructor** TPlayer.Create;

**begin**

**inherited;**

Gun1 := TGun.Create;

**raise** Exception.Create('Вызываем exception из constructor-a!');

Gun2 := TGun.Create;

**end;**

**destructor** TPlayer.Destroy;

**begin**

{ В данном случае в результате ошибки в constructor-е, у нас  
может оказаться Gun1 <> nil и Gun2 = nil. Смиритесь.

В таком случае, FreeAndNil справится с задачей без каких-либо  
дополнительных действий с нашей стороны, поскольку FreeAndNil

проверяет

является ли экземпляр класса nil перед вызовом соответствующего

**destructor-a. }**

FreeAndNil(Gun1);

FreeAndNil(Gun2);

**inherited;**

**end;**

**begin**

**try**

TPlayer.Create;

**except**

**on** E: Exception **do**

WriteLn('Ошибка ' + E.ClassName + ': ' + E.Message);

**end;**

**end.**

---



## 9. #####

### 9.1. ##### (CORBA) #####

#####, ## ## ## # #####, ##### API, ## ## ##### ## ## #####  
#####. ##### ##### ##### ##### ##### #####, #####  
##### ##### #####.

##### ##### ##### ##### # ##### #####, #####  
## #####, # ##### ##### ## ##### #####  
## ##### ##### ##### ##### #####  
##### ## ##### ##### #####, ##### #####  
##### ##### # #####, ## ##### #####  
#####.

##### CORBA # Object Pascal ##### # ##### ##### ##,  
## # ##### # Java (<https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>) ## # C# (<https://msdn.microsoft.com/en-us/library/ms173156.aspx>).

---

```
{ $mode objfpc } { $H+ } { $J- }  
{ $interfaces corba }
```

#### uses

```
SysUtils, Classes;
```

#### type

```
IMyInterface = interface  
[ '{79352612-668B-4E8C-910A-26975E103CAC}' ]  
  procedure Shoot;  
end;
```

```
TMyClass1 = class(IMyInterface)  
  procedure Shoot;  
end;
```

```
TMyClass2 = class(IMyInterface)  
  procedure Shoot;  
end;
```

```
TMyClass3 = class  
  procedure Shoot;  
end;
```

```
procedure TMyClass1.Shoot;
begin
  WriteLn('TMyClass1.Shoot');
end;

procedure TMyClass2.Shoot;
begin
  WriteLn('TMyClass2.Shoot');
end;

procedure TMyClass3.Shoot;
begin
  WriteLn('TMyClass3.Shoot');
end;

procedure UseThroughInterface(I: IMyInterface);
begin
  Write('Стреляем... ');
  I.Shoot;
end;

var
  C1: TMyClass1;
  C2: TMyClass2;
  C3: TMyClass3;
begin
  C1 := TMyClass1.Create;
  C2 := TMyClass2.Create;
  C3 := TMyClass3.Create;
  try
    if C1 is IMyInterface then
      UseThroughInterface(C1 as IMyInterface);
    if C2 is IMyInterface then
      UseThroughInterface(C2 as IMyInterface);
    if C3 is IMyInterface then
      UseThroughInterface(C3 as IMyInterface);
  finally
    FreeAndNil(C1);
    FreeAndNil(C2);
    FreeAndNil(C3);
  end;
end.
```

---



##### **#####** # ##### Object Pascal **###**  
#####

- 
- ##### **#####** # ##### COM (**#####**  
**#####** **#####**).

#### **##### # ## ##### ##### COM #####?**

##### **#####**, ##### COM ##### # #####  
##### **#####** #####. ## ##### **#####** — ##### reference-  
counting ##### **#####**. ##### **#####** ##### **#####** #  
**#####** (**#####** **####**, ##### **#####** ## "**#####**" **#####**),  
## #### **#####**, ##### **#####** #####. # ##### ## **#####** #####,  
**#####** # ##### # ##### **#####**.

- ##### **#####** ##### **#####** ##### **#####** (**##** #####  
**#####** **#####**) **#####** ##### **#####**.

- ##### **#####** ##### **#####** ##### # ##### **#####**  
reference-counting.

- ##### **#####**-##### **#####** ## ##### **#####** COM #####.  
## #### **##** - ##### ## **#####** #####. ##### ## # #####  
**#####** #####, ## #### **#####** #####, #####. # #### ## **#####** #####  
**#####**, ##### **#####** ##### **#####** **#####**: ##### reference-  
counting ##### COM, ##### **#####** # ##### **#####**  
**#####** **\_AddRef** # **\_ReleaseRef**, ## ##### **#####** #  
**#####**. ##### **#####** ##### # **#####**, ##### **#####** ## **#####**  
**#####** ##### ## **#####** ##### **#####** ##### **#####**  
**#####** ##### **#####** ## **#####** ##### **#####**.

##### ## #####, ## **#####**: ##### **#####** ##### # ##### CORBA  
# **#####** **#####** **{\$interfaces corba}** ##### # **#####** #####,  
**#####** **#####** # **#####**. ## #### **#####**, COM ##### **##**  
**#####** "**#####**" #####.

#####, ##### **#####** **#####**, ##### **#####** ##### **#####** ##### # # COM  
**#####**.

#### **##### ## ##### reference-counting ##### # ##### CORBA?**

**#####**. ##### **#####** ##### **#####** **\_AddRef** / **\_ReleaseRef**.  
## **#####** **#####** **#####** **IUnknown**. #####, #  
**#####** **#####**, ##### **#####** **#####** **#####**  
reference-counting # **#####**, ##### **#####** **#####** COM  
**#####**.





```
##### # # ##### Object Pascal ###  
#####
```

---

```
##### # # # # #. #. ##### "7.7 Reference counting" #  
##### FPC (http://freepascal.org/docs-html/ref/refse47.html).
```

```
##### ##### COM #####
```

- ##### # # # # # reference-counting,
- ##### # TInterfacedObject
- # ##### # # # # #, ##### # # # # #  
##### # # # # #, ##### # # # # # reference-counting ##### # # # # #.

```
### ##### # # # # #:
```

---

```
{ $mode objfpc } { $H+ } { $J- }  
{ $interfaces com }
```

**uses**

```
SysUtils, Classes;
```

**type**

```
IMyInterface = interface
```

```
[ '{3075FFCD-8EFB-4E98-B157-261448B8D92E}' ]
```

```
    procedure Shoot;
```

```
end;
```

```
TMyClass1 = class(TInterfacedObject, IMyInterface)
```

```
    procedure Shoot;
```

```
end;
```

```
TMyClass2 = class(TInterfacedObject, IMyInterface)
```

```
    procedure Shoot;
```

```
end;
```

```
TMyClass3 = class(TInterfacedObject)
```

```
    procedure Shoot;
```

```
end;
```

```
procedure TMyClass1.Shoot;
```

```
begin
```

```
    WriteLn('TMyClass1.Shoot');
```

```
end;
```

```
procedure TMyClass2.Shoot;
```

```
begin
```

```
##### ##### # ##### Object Pascal ###  
#####
```

---

```
    WriteLn('TMyClass2.Shoot');  
end;  
  
procedure TMyClass3.Shoot;  
begin  
    WriteLn('TMyClass3.Shoot');  
end;  
  
procedure UseThroughInterface(I: IMyInterface);  
begin  
    Write('Стреляем... ');  
    I.Shoot;  
end;  
  
var  
    C1: IMyInterface; // COM управляет освобождением памяти  
    C2: IMyInterface; // COM управляет освобождением памяти  
    C3: TMyClass3;    // Здесь управлять освобождением памяти придётся ВАМ  
begin  
    C1 := TMyClass1.Create as IMyInterface;  
    C2 := TMyClass2.Create as IMyInterface;  
    C3 := TMyClass3.Create;  
    try  
        UseThroughInterface(C1); // Нет необходимости в операторе "as"  
        UseThroughInterface(C2);  
        if C3 is IMyInterface then  
            UseThroughInterface(C3 as IMyInterface); // Так не сработает  
    finally  
        { Переменные C1 и C2 выходят из поля зрения  
          и будут автоматически уничтожены сейчас.  
  
          а переменная C3 является экземпляром класса  
          и не управляется интерфейсом,  
          по этому её необходимо совободить вручную. }  
        FreeAndNil(C3);  
    end;  
end.
```

---

## 9.5. ##### COM ##### # ##### reference-counting

```
###   ###   ####   #####   #   #####   #####,   #   #####,  
#####   TComponent   (####   #####   #####,   #####   ###  
TNonRefCountedInterfacedObject   #
```

---





```
    WriteLn('TMyClass1.Shoot');
end;

procedure TMyClass2.Shoot;
begin
    WriteLn('TMyClass2.Shoot');
end;

procedure TMyClass3.Shoot;
begin
    WriteLn('TMyClass3.Shoot');
end;

procedure UseThroughInterface(I: IMyInterface);
begin
    write('Стреляем... ');
    I.Shoot;
end;

var
    C1: TMyClass1;
    C2: TMyClass2;
    C3: TMyClass3;

procedure UseInterfaces;
begin
    if C1 is IMyInterface then
        //if Supports(C1, IMyInterface) then // эта строчка идентична проверке
        "is" выше
        UseThroughInterface(C1 as IMyInterface);
    if C2 is IMyInterface then
        UseThroughInterface(C2 as IMyInterface);
    if C3 is IMyInterface then
        UseThroughInterface(C3 as IMyInterface);
end;

begin
    C1 := TMyClass1.Create(nil);
    C2 := TMyClass2.Create(nil);
    C3 := TMyClass3.Create(nil);
    try
        UseInterfaces;
    finally
        FreeAndNil(C1);
        FreeAndNil(C2);

```

```
#####  
##### # ##### Object Pascal ###  
#####
```

---

```
FreeAndNil(C3);  
end;  
end.
```

---

## 9.6. #####

```
#### ##### CORBA, ### # COM #####. ####, ####  
##### CORBA #####.
```

1. ##### as ##### #  
#####.

---

```
UseThroughInterface(Cx as IMyInterface);
```

---

```
#### ##### C1, C2, C3 ##  
##### # #####.  
##### C3, # ##### IMyInterface.
```

```
##### as ##### ## ##,  
##### # Cx ## (#, TMyClass2) ## ##  
##### (#, IMyInterface2).
```

However, it is not allowed for CORBA interfaces.

2. #####, ##### ##  
#####:

---

```
UseThroughInterface(Cx);
```

---

```
# ##### # ##### # #####.  
### ##### C1 # C2 (#, ##,  
##### IMyInterface), ## ## C3.
```

```
## #####, ##### ## ## ## ##  
#####. # ##### TMyClass, #####  
##### TMyClass, #####  
##### TMyClass ##### # ##  
#####. ## ##### # #####  
#####.
```

3. ##### # ##### # ##### # ##### # ##### IMyInterface(Cx) #####  
#####:

UseThroughInterface(IMyInterface(Cx));

#####, ##### # ##### # ##### # ##### # ##### # #####, ##  
#####. ##### # ##### # ##### # ##### # ##### # #####,  
##### # ##### # #####, ## # ##### # ##### # ##### # ##### # #####,  
## # ##### # ##### # ##### # ##### # ##### # #####.

##### # ##### # ##### # #####: ##### Cx # ##### # class (#####  
TMyClass2), ## # ##### # ##### # ##### # ##### # ##### # ##### # #####  
#####. ##### # #####, # ##### # ##### # ##### # ##### # #####  
##### - ##### # ##### (##### # ##### # #####).

##### # # # # #####, ##### # # ##### # #####:

{ \$mode objfpc } { \$H+ } { \$J- }

// { \$interfaces corba } // обратите внимание, что приведение типа с помощью  
"as" для интерфейсов типа CORBA не компилируется

uses Classes;

type

IMyInterface = **interface**  
[ '{7FC754BC-9CA7-4399-B947-D37DD30BA90A}' ]  
    **procedure** One;  
**end;**

IMyInterface2 = **interface**(IMyInterface)  
[ '{A72B7008-3F90-45C1-8F4C-E77C4302AA3E}' ]  
    **procedure** Two;  
**end;**

IMyInterface3 = **interface**(IMyInterface2)  
[ '{924BFB98-B049-4945-AF17-1DB08DB1C0C5}' ]  
    **procedure** Three;  
**end;**

TMyClass = **class**(TComponent, IMyInterface)  
    **procedure** One;  
**end;**

```
TMyClass2 = class(TMyClass, IMyInterface, IMyInterface2)
  procedure One;
  procedure Two;
end;

procedure TMyClass.One;
begin
  Writeln('TMyClass.One');
end;

procedure TMyClass2.One;
begin
  Writeln('TMyClass2.One');
end;

procedure TMyClass2.Two;
begin
  Writeln('TMyClass2.Two');
end;

procedure UseInterface2(const I: IMyInterface2);
begin
  I.One;
  I.Two;
end;

procedure UseInterface3(const I: IMyInterface3);
begin
  I.One;
  I.Two;
  I.Three;
end;

var
  My: IMyInterface;
  MyClass: TMyClass;
begin
  My := TMyClass2.Create(nil);
  MyClass := TMyClass2.Create(nil);

  // Следующий код не скомпилируется, так как в момент компиляции
  // неизвестно является ли My интерфейсом IMyInterface2.
  // UseInterface2(My);
  // UseInterface2(MyClass);
```

```
##### ##### # ##### Object Pascal ###
#####
```

---

```
// Это скомпилируется и работает правильно.
UseInterface2(ИMyInterface2(My));
// А это не скомпилируется. Приведение типа InterfaceType(ClassType)
проверяется в момент компиляции.
// UseInterface2(ИMyInterface2(MyClass));

// Это скомпилируется и работает правильно.
UseInterface2(My as ИMyInterface2);
// Это скомпилируется и работает правильно.
UseInterface2(MyClass as ИMyInterface2);

// Это скомпилируется, но приведёт к непонятной ошибке "Access
violation" при выполнении программы.
// UseInterface3(ИMyInterface3(My));
// Это не скомпилируется. Приведение типа InterfaceType(ClassType)
проверяется в момент компиляции.
// UseInterface3(ИMyInterface3(MyClass));

// Это скомпилируется, но приведёт к понятному сообщению об ошибке
"EInvalidCast: Invalid type cast" и укажет на проблему.
// UseInterface3(My as ИMyInterface3);
// Это скомпилируется, но приведёт к понятному сообщению об ошибке
"EInvalidCast: Invalid type cast" и укажет на проблему.
// UseInterface3(MyClass as ИMyInterface3);

writeln('Готово');
end.
```

---

## 10. ### #####

Copyright Michalis Kamburelis.

```
##### ##### ##### ##### # ##### AsciiDoc ##### ##
#####: https://github.com/michaliskambi/modern-pascal-introduction. #####
### #####, #####, ##### # pull
request-## :). # ##### ##### ##### GitHub ##### ## e-mail:
michalis@castle-engine.io1. #####: https://michalis.xyz/.
```

```
#### ##### ##### ##### # #####
##### ##### Wikipedia, ##. https://en.wikipedia.org/wiki/
Wikipedia:Copyrights:
```

---

<sup>1</sup> <mailto:michalis@castle-engine.io>

#####  
##### # ##### Object Pascal ###  
#####

---

- *Creative Commons Attribution-ShareAlike 3.0 Unported License (CC BY-SA)*

####

- *GNU Free Documentation License (GFDL) (unversioned, with no invariant sections, front-cover texts, or back-cover texts) .*

##### ## #####!

##### ## ##### #: ##### # ##### 2016-2019

##### ## ##### # ##### # ##### #: vityal #  
pupsik @ freepascal.ru