
-

#####

Michalis Kamburelis

#####

1. #####	3
2. #####	4
2.1. ##### "Hello world"	4
2.2. #####, #####, #####	4
2.3. ##### (if)	7
2.4. #####, ##### # #####	8
2.5. ##### ## ##### ## ##### (case)	10
2.6. ##### # #####, ##### # ##### # #####	10
2.7. ##### (for, while, repeat, for .. in)	12
2.8. ##### ## #####, #####	15
2.9. ##### # ##	16
3. ##### (Unit-#)	18
3.1. Unit-#, ##### ## #####	19
3.2. ##### ## ##### # ##### unit-#	20
3.3. ##### ## ##### ## unit #####	23
4. #####	24
4.1. #####	24
4.2. #####, ##### (is), ##### ## ##### (as)	25
4.3. #####	28
4.4. ##### - #####	31
4.5. ##### ## #####	32
4.6. ##### ## #####	33
4.7. Self	33
4.8. ##### ## #####	33
4.9. #####, ##### # #####	37
5. ##### ## #####	41
5.1. ##### ## ##### ## #####	41
5.2. ## ## #####	41
5.3. ##### # #####	42

5.4. ##### Destroy	45
5.5. #####	46
5.6. ##### (Castle Game Engine) ...	49
6. #####	51
6.1. #####	51
6.2. #####	52
6.3. #####	53
6.4. Finally (#####)	56
6.5. #####	59
7. Run-time #####	59
7.1. #####/##### # #####	59
7.2. ##### (#####), #####, #####	61
7.3. #####: TPersistent.Assign	67
8. #####	72
8.1. ##### (#####) #####	72
8.2. Callbacks (#####, #####, #####, #####, #####, #####, #####)	73
8.3. #####	76
8.4. Overloading	78
8.5. #####	78
8.6. #####	81
8.7. #####, #####	83
8.8. #####	84
8.9. #####	85
9. #####	88
9.1. ##### # #####	88
9.2. ##### # #####	89
9.3. #####	90
9.4. #####	91
9.5. #####	93
9.6. ##### # #####	95
9.7. #####	96
9.8. #####, #####	98
9.9. ##### # #####	98
10. #####	100
10.1. ##### (CORBA) #####	100
10.2. ##### CORBA # COM	102
10.3. ##### GUIDs	104

10.4. ##### # ##### ## ##### (COM)	105
10.5. ##### ## COM ##### # ##### #####	108
10.6. ##### ## #####	110
11. ##### #####	113

```
<style> body { font-family: "Open Sans", "DejaVu Sans", sans-serif; } </style>
```

1.

#####, ## # ##### ## ## ##### #### ##
#####, ##### # ## #####, #####¹ ## #####².

```
#####  
#####  
#####.  
#####  
#####  
#####  
#####  
#####  
#####  
#####
```

C++, Java ### C#.

- ### ## ##### #####, ##### ## ##### - #####, #####, #####³, ##### ...
- ### ## ##### ##### ## ##### ##,
- ### # #####,
- ### # ##### ## ##### ##, ## ##### ## # # ## ##### ## #####.

##, ##### Free Pascal Compiler, <http://freepascal.org/> . ## # ##### IDE (##### , Debugger, ##### ## ##### , ##### ## ####), ##### Lazarus <http://lazarus.freepascal.org/> . ##### ## ## ## Castle Game Engine, <https://castle-engine.io/> , ##### # 3D # 2D ##### # ##### ##, ##### ## ##### ## ## ## ## ## (Windows, Linux, macOS, Android, iOS, Nintendo Switch; ##### ## # WebGL).

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

- 1 ##### = Unit
- 2 ##### = Generics
- 3 ##### = Interface


```

procedure MyProcedure(const A: Integer);
begin
    WriteLn('A + 10 е: ', A + 10);
end;

function MyFunction(const S: string): string;
begin
    Result := S + 'низове се управляват автоматично';
end;

var
    X: Single;
begin
    WriteLn(MyFunction('Забележка: '));
    MyProcedure(5);

    // Делението с "/" винаги дава резултат float,
    // използвайте "div" за целочислено делене
    X := 15 / 5;
    WriteLn('X сега е: ', X); // научна нотация
    WriteLn('X сега е: ', X:1:2); // 2 десетични знака
end.

```

```

## ## ##### ## ## ##, ##### ## #####
##### Result. ##### ## ##### ##### Result, #####
##### # ##### ##### #####.

```

```

function MyFunction(const S: string): string;
begin
    Result := S + 'нещо';
    Result := Result + ' още нещо!';
    Result := Result + ' и още!';
end;

```

```

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

```

```

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

```

```

    Result := Count;
end;

begin
    Count := 10;
    CountMe; // функцията се изпълнява но резултата ѝ се игнорира, Count
сега е 11
    MyCount := CountMe; // резултата от функцията се използва, MyCount става
равно на Count, което сега е 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
        DoSomethingElse;

    // еквивалентно на горното
    B := A > 10;
    if B then
        DoSomething
    else
        DoSomethingElse;
end;

```

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

#####:

```
var
  A, B: Integer;
begin
  if A = 0 and B <> 0 then ... // НЕКОРЕКТЕН пример
```

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

#. ##### (short-circuit evaluation). #####:

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

- ##### #, ## ##### MyFunction(X).
- ### MyFunction(X) ##### false, ##### #
(##### # false and каквото_и_да_е # 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;
  3..10: DoSomethingInCaseItsInThisRange;
  11, 21, 31: AndDoSomethingForTheseSpecialValues;
  else DoSomethingInCaseOfUnexpectedValue;
end;
```

else # ##### (# ##### # default # C-#####
#####). ##### else ##
#####.

C-##### # ##### switch, ##
(fall-through) ## ##.
#####. ## # ##### # ##
break. ## ##### -#####
case, #####.

2.6. #####, ##### #

#####, ##### ##
##-##### ## enums # #####:)

type


```

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+} // включена проверка на диапазона - подходящо за дебъг
var
  MyArray: array [0..9] of Integer;
  I: Integer;
begin
  // инициализация
  for I := 0 to 9 do
    MyArray[I] := I * I;

  // показване
  for I := 0 to 9 do
    WriteLn('Квадрата е ', MyArray[I]);

  // прави същото като горното
  for I := Low(MyArray) to High(MyArray) do
    WriteLn('Квадрата е ', MyArray[I]);

  // прави същото като горното
  I := 0;
  while I < 10 do
  begin
    WriteLn('Квадрата е ', MyArray[I]);
    I := I + 1; // или "I += 1", или "Inc(I)"
  end;

  // прави същото като горното
  I := 0;

```

```
##### for I in .. do .. # ##### # foreach # ##### #####  
## #####. ## ## # ##### # ##### # #####:
```

- ##### # (#####).
- ##### #:

```
var  
  AK: TAnimalKind;  
begin  
  for AK in TAnimalKind do...
```

- #####:

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

- # ##### # #####, #####, #####
TObjectList or 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); // true = притежава елементите си  
  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('Квадрата на ', C.I, ' е ', C.Square);
finally
  FreeAndNil(List);
end;
end.
```

#####, #####

#####. #####

:)

2.8. #####,

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

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

```
WriteLn('Hello world!');
WriteLn('Може да отпечатате цяло число: ', 3 * 4);
WriteLn('Може да разширите полето на цяло число: ', 666:10);
WriteLn('Може да отпечатате число с плаваща запетая: ', Pi:1:4);
```

```
## ## ##### # ## # ##, ##### LineEnding
(## FPC RTL). (Castle Game Engine ##### #-#####
NL.) ##### # ##### # #####, ##
#####
```

```
WriteLn('One line.\nSecond line.');// НЕКОРЕКТЕН пример
```

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

```
WriteLn('Първи ред.' + LineEnding + 'Втори ред.');
```


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 е равно на: ', A + 10);
end;

function MyFunction(const S: string): string;
begin
    Result := S + 'низове се управляват автоматично';
end;

end.

```

myprogram.lpr (lpr
= Lazarus program file; # Delphi ##### .dpr). ##### ##

.pas ## ## ## ## .pp ## unit-# ## ##
.pas ## unit-# # .lpr ## FPC/Lazarus #####.

unit ### ## uses :

```

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

program MyProgram;

```

uses

MyUnit;

begin

WriteLn(MyFunction('Забележка: '));

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

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

unit AnotherUnit;

interface

uses Classes;

{ Типът (клас) "TComponent" е дефиниран в unit Classes.

Поради тази причина трябва да използваме uses Classes; по-горе. }

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

implementation

uses SysUtils;

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

begin

{ Процедурата FreeAndNil е дефинирана в unit SysUtils.

Тъй като го използваме само в реализацията а не в интерфейсната част,
достатъчно е да използваме uses SysUtils; в секция "implementation". }

FreeAndNil(C);

end;

end.

unit-# # #####
unit-# ## ## ##### # #####
interface. ##### ## ## #, ## ## "#####"
unit, ##### ## ## "#####"
unit-#, ##### ## ##. #####

Makefile ## #####

#####.

unit-# ## ##
implementation. ##### unit A
B # ## interface # ## unit B
unit A # ## implementation.

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

unit-# ## ## ##

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

unit-# ## ##
unit-# ## ## ## MyUnit.MyIdentifier. ##
#####


```
uses GoogleMapsEngine;

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

end.
```

```
# unit Graphics (## Lazarus LCL) ## ##### TColor. ## #####
## ## ##### unit, ##### ## ## ## ##
##### ShowColor, ##### ## ##### # interface.
##### # ## unit GoogleMapsEngine ##### ## ## TColor.
##### ## ##### ## ## implementation, #####
##### TColor ##### # implementation. ##### ## ##
unit, ##### ## ## ##, ## #####:
```

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

```
// НЕКОРЕКТЕН пример
// Ето какво "вижда" компилатора когато се опитва да компилира предишното
```

```
interface

uses Graphics;

procedure ShowColor(const Color: Graphics.TColor);

implementation

uses GoogleMapsEngine;

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

end.
```

```
##### ## ##### # ##### # ##### — ##### ## ## # implementaton
## ## ##### TColor ## unit Graphics. ##### # ## ## ##
##### GoogleMapsEngine # ##### interface ##### 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.

```
{ $mode objfpc } { $H+ } { $J- }  
unit MyUnit;  
  
interface  
  
uses Graphics;  
  
type
```


- ```
type
 TMyClass = class
 MyInt: Integer; // това е поле
 property MyIntProperty: Integer read MyInt write MyInt; // това е
 СВОЙСТВО
 procedure MyMethod; // това е метод
 end;

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

*##### ##### ##### ## #### # ##### #####.*

```
{ $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 shows MyInt + 10: ', MyInt + 10);
```

end;

**procedure** TMyClassDescendant.MyVirtualMethod;

**begin**

    WriteLn('TMyClassDescendant shows 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. ##  
 ##### 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('MyMethod');
end;

procedure TMyClassDescendant.MyMethodInDescendant;
begin
 WriteLn('MyMethodInDescendant');
end;

var
 Descendant: TMyClassDescendant;
 C: TMyClass;
begin
 Descendant := TMyClassDescendant.Create;
 try
 Descendant.MyMethod;
 Descendant.MyMethodInDescendant;

 { Descendant има цялата функционалност, която се очаква от
 TMyClass, така че това присвояване е OK }
 C := Descendant;
 C.MyMethod;

 { Това не може да сработи, тъй като TMyClass не дефинира този метод }
 //C.MyMethodInDescendant;
 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;
// долното е малко по-бързо
if A is TMyClass then
 TMyClass(A).CallSomeMethodOfMyClass;
.....
```

### 4.3. #####

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

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

type

```
TWebPage = class
private
 FURL: string;
 FColor: TColor;
 function SetColor(const Value: TColor);
public
 { Няма начин да се запише директно.
 Извикайте метода Load, например Load('http://www.freepascal.org/'),
 за да заредите страницата и да установите свойството. }
 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;
 // за пример: предизвиква обновяване всеки път при промяна на
 стойността
 Repaint;
 // пак за пример: осигурява, че някаква друга вътрешна инстанция,
 // като "RenderingComponent" (каквато и да е тя),
 // съдържа същата стойност за Color.
 RenderingComponent.Color := Value;
 end;
end;
```

#####, ## ##### ## ## ##### ##, #### ## ## ##### # ## ##  
 ##### (#####) ## ##### ##. # #####  
 #####, ##### Color ##### ## (setter SetColor. ## ##  
 ##### Color ##### ##  
 ##### FColor. ##### # ##-#####  
 "#####" ##### ##. ##-##### ##, #### ##  
 #####.

##### ## #####:

1. ##### ## ## ##### # ## (# ##### ## ##### ##  
 ##### getter);
2. # ##### — ##### ## ## ##### # ## (# ##### ## #####  
 ## ## ##### setter).

##### ##### ##### ## ##### ##### # #####  
 ##### ## #####. #####, ## ## ##### Integer  
 #####, ##### ## ##### ## ## Integer ## #####  
 #####, ##### Integer.

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

-  ##### ## ## ##### # ##### ##, ## ## unit. ##### ##### ## ##### — ##### ## #####, ## ##### # ##### ## ##### ##### ## getter # setter.



```

procedure TMyClass.MyMethod;
begin
 if Random > 0.5 then
 raise Exception.Create('Raising an exception!');
end;

var
 C: TMyClass;
begin
 Randomize;
 C := TMyClass.Create;
 try
 C.MyMethod;
 finally
 FreeAndNil(C);
 end;
end.

```

---

```

#####, ## finally ## #####
Exit (## / / ##) ## Break
Continue (## ##).

```

```

6, „#####“ ## #-##### ## #####.

```

## 4.5. #####

```

#####-#####, #
/ / .

```

```

#:

```

**public**

```

##, # ##### unit-#.

```

**private**

```

.

```

**protected**

```

.

```

```

#-##### private # protected ## # .
unit ## #,
private ## protected. #####,
#-#####. strict

```



private ### strict protected ## ## #####  
#####. ##-##### ## # ##### # ##### 9.1, „##### # #####“.

### ## #####, ## ##### public. #####  
## ## ## ##### # ##### {M+}, ## #####  
## ## ## # {M+}, ##### #####  
TPersistent, ##### # ##### TComponent  
(##### TComponent # ##### TPersistent). ## ##  
##### # published, ##### # public, ## #  
##### ## ##### ## ## ##.

## ##### # ##### ## ## # published (## ## ##  
## ## ##### # ##### ## ##### ## ##  
##### public, ## ## # ## ## ## ## ##  
#### # # ##### ## #####.

## 4.6. #####

### ## ##### ##, ## ## ##, ## ##  
##### TObject.

## 4.7. Self

##### Self (##) ## ## ##  
## ## ##, ## ## ## ##. ## #  
##### ## this ## C++, Java # #####.

## 4.8. #####

# ##### ## ##, ## ##, ##  
##### ## ## ##. #  
## ##-####, TMyClass2.MyOtherMethod ##### MyMethod, #####  
##### TMyClass2.MyMethod.

---

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

```
uses SysUtils;
```

```
type
```

```
 TMyClass1 = class
 procedure MyMethod;
 end;
```

```

TMyClass2 = class(TMyClass1)
 procedure MyMethod;
 procedure MyOtherMethod;
end;

procedure TMyClass1.MyMethod;
begin
 Writeln('TMyClass1.MyMethod');
end;

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

procedure TMyClass2.MyOtherMethod;
begin
 MyMethod; // this calls TMyClass2.MyMethod
end;

var
 C: TMyClass2;
begin
 C := TMyClass2.Create;
 try
 C.MyOtherMethod;
 finally FreeAndNil(C) end;
end.

```

---

```

#, ##### # #####
#. #####, ##### MyMethod # #####
TMyClass2, #####

```

- ##### TMyClass2.MyMethod.
- ### # # #, ##### TMyClass1.MyMethod.
- ### # # #, ##### TObject.MyMethod.
- ### # # #, ##### # # #.

```

#
TMyClass2.MyMethod # #-##### #. ##### # # # # #
TMyClass2.MyOtherMethod # # # ##### TMyClass1.MyMethod.

```

#####  
#####  
##### (#####  
#####).  
##### inherited  
##### MyMethod

inherited MyMethod;

#####  
#####  
MyMethod # TMyClass1.MyMethod, ##### TObject.MyMethod #  
##### TMyClass2.MyMethod.



##### TMyClass2.MyOtherMethod  
inherited MyMethod #  
#####.

#####  
#####  
#####  
#####.

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

```
type
 TMyClass1 = class
 constructor Create;
 procedure MyMethod(const A: Integer);
 end;

 TMyClass2 = class(TMyClass1)
 constructor Create;
 procedure MyMethod(const A: Integer);
 end;

constructor TMyClass1.Create;
begin
 inherited Create; // this calls TObject.Create
 Writeln('TMyClass1.Create');
end;

procedure TMyClass1.MyMethod(const A: Integer);
```



```
inherited;
```

---

36

```
begin
 writeln('TMyClass2.MyMethod начало ', A);
 A := 456;
 { Това извиква TMyClass1.MyMethod with A = 456,
 независимо от стойността на A подадена на този метод
 (TMyClass2.MyMethod). }
 inherited;
 writeln('TMyClass2.MyMethod крайно ', A);
end;
```

##### 2: ##### MyMethod (## "#####  
 ## "#####") ##### ##  
 ##### ## # ##### ##-####. ## inherited #####  
 ##### ## inherited #####, ##  
 ##### ##  
 ## ##, ## ##

#### 4.9. #####, #####

## #####. #### # C++ # ##  
 ##### ## Java.

##### ##, ##### ##  
 ##### ##, # ##  
 ##### ##. #####, ## #  
 ##### ##, ## # TFruit, ##  
 ## ##-##### TApple.

##### ##-##### #, ##  
 ##### # #####  
 ##### ##  
 ##### ##  
 #####:.

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

type
 TFruit = class
 procedure Eat;
 end;
```

```

TApple = class(TFruit)
 procedure Eat;
end;

procedure TFruit.Eat;
begin
 Writeln('Изядохме плод');
end;

procedure TApple.Eat;
begin
 Writeln('Изядохме ябълка');
end;

procedure DoSomethingWithAFruit(const Fruit: TFruit);
begin
 Writeln('Имаме плод от клас ', Fruit.ClassName);
 Writeln('Ядем го:');
 Fruit.Eat;
end;

var
 Apple: TApple; // Забележка: тук също така може да декларирате "Apple:
 TFruit"
begin
 Apple := TApple.Create;
 try
 DoSomethingWithAFruit(Apple);
 finally FreeAndNil(Apple) end;
end.

```

#### #####

---

```

Имаме плод от клас TApple
Ядем го:
Изядохме плод

```

---

```

Fruit.Eat ##### TFruit.Eat
TFruit.Eat .

```

```

#####, ##### ## ## ## #####:
Fruit.Eat, ##### Fruit ##
TFruit. ##### Eat # TFruit.###
TFruit ## #####, ##### # #####

```



```

 writeln('Изядохме ябълка');
end;

procedure DoSomethingWithAFruit(const Fruit: TFruit);
begin
 writeln('Имаме плод от клас ', Fruit.ClassName);
 writeln('Ядем го:');
 Fruit.Eat;
end;

var
 Apple: TApple; // Забележка: тук също така може да декларирате "Apple:
 TFruit"
begin
 Apple := TApple.Create;
 try
 DoSomethingWithAFruit(Apple);
 finally FreeAndNil(Apple) end;
end.

```

#### ##### ## #####

---

```

Имаме плод от клас TApple
Ядем го:
Изядохме ябълка

```

---

#####  
##### (VMT),  
#####  
Eat ,  
##### Fruit ,  
Eat #####

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



## 5. #####

### 5.1. #####

#####. # #####  
#####. #####  
-gl -gh ## FPC ## ##### (### [https://castle-engine.io/manual\\_optimization.php#section\\_memory](https://castle-engine.io/manual_optimization.php#section_memory) ).

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

### 5.2. #####

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

#####:

```
if A <> nil then
begin
 A.Destroy;
 A := nil;
end;
```

##### FreeAndNil #####  
nil ## A #####  
##### — #####, ## "#####"  
#####.

##### A.Free, #####:

```
if A <> nil then
 A.Destroy;
```

##### A (# ), #####  
 ## # nil.

#####  
 ##### nil. ##### A.Free #####  
 ##### Free #####  
 ##### "#####" # — #####  
 Self <> nil. ##### (#####  
 #####).

##### FreeAndNil(A) #####, #####  
 ##### Free ##### Destroy. Castle  
 Game Engine #####. #####  
 ##### nil, #####.

### 5.3. #####

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

```

---

```

uses SysUtils, Classes;

```

```

type
 TGun = class(TComponent)
 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;

```

---

```

#####
TComponent.#### ## #####.
(##### — ##### # reintroduce.## #####
#####, ## ## #####, ##### ## ##
#####, ## ##
#####.)

```

---

```
, ## ##### nil.
"#####" ####
#####. #### ## , ## #####
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;
```

```
 // Всъщност, стойността true (за OwnsObjects) е зададена по подразбиране
```

```
 Guns := TGunList.Create(true);
```

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

```
 Guns.Add(Gun1);
```

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

```
 Guns.Add(Gun2);
```

```
end;
```

```
destructor TPlayer.Destroy;
```

```
begin
```

```
 { Трябва да се погрижим за освобождаването на списъка.
```

```
 Той ще освободи елементите си автоматично. }
```

```
 FreeAndNil(Guns);
```

{ Вече няма нужда да освобождаваме ръчно Gun1, Gun2. Хубав навик е да установим на "nil"  
техните препратки, тъй като знаем, че са освободени. В този прост клас и с този прост деструктор, очевидно е, че те няма да бъдат достъпвани повече --  
но правейки така ще ни помогне в случая на по-големи и по-сложни деструктори.

Алтернативно, можем да си спестим декларирането на Gun1 и Gun2, и вместо това да използваме Guns[0] и Guns[1] в нашия код.

Или да създадем метод Gun1, който връща Guns[0]. }

```
Gun1 := nil;
Gun2 := nil;
inherited;
end;
```

#####  
#####  
#####  
##### Extract #####  
#####  
#####  
#####

```
Castle Game Engine: ##### TX3DNode
children
TX3DNode. ##### X3D #####, TX3DRootNode, ##
TCastleSceneCore.
-
OwnsXxx.
```

## 5.4. ##### Destroy

#####  
##### деструктор, ##### Destroy.

```


Destroy, ##### Free, #####
FreeAndNil.
```

```
Destroy # TObject # #####
override
```

#### (### ##### TObjekt). #### #  
##### ## Free. ##### ##  
##### ## 4.9, „#####“.



```

#####.

#####.
Create # #####
#####, ## # ##### # #####
#####.

Create # TObjekt ## #
#####, ##### ## ## ## override #
#####.

#####. ##### ## ## ## ##

#####, ## # ##### #
TComponent. TComponent
Create(AOwner: TComponent)

#####. ##### TComponent, ##### #
(## ## #
override) # ## ##

"#####". ##### ## ##

Create(AOwner: TComponent), # #####

Lazarus.
```

## 5.5. #####

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



```
#####, ##### ##### Obj1,
nil ## ##### Obj2 .
#####.
```

- ## #####  
## #####  
#####  
#####  
#####

```
#####, ##### ## #####,
#####/##### # ##### ## ##### ## setter.
##-#####, ## ##### # #####, #####
:)
```

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

---

48



```

 read FSomeSpecialControl write SetSomeSpecialControl;
end;

implementation

procedure TContainer.Notification(AComponent: TComponent; Operation:
TOperation);
begin
 inherited;
 if (Operation = opRemove) and (AComponent = FSomeSpecialControl) then
 { set to nil by SetSomeSpecialControl to clean nicely }
 SomeSpecialControl := nil;
end;

procedure TContainer.SetSomeSpecialControl(const Value: TControl);
begin
 if FSomeSpecialControl <> Value then
 begin
 if FSomeSpecialControl <> nil then
 FSomeSpecialControl.RemoveFreeNotification(Self);
 FSomeSpecialControl := Value;
 if FSomeSpecialControl <> nil then
 FSomeSpecialControl.FreeNotification(Self);
 end;
end;

destructor TContainer.Destroy;
begin
 { set to nil by SetSomeSpecialControl, to detach free notification }
 SomeSpecialControl := nil;
 inherited;
end;

```

---

## 5.6. ##### (Castle Game Engine)

```

Castle Game Engine ##### ##
TFreeNotificationObserver ## ##### CastleClassUtils #####
FreeNotification, RemoveFreeNotification
Notification.

TFreeNotificationObserver ##### ##-
FreeNotification ##### (#####

```

```
#####, ## # ##### ## #). ## #-#####, ##### #
#####
TFreeNotificationObserver # ##### #-##### ## ##### (#####
FreeNotification # ##### ##### ## ##### #####,
#####).
```

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

---

## type

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

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

### private

```
FSomeSpecialControlObserver: TFreeNotificationObserver;
```

```
FSomeSpecialControl: TControl;
```

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

```
procedure SomeSpecialControlFreeNotification(const Sender:
```

```
TFreeNotificationObserver);
```

### public

```
constructor Create(AOwner: TComponent); override;
```

```
property SomeSpecialControl: TControl
```

```
read FSomeSpecialControl write SetSomeSpecialControl;
```

```
end;
```

## implementation

```
uses CastleComponentSerialize;
```

```
constructor TContainer.Create(AOwner: TComponent);
```

```
begin
```

```
inherited;
```

```
FSomeSpecialControlObserver := TFreeNotificationObserver.Create(Self);
```

```
FSomeSpecialControlObserver.OnFreeNotification := {$ifdef FPC}@{$endif}
```

```
SomeSpecialControlFreeNotification;
```

```
end;
```

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

```
begin
```

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

```
begin
```

```
FSomeSpecialControl := Value;
```

```
FSomeSpecialControlObserver.Observed := Value;
```

```

end;
end;

procedure TContainer.SomeSpecialControlFreeNotification(const Sender:
 TFreeNotificationObserver);
begin
 // set property to nil when the referenced component is freed
 SomeSpecialControl := nil;
end;

```

---

##### [https://castle-engine.io/custom\\_components](https://castle-engine.io/custom_components) .

## 6. #####

### 6.1. #####

#####

- ##### **raise**. ##### **raise ...**, #####

- ##### **try ... except ... end**. ##### **"** #####

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

# # # LCL ##### (events) #####  
 ##### (# LCL #####), #####  
 #####.

# # Castle Game Engine ##### CastleWindow,  
 #####  
 (# #####).

# #####  
 #####(##### # LCL ###, # CGE ###...  
 ).



```

 raise EInvalidParameter.Create('Invalid parameter, space is not
allowed');
end;

```

```

,## ##### raise ##### ## ####
####. ##### ## ##### ## ##### ##
#####.

```

```

CreateFmt, ##### #
Create(Format(MessageFormat, MessageArguments)).
#####
#####. ##### ## ##### #####:

```

```

type
 EInvalidParameter = class(Exception);

function ReadParameter: String;
begin
 Result := Readln;
 if Pos(' ', Result) <> 0 then
 raise EInvalidParameter.CreateFmt('Невалиден параметър %s, не са
позволен интервали.', [Result]);
end;

```

### 6.3. #####

```

#####:

```

```

var
 Parameter1, Parameter2, Parameter3: String;
begin
 try
 Writeln('Въведете 1-ви параметър:');
 Parameter1 := ReadParameter;
 Writeln('Въведете 2-ри параметър:');
 Parameter2 := ReadParameter;
 Writeln('Въведете 3-ти параметър:');
 Parameter3 := ReadParameter;
 except
 // прихващане на EInvalidParameter предизвикан от някое от
извикванията на ReadParameter
 on EInvalidParameter do
 Writeln('Възникна изключение EInvalidParameter');

```

```
end;
end;
```

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

```
try
...
except
 on E: EInvalidParameter do
 Writeln('Възникна изключение EInvalidParameter със съобщение: ' +
 E.Message);
 end;
```

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

```
try
...
except
 on E: EInvalidParameter do
 Writeln('Възникна изключение EInvalidParameter със съобщение: ' +
 E.Message);
 on E: ESomeOtherException do
 Writeln('Възникна изключение ESomeOtherException със съобщение: ' +
 E.Message);
 end;
```

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

```
try
...
except
 Writeln('Предупреждение: Възникна изключение');
end;
// ПРЕДУПРЕЖДЕНИЕ: НЕ СЛЕДВАЙТЕ ПРИМЕРА БЕЗ ДА СТЕ ПРОЧЕЛИ ЗАБЕЛЕЖКАТА ПО-
ДОЛУ
// ОТНОСНО "ПРИХВАЩАНЕ НА ВСИЧКИ ИЗКЛЮЧЕНИЯ"
```

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

##### # ##### (##### Exception  
 ### ##### TObject ), ### #####  
 ##### # #-  
 ##### # #####. #####  
 ##### # #####, #####, #####  
 ## #####, # ##### ## ## #####, # ##### ## ##  
 ## #####. #-#####, ## ##### ## ##  
 #####, ## ##### ## #####.

- #####? #####  
 ##### ## ##.
- #####? ##### ##  
 ####, ## ## ## ##.

#### ##### #:

```
try
...
except
 on E: TObject do
 Writeln('Предупреждение: Възникна изключение');
end;
// ПРЕДУПРЕЖДЕНИЕ: НЕ СЛЕДВАЙТЕ ПРИМЕРА БЕЗ ДА СТЕ ПРОЧЕЛИ ЗАБЕЛЕЖКАТА ПО-
ГОРЕ
// ОТНОСНО "ПРИХВАЩАНЕ НА ВСИЧКИ ИЗКЛЮЧЕНИЯ"
```

##### # ##### Exception:

```
try
...
except
 on E: Exception do
 Writeln('Предупреждение: Възникна изключение: ' + E.ClassName + ',
 съобщение: ' + E.Message);
end;
// ПРЕДУПРЕЖДЕНИЕ: НЕ СЛЕДВАЙТЕ ПРИМЕРА БЕЗ ДА СТЕ ПРОЧЕЛИ ЗАБЕЛЕЖКАТА ПО-
ГОРЕ
// ОТНОСНО "ПРИХВАЩАНЕ НА ВСИЧКИ ИЗКЛЮЧЕНИЯ"
```

##### # "#####" ##### except ... end, ### #  
 #####. ##### # raise E; , ### # E, #####  
 ##### # raise ## ##.

```
try
...
except
 on E: EInvalidSoundFile do
 begin
 if E.InvalidUrl = 'http://example.com/blablah.wav' then
 Writeln('Предупреждение: зареждането на http://example.com/
blablah.wav се провали, игнорирайте го')
 else
 raise;
 end;
 end;
end;
```

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

## 6.4. Finally (##### ## ## ##### ## #####)

##### try .. finally .. end , ## #####  
 ## ##### ## , ##### # ##### ##  
 ##### ##. ##### ## ##### # #####:

```
procedure MyProcedure;
var
 MyInstance: TMyClass;
begin
 MyInstance := TMyClass.Create;
 try
 MyInstance.DoSomething;
 MyInstance.DoSomethingElse;
 finally
 FreeAndNil(MyInstance);
 end;
end;
```

#### ##### # ## ##### ## ## ,  
 #### ## MyInstance.DoSomething ## MyInstance.DoSomethingElse  
 #####.



#####  
MyInstance ##-####, ##### (#### ##  
##### "##### # #####") #####. #####  
##### ##  
##### ## ## ## ##:

---

// НЕКОРЕКТЕН ПРИМЕР:

```
procedure MyProcedure;
var
 MyInstance: TMyClass;
begin
 try
 CallSomeOtherProcedure;
 MyInstance := TMyClass.Create;
 MyInstance.DoSomething;
 MyInstance.DoSomethingElse;
 finally
 FreeAndNil(MyInstance);
 end;
end;
```

---

##### # #####: ### ##### # TMyClass.Create  
(#####) ### #  
## CallSomeOtherProcedure, ##### MyInstance ##  
##### FreeAndNil(MyInstance) ## ##  
##### MyInstance, ##### ##-##### ## ## Access  
Violation (Segmentation Fault). ##### ##  
#####  
#####  
#####.

##### ##  
##### nil (##### FreeAndNil #  
#####). #####  
##### ##-##### ## ##:

---

```
procedure MyProcedure;
var
 MyInstance1: TMyClass1;
 MyInstance2: TMyClass2;
 MyInstance3: TMyClass3;
begin
 MyInstance1 := TMyClass1.Create;
 try
```

```

MyInstance1.DoSomething;

MyInstance2 := TMyClass2.Create;
try
 MyInstance2.DoSomethingElse;

 MyInstance3 := TMyClass3.Create;
 try
 MyInstance3.DoYetAnotherThing;
 finally
 FreeAndNil(MyInstance3);
 end;
finally
 FreeAndNil(MyInstance2);
end;
finally
 FreeAndNil(MyInstance1);
end;
end;

```

---

##### ##-##### ##-#####:

---

```

procedure MyProcedure;
var
 MyInstance1: TMyClass1;
 MyInstance2: TMyClass2;
 MyInstance3: TMyClass3;
begin
 MyInstance1 := nil;
 MyInstance2 := nil;
 MyInstance3 := nil;
 try
 MyInstance1 := TMyClass1.Create;
 MyInstance1.DoSomething;

 MyInstance2 := TMyClass2.Create;
 MyInstance2.DoSomethingElse;

 MyInstance3 := TMyClass3.Create;
 MyInstance3.DoYetAnotherThing;
 finally
 FreeAndNil(MyInstance3);
 FreeAndNil(MyInstance2);
 FreeAndNil(MyInstance1);
 end;

```

---



```

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));
 finally
 FreeAndNil(S);
 end;

 WriteLn('Read from file got integer: ', OutputInt);
end.

```

---

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

---

```

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

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

S := Download('castle-data:/gui/my_image.png');

```

---

**## ## #####, ##### ## #####**  
**TStreamReader. ### ##### API # ##### # ##### TStream.**  
**##### TStreamReader ##### ## ##### URL ##### ## #####**  
**## ##### ## ##### TStream.**

---

```
Text := TTextReader.Create('castle-data:/my_data.txt');
try
 while not Text.Eof do
 WriteLnLog('NextLine', Text.ReadLn);
 finally
 FreeAndNil(Text);
 end;
```

## 7.2. ##### (#####, #####), #####

##### # run-time #####.   
 ### ##### "#####" ##### (#### TList # TObjectList ## #####   
 Contnrs), ### # ##### (array of TMyType). ## ## ##   
 ###-##### # #####, ##### ## #####   
 ## #####.

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

# ##### ## ##, ##### # FPC:

- ##### Generics.Collections (## FPC >= 3.2.0)
- ##### FGL
- ##### GVector (##### # fcl-stl)

##### ## ## Generics.Collections. #####   
 ##### ##:

- #####,
- ##### (##### ## #####<sup>5</sup> # #####),
- ##### FPC # Delphi,
- ##### # #####  
 (##### Contnrs).

# Castle Game Engine: ### ##### Generics.Collections #   
 ##### Generics.Collections # #####!

<sup>5</sup> ##### = Dictionary, a.k.a. Associative array

###-##### ## Generics.Collections ##:

## TList

##### ## #####.

## TObjectList

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

## TDictionary

#####<sup>5</sup>.

## TObjectDictionary

#####, ##### "#####" ##### #/###.

### ## ##### TObjectList:

---

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

type
 TApple = class
 Name: string;
 end;

 TAppleList = specialize TObjectList<TApple>;

var
 A: TApple;
 Apples: TAppleList;
begin
 Apples := TAppleList.Create(true);
 try
 A := TApple.Create;
 A.Name := 'my apple';
 Apples.Add(A);

 A := TApple.Create;
 A.Name := 'another apple';
 Apples.Add(A);

 Writeln('Count: ', Apples.Count);
 Writeln(Apples[0].Name);
 Writeln(Apples[1].Name);
 finally FreeAndNil(Apples) end;
```

end.

```
, ##
(####. #### Sort # IndexOf).
Generics.Collections
(#
#####
IndexOf).
```

```
#####
#####. ##### # ##### # ##### IComparer.
#####
TComparer<T>.Construct, ## # ##### # ##### # ##### #
IComparer. ##### # ##### # #-####:
```

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

```
{ If GENERICS_CONSTREF is defined, then various routines used with
 Generics.Collections
 (like callbacks we pass to TComparer, or OnNotify callback or Notify
 virtual method)
 should have "constref" parameter, not "const".
 This was the case of FPC<= 3.2.0, FPC changed it in
 https://gitlab.com/freepascal.org/fpc/source/-/
 commit/693491048bf2c6f9122a0d8b044ad0e55382354d .
 It is also applied to FPC fixes branch 3.2.3. }
{$ifdef VER3_0} {$define GENERICS_CONSTREF} {$endif}
{$ifdef VER3_2_0} {$define GENERICS_CONSTREF} {$endif}
{$ifdef VER3_2_2} {$define GENERICS_CONSTREF} {$endif}
```

```
uses SysUtils, Generics.Defaults, Generics.Collections;
```

```
type
```

```
 TApple = class
 Name: string;
 end;
```

```
 TAppleList = specialize TObjectList<TApple>;
```

```
function CompareApples(
```

```
 {$ifdef GENERICS_CONSTREF} constref {$else} const {$endif}
 Left, Right: TApple): Integer;
```

```
begin
```

```

 Result := AnsiCompareStr(Left.Name, Right.Name);
end;

type
 TAppleComparer = specialize TComparer<TApple>;
var
 A: TApple;
 L: TAppleList;
begin
 L := TAppleList.Create(true);
 try
 A := TApple.Create;
 A.Name := '11';
 L.Add(A);

 A := TApple.Create;
 A.Name := '33';
 L.Add(A);

 A := TApple.Create;
 A.Name := '22';
 L.Add(A);

 L.Sort(TAppleComparer.Construct(@CompareApples));

 Writeln('Count: ', L.Count);
 Writeln(L[0].Name);
 Writeln(L[1].Name);
 Writeln(L[2].Name);
 finally FreeAndNil(L) end;
end.

```

##### TDictionary #####, ##### map (key → value), #####  
 ##### associative array. ##### API # ##### TDictionary # C#.  
 #####, ##### →#####.

#####:,

```

{$mode objfpc}{$H+}{$J-}
uses SysUtils, Generics.Collections;

type
 TApple = class
 Name: string;
 end;

```



```

TAppleDictionary = specialize TDictionary<string, TApple>;

var
 Apples: TAppleDictionary;
 A, FoundA: TApple;
 ApplePair: TAppleDictionary.TDictionaryPair;
 AppleKey: string;
begin
 Apples := TAppleDictionary.Create;
 try
 A := TApple.Create;
 A.Name := 'моята ябълка';
 Apples.AddOrSetValue('ключ за ябълка 1', A);

 if Apples.TryGetValue('ключ за ябълка 1', FoundA) then
 Writeln('Намерена ябълка с ключ "ключ за ябълка 1" с име: ' +
 FoundA.Name);

 for AppleKey in Apples.Keys do
 Writeln('Намерен ключ за ябълка: ' + AppleKey);
 for A in Apples.Values do
 Writeln('Намерена ябълка с име: ' + A.Name);
 for ApplePair in Apples do
 Writeln('Намерен ключ за ябълка->име на ябълка: ' +
 ApplePair.Key + '->' + ApplePair.Value.Name);

 { Долният ред също работи, но може да се използва само да
 зададе стойност на *съществуващ* ключ в речника.
 Вместо това обикновено се използва AddOrSetValue
 за да се зададе или добави нов ключ ако е необходимо. }
 // Apples['ключ за ябълка 1'] := ... ;

 Apples.Remove('ключ за ябълка 1');

 { Забележете, че TDictionary не притежава елементите си
 и трябва да ги освобожавате ръчно.
 Може да използвате TObjectDictionary за да имате автоматичен
 режим за притежание. }
 A.Free;
 finally FreeAndNil(Apples) end;
end.

```

---

```

TObjectDictionary ##### #/###, #####
#####. #####

```

#####/#####. ##, ##  
 ## Integer (##. ##  
 ## Integer, # doOwnsKeys), ##  
 #####.

##### TObjectDictionary # ##-##.  
 ##### # *memory leak detection*, ####. #### fpc -gl -gh  
 generics\_object\_dictionary.lpr, ## ##, ##  
 #####.

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

uses SysUtils, Generics.Collections;

type

TApple = class  
 Name: string;  
end;

TAppleDictionary = specialize TObjectDictionary<string, TApple>;

var

Apples: TAppleDictionary;  
 A: TApple;  
 ApplePair: TAppleDictionary.TDictionaryPair;

begin

Apples := TAppleDictionary.Create([doOwnsValues]);

try

A := TApple.Create;  
 A.Name := 'my apple';  
 Apples.AddOrSetValue('apple key 1', A);

for ApplePair in Apples do

Writeln('Found apple key->value: ' +  
 ApplePair.Key + '->' + ApplePair.Value.Name);

Apples.Remove('apple key 1');

finally FreeAndNil(Apples) end;

end.

.....  
 ##### FGL #####  
 Generics.Collections, ##-##### FGL ##:

## TFPGList

#####.

## TFPGObjectList

#####. ##### "#####"  
#####.

## TFPGMap

#####<sup>5</sup>.

# ##### FGL, TFPGList ##### # #####, #####  
##### (=). ### TFPGMap #####  
##### "##-#####" (>) # "##-#####" (<). ###  
##### # #####, #####  
##### (#####), ## #####  
##### # ##### 8.9, „#####“.

# **Castle Game Engine** ##### CastleGenericLists, #####  
##### TGenericStructList # TGenericStructMap. ##  
##### TFPGList # TFPGMap, ## #####  
##### (##### # #####  
##### # ##### # #####). ##  
6.3 ##### CastleGenericLists # ##### (deprecated) #  
##### Generics.Collections #####.

##### 8.3, „#####“.

## 7.3. #####: TPersistent.Assign

##### :=  
#####.

```
var
 X, Y: TMyObject;
begin
 X := TMyObject.Create;
 Y := X;
 // X и Y сега са два указателя към една и съща инстанция
 Y.MyField := 123; // ще се промени също и X.MyField
 FreeAndNil(X);
end;
```

```
##,
TPersistent, # ## #####
Assign. ##### ## ##### ## TMyObject, ## ## ##
#####:
```

---

```
var
 X, Y: TMyObject;
begin
 X := TMyObject.Create;
 Y := TMyObject.Create;
 Y.Assign(X);
 Y.MyField := 123; // това не променя X.MyField
 FreeAndNil(X);
 FreeAndNil(Y);
end;
```

---

```
#####, ##### # ##### ## ##### Assign #####
#####. #####
Assign, ## ## #####, ##### ## ## ##### ##
####.
```

---

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

type
 TMyClass = class(TPersistent)
 public
 MyInt: Integer;
 procedure Assign(Source: TPersistent); override;
 end;

 TMyClassDescendant = class(TMyClass)
 public
 MyString: string;
 procedure Assign(Source: TPersistent); override;
 end;

procedure TMyClass.Assign(Source: TPersistent);
var
 SourceMyClass: TMyClass;
begin
 if Source is TMyClass then
```

```

begin
 SourceMyClass := TMyClass(Source);
 MyInt := SourceMyClass.MyInt;
 // Xxx := SourceMyClass.Xxx; // копирайте още полета ако е
необходимо ...
end else
 { Поради това, че TMyClass е директен наследник на TPersistent,
 той извиква inherited САМО когато не знае как да обработи Source.
 Виж коментарите по-долу. }
 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; // копирайте още полета ако е
необходимо ...
 end;

 { Поради това, че TMyClassDescendant има предшественик, който вече е
 заменил Assign (in TMyClass.Assign), той извиква inherited ВНАГИ,
 за да позволи TMyClass.Assign да копира останалите полета.
 Виж коментарите по-долу за детайлно обяснение. }
 inherited Assign(Source);
 end;

var
 C1, C2: TMyClass;
 CD1, CD2: TMyClassDescendant;
begin
 // тест TMyClass.Assign
 C1 := TMyClass.Create;
 C2 := TMyClass.Create;
 try
 C1.MyInt := 666;
 C2.Assign(C1);
 WriteLn('C2 state: ', C2.MyInt);
 finally
 FreeAndNil(C1);
 FreeAndNil(C2);
 end;
end;

```

```

end;

// test TMyClassDescendant.Assign
CD1 := TMyClassDescendant.Create;
CD2 := TMyClassDescendant.Create;
try
 CD1.MyInt := 44;
 CD1.MyString := 'blah';
 CD2.Assign(CD1);
 WriteLn('CD2 state: ', CD2.MyInt, ' ', CD2.MyString);
finally
 FreeAndNil(CD1);
 FreeAndNil(CD2);
end;
end.

```

---

```

##-##### ## ##### AssignTo # #####,
Assign # #####, ## #####.

#####, ##### inherited # ##### Assign.
#####:

TPersistent.(### ## #
TPersistent, ## ##### ##### Assign.)
inherited
(## ##### TPersistent.Assign) ##### ##
#####.

Assign.
inherited (## ## ##### Assign).
inherited # #####.

(##### # ##
inherited ## ##### Assign) # ##
AssignTo, ##### TPersistent.Assign
TPersistent.AssignTo #####:

```

---

```

procedure TPersistent.Assign(Source: TPersistent);
begin
 if Source <> nil then
 Source.AssignTo(Self)
 else

```

```

 raise EConvertError...
end;

procedure TPersistent.AssignTo(Destination: TPersistent);
begin
 raise EConvertError...
end;

```



```

TPersistent.#####
FPC #####,
#####
#####.

```

#####, #####

- ##### Assign, ##### AssignTo #####
- ##### #####, ##### TPersistent, ##### (#####) #####. ##### Assign ##### RTTI (#####) #####.

```

TApple, ##### TApple.Assign
#####
TApple (##### TApple, ##### TFruit).
TApple.Assign ##### Source is
TApple #
inherited, ##### TFruit
#####.

```

```

TFruit.Assign # TApple.Assign
#####

```

- ##### TApple # TApple.Assign, #####
- ##### TOrange # TApple.Assign, ##### TOrange # TApple. # - ##### TFruit.

- `##### TWerewolf ## TApple.Assign, ##  
## ##### (##### TApple.Assign ## #####  
TFruit.Assign, ##### ## ##### TPersistent.Assign, ##### ##  
#####).`



```
#####, ## ##### TPersistent, ##
published, ## ##
TPersistent.

published. ## ##### # ## #
#####, ##### ## public.
4.5, „#####“.
```

## 8. #####

### 8.1. #####

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

```
(#### # #####) #####
#####, ##### # #####
#####, ##### ## ##
#-##### ## #
— ## ##### (#
#####) ##### # #####, #####
#####.
```

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

```
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;
end;

begin
 Result := 0;
 for I := 0 to N do
 Result := Result + Square;
 end;
```

##### — #####  
##### , ## ##### # ##### # #####  
##### . ##### (## ## , ## #####  
##### , ## ##### :).

## 8.2. Callbacks (##### , ##### )

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

Callback-## ## ##:

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

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

```

function Add(const A, B: Integer): Integer;
begin
 Result := A + B;
end;

function Multiply(const A, B: Integer): Integer;
begin
 Result := A * B;
end;

type
 TMyFunction = function (const A, B: Integer): Integer;

function ProcessTheList(const F: TMyFunction): Integer;
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);
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 callback, ##  
 ## ##### ## ##### ## ##### ## ## ##  
 ##### 9.3, „#####“ ## #####.

---

```

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;

```

---

##### @TMyClass(nil).Add  
##### @TMyClass.Add.

- (#####) #####: ##### # is nested # #### # ##  
#####, ## ##### { \$modeswitch nestedprocvars } .  
## ##### # ##### 8.1, „##### (#####) #####“.

### 8.3. #####

#####. ##### # ####  
(#####) ##### # ##### # ####. ###-#####  
##### #, ##### # ##### (#####, #####, #####,  
####...): ##### # ##### # ##### # T, # ##### # ##  
##### # # ##### # ##### # ##### #, ##### #  
#####, ##### # ##### TMyRecord # #.

##### # Pascal ##### # C++. #####,  
## # "#####" # #####, ##### #  
(## # ##### #-##### # #; ##### # #####  
## # #, # # #, ##### #  
"#####" ##### # #). ##  
#### #, ## # # (##### # #  
##### #) # # # #. #####  
##### # # # (#####, float), #####  
#####, ##### # #.

---

```

{ $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;

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('FloatCalc: ', FloatCalc.Value:1:2);
 finally
 FreeAndNil(FloatCalc);
 end;

 StringCalc := TMyStringCalculator.Create;
 try
 StringCalc.Add('something');
 StringCalc.Add(' more');
 WriteLn('StringCalc: ', StringCalc.Value);
 finally
 FreeAndNil(StringCalc);
 end;
end.

```

---

```

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

```

---

```

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

```

{ Note: this example requires FPC 3.1.1 (will not compile with FPC 3.0.0 or older). }

```
generic function Min<T>(const A, B: T): T;
begin
 if A < B then
 Result := A else
 Result := B;
end;

begin
 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.
```

##### 7.2, „##### (#####, #####), #####“  
#####.

## 8.4. Overloading

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

## ##### overloading-## ##### FPC #####, #####  
## ##### # ##### (##### unit) ##  
##### # ##### # ##### # ##-  
#####. #####, ### ##### Foo(Integer)  
# Foo(string) # ##### Foo(Float), #####  
##### Foo(Float)  
##### (## --- ##) #####  
###-#####). ## #####  
overload.

## 8.5. #####

##### ##:

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

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

##### # ## # ##. ##### # ##  
##### # .....  
## # ##. ##### # ##  
##### # ##, #####, ## # "#####"  
##### # ##### Pascal. ##### # ##, ## # "#####"

---

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

```
{ $ifdef FPC }
{ Това е дефинирано само ако се компилира с FPC, не с други компилатори
 (напр. Delphi). }
procedure Foo;
{ $endif }
```

{ Дефиниране на константата NewLine. Тук може да видите как нормалния синтаксис на Паскал се "чупи" с препроцесорните директиви. Когато компилирате за Unix (вкл. Linux, Android, Mac OS X), компилатора вижда това:

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

Когато компилирате за Windows, компилатора вижда това:

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

За други операционни системи, кодът няма да се компилира, защото компилатора вижда това:

```
const NewLine = ;
```

\*Хубаво е\*, че компилирането се проваля в този случай -- така ако трябва да пригледите програмата към ОС, която не е Unix или Windows, компилатора ще ви припомни да изберете конвенция за нов ред (newline) за тази система. }

```
const
 NewLine =
 { $ifdef UNIX } #10 { $endif }
```





- ##### # # # # ##### unit # # ##### , ##### #  
##### # # # # unit ##### . # # # # #  
##### - ##### # # ##### # # # # # unit # #  
##### unit-# , # # # # ##### unit # # ##### .  
### ### # # ##### . ##### # # # "#####"  
## ## # unit-## , #### # # # # # # # # #  
### #####. ##### , #### # # #-##### # # ##### unit # "#####"   
##### UI #####" ##### # # ##### ## ## unit # # ## UI  
##### ## , ## ## ##### # # ##### uses #####  
(### ### ##### UI # # ##### # # ##### UI #####). # # #####  
## ##### ## UI ##### # ##### myunit.pas # # # #  
#### # # # # # # # # # # # # # # # # # # # #  
##### # # # # # # # # # # # # # # # # # # # #

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

## 8.6. #####

---

81



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

- ##### ## #####, #### ##  
#####.
- ##### ## ##### (#####  
#####) # #####: ##### C layout  
##### packed record. #####:

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

- ##### case, ##### unions # C-  
##### ## ## ## ## ##  
##### ##, # #####. #####  
##### # #####. # #####, #####  
#####).

## 8.7. #####

##### Turbo Pascal #####  
## ##, ##### object. #####  
##### "#####" # ##### "####".

- ##### / #####  
##### / #####.
- ## ## ##### # #####,  
##### "#####" ##  
(#####) ## ##, #####  
##### ##, #####  
#####.
- #####, #####  
##### — #####

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

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

## 8.8. #####

##### ## ##### ##### ## #####. ##### ##  
TMyRecord ## ##### ^TMyRecord # ## #####  
PMyRecord. #-##### # ##### ## #####  
#####, #####:

**type**

```
PMyRecord = ^TMyRecord;
TMyRecord = record
 Value: Integer;
 Next: PMyRecord;
end;
```

##### ## ##### (### PMyRecord ##  
# ## TMyRecord, ##### TMyRecord #  
## PMyRecord). ##### # ## ##  
##### ##, ##### ## ##  
type.

##### # ##### # #####  
##### New # Dispose ## (## #-#####  
##### GetMem # FreeMem. ## ##  
##### ^ (например `MyInteger :=  
MyPointerToInteger^). ## ##  
##### ##  
#####-##### @ (##### MyPointerToInteger := @MyInteger).

## ## Pointer, ##### ## void\* # C-#####. ##  
# ##### # ## ##  
## #####.

#####  
#####  
#####  
#####

type

```
TMyClass = class
 Value: Integer;
 Next: TMyClass;
end;
```

## 8.9. #####

#####  
#####  
#####

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

uses

```
StrUtils;
```

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

begin

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

end;

begin

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

```

---

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

---

```

{$mode objfpc}{$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.

```

---

```

{$modeswitch
advancedrecords} # ## ##### ##### class operator #####
#####. ##### ## ## ##### #####,
(#### TFPGList,
#####) # #####. #
"#####" ##### ## ##### (#####) ####
(##### # # ##### # ####, ##### ##### TFPGList) # ####
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.

```

---

## 9. ##### ## #####

### 9.1. #####

##### `private` #####, ## ##### (##) ## # ##### #####  
 #####, # ##### # #####. ##### ##### #####: #####  
 # ##### ##### ## ##### # ##### # #####. ##### ##  
 C++ ## ##### ## ####, ## ##### # ##### ## "#####"<sup>6</sup>. ####  
 ##### # ##### # ## ##### ##### # #####  
 ##### # # ##### ## #### #####.

## #####, ## ##### ##### # #####, ##### ## ##  
 ##### # ####, # ##-##### ## ##### `strict`  
`private`. ## ##### # ##### ## ##### (##) #### #  
 ##### ## #####. ## #####.

##### — ##### `protected` #####, ## ##### ##  
 # ##### # "#####" # #####, ##### `strict`  
`protected`, ## # ##### ## #####.

---

<sup>6</sup> ##### = friends





```

procedure TMyClass.TInternalClass.DoSomething;
begin
end;

```

### 9.3. #####

####, ##### ( 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;

```

##### - #####  
##### 9.4, „#####“.

##### 4.5, „#####  
#####“ ##### `private` or `protected` #####.

#####  
##### MyInstance := TMyClass.Create(...);  
#####  
#####. #####  
##### "#####", ##### (#####  
#####) #####  
##### (#####).



#####  
#####.

---

**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;
```

**var**

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

**begin**

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

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

```
{ Това ще предизвика изключение по време на изпълнение
 защото DoSomething е абстрактен в TMyClass. }
```

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

**end**;

---

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

```
ClassType ##
#####. ##### ##
```

Clone, ##### 7.3, „#####: TPersistent.Assign“  
 #####, ##### "#####"

TComponent, TComponent.Create(AOwner: TComponent).

type

```
TMyClass = class(TComponent)
 procedure Assign(Source: TPersistent); override;
 function Clone(AOwner: TComponent): TMyClass;
end;
```

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

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

```
begin
```

```
 // Това трябва винаги да създаде инстанция точно от клас TMyClass:
 //Result := TMyClass.Create(AOwner);
 // Това може потенциално да създаде инстанция от наследник на TMyClass:
 Result := TMyClassRef(ClassType).Create(AOwner);
 Result.Assign(Self);
```

```
end;
```

## 9.5. #####

##### (#####).  
 ##### (#####).  
 Self #  
 #####: #####  
 ##### (#####)

#####  
 # #####:

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

type

TMyCallback = **procedure** (A: **Integer**);

TMyClass = **class**

**class procedure** Foo(A: **Integer**);

end;

**class procedure** TMyClass.Foo(A: **Integer**);

**begin**

end;

var

Callback: TMyCallback;

**begin**

// Грешка: TMyClass.Foo не е съвместим с TMyCallback

Callback := @TMyClass(**nil**).Foo;

**end.**



```
Delphi ##### ## ##### ##
TMyClass.Foo ##### TMyClass(nil).Foo #####
#####. ##### ## ## #####, ## TMyClass.Foo
##-##### # ##### ## ##### ##-
#####. ##### ## TMyClass(nil).Foo
##... ## ##### (#####) # ##### ObjFpc,
#####.

#####, ##### ## TMyClass.Foo
Callback ##-#### ## ##### Delphi, #####
#####.
```

```
#####, ##### ## Callback ## #
Foo. ##### ##, ##### ##### Foo ###
implicit ##### ## #####.
```

```
##
TMyCallback ## #####: TMyCallback = procedure (A: Integer) of
object; . ## ##### ## ## #####.
```

```
static. ## #####
/ #####, # #####, ## #####
#####. ## ##### ##### ## ## (##
#####).
```

#####  
#####. #####:

```

{$mode objfpc}{$H+}{$J-}
type
 TMyCallback = procedure (A: Integer);

 TMyClass = class
 class procedure Foo(A: Integer); static;
 end;

class procedure TMyClass.Foo(A: Integer);
begin
end;

var
 Callback: TMyCallback;
begin
 Callback := @TMyClass.Foo;
end.

```

## 9.6. #####

##### class var #####  
#####  
#####.

#####  
#####  
##### class property ##### property # # ##### getter  
# / ### setter, ##### ##### # # #####-#####. ##### 9.5,  
„#####“.

##### (##### 4.3, „#####“), #####  
#####-#####, ##### # # #####. #####  
#####.

```

{$mode objfpc}{$H+}{$J-}
type
 TMyClass = class
 strict private

```

```
// Alternative:
// FMyProperty: Integer; static;
class var
 FMyProperty: Integer;
class procedure SetMyProperty(const Value: Integer); static;
public
 class property MyProperty: Integer
 read FMyProperty write SetMyProperty;
end;

class procedure TMyClass.SetMyProperty(const Value: Integer);
begin
 Writeln('MyProperty changes!');
 FMyProperty := Value;
end;

begin
 TMyClass.MyProperty := 123;
 Writeln('TMyClass.MyProperty is now ', TMyClass.MyProperty);
end.
```

## 9.7. #####

##### # ##### ##### ### ##### ##### # #####a. #####  
 ## ##### ##### ##### MyInstance.MyMethod(...). ##  
 ##### ##### ## #####, ## ### ##### ## #####  
 Action # ##### 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);
 end;
```

```
####, ##, ##
X.Action(...),
Render(X, ...).##
X.Render(...), ## Render
TMy3DObject.
```

```
##, ##, ##
/, ##,
-
TMy3DObject.
```

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

procedure TMy3DObjectHelper.Render(const Color: TColor);
var
 I: Integer;
begin
 { забележете, че тук достъпваме ShapesCount и Shape без да ги
 квалифицираме }
 for I := 0 to ShapesCount - 1 do
 RenderMesh(Shape[I].Mesh, Color);
 end;
```



##-##### # "##### ##". #### ##  
 ## #####, ####  
 ##### enum. ##### "#####  
 #####" ## (#####...) #####. ##### <http://lists.freepascal.org/fpc-announce/2013-February/000587.html> .

## 9.8. #####, #####

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

##### Create.

##### — #####  
CreateMy, ##### Create, # #####  
##### Create # #####, #####  
CreateMy #####.

# TObject # # # # #, #####  
##### #. #####  
##### (#####: # # overload, #####  
#####).

# TComponent ##### constructor  
Create(AOwner: TComponent);. # # # # #,  
##### # # # # #,  
##### (### 9.4, „#####“ #-###).

## 9.9. #####

##### # # # # #  
#####?

---

X := TMyClass.Create;

---

##### X # # # # # ...  
#####?

##### Object Pascal #, # # # # #, # # # # # # # # # #  
##### # # # # #. # # # # #, # # # # #  
##### # # # # #, #. # # # # # # # # # #  
##### # # # # #. ##### # # #  
##### FreeAndNil.

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

```
#####, ## # ##### ##### ##### ## nil, ##### ##### ##
0 # #####.
```

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

---

```
{ $mode objfpc } { $H+ } { $J- }
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('Предизвикано изключение от конструктор!');
 Gun2 := TGun.Create;
end;

destructor TPlayer.Destroy;
begin
 { в случай, че конструктора крашне, бихме могли
 да имаме ситуация с Gun1 <> nil и Gun2 = nil. Справете се с това.
 ... Всъщност в случая FreeAndNil ще се справи без
 допълнителни усилия от наша страна, защото FreeAndNil проверява
 дали инстанцията е nil преди да извика деструктора. }
 FreeAndNil(Gun1);
 FreeAndNil(Gun2);
 inherited;
end;

begin
 try
 TPlayer.Create;
 except
 on E: Exception do
 WriteLn('Уловено ' + E.ClassName + ': ' + E.Message);
```

end;  
end.

---

## 10. #####

### 10.1. #### (CORBA) #####

##### (API<sup>7</sup>), ## ##, ## ##  
#####. ##### ## ##  
#####, ## ## ## ##.

#### ## ##, #####  
##### # ##### ## ##. ####  
##### ## ## ##, #####  
## ##, ## ## ## ##. #####  
##### # ##### C++.

CORBA ##### # #####  
## ##### # 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;
```

---

<sup>7</sup> API = Application Program Interface

```
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('Shooting... ');
 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);
 // The "C3 is IMyInterface" below is false,
 // so "UseThroughInterface(C3 as IMyInterface)" will not execute.
 if C3 is IMyInterface then
 UseThroughInterface(C3 as IMyInterface);
 finally
 FreeAndNil(C1);
 FreeAndNil(C2);
```

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

## 10.2. ##### CORBA # COM

#### ##### -#### ##### "CORBA"?

##### CORBA # #####. #-##### ## ## ##### #####. #####  
##### ## "#####". ##### ## #####  
##### ## #####, ## ##### ## ## ##### #  
#### API.

#####, ## ##### ##### ## ## ##### #  
##### CORBA (Common Object Request Broker Architecture) (see [https://en.wikipedia.org/wiki/Common\\_Object\\_Request\\_Broker\\_Architecture](https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture)), ## ## ##  
##### ## ##### # ##.

##### ## # ##### {\$interfaces corba}?

##### #, ##### ## ##### COM #####. ##### ##  
## ##### # {\$interfaces com}, ## ##### ## # #####  
##### # ##### # #####.

## ##### COM #####, ##### ## #####  
##### # #####. CORBA ##### # ##### ##  
##### ## ##### # C# ## Java. COM #####  
## ##### ## #####, ##### ##  
##### # #####.

#####, ## ##### {\$interfaces xxx} ## #####  
## ##### (#### # #####  
interface # ## interface(ISomeAncestor), #.#. ## ##  
#####) ## ##### ## ## ##  
#### ## ## ##, #####  
{\$interfaces xxx}.

#### # COM #####?

COM ##### \_#####  
##### IUnknown \_. ##### IUnknown:

- ##### \_AddRef #  
\_ReleaseRef. #####

##### (reference-counting).

- ##### `QueryInterface`.
- ##### `COM` (Component Object Model).

#### #### ## ##### ## COM #####?

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

## ## #####: **reference-counting**, ##### ##  
 ##### (# ##### # ##), # #####  
 #####. ## ##### (##### ##  
 #####) # #####. ##### ##  
 #####.

- ##### ## # ##### (##### #  
 #####) ##### ## #####.
- ##### ## # ##### ## #####  
 ##### # ##### ## #####.
- ##### ## ##### ## ##### `COM`.

## ##### # #####. #####  
 ## #####-#####, #####:

- ### ##### ## ##### `API`, ## ##  
 ##### ## ##### # #####  
 ## ##### (##### ## ##), ##### `COM`  
 ## \_\_\_\_\_. ##### ## \_\_\_\_\_.  
`_AddRef` # `_ReleaseRef` \_\_\_\_\_, ## ##  
 ##### ## ## #####, ## ##  
 ## ##### ## \_\_\_\_\_. #####  
 #####.
- ### ##### ## #####, ## #####  
`API` ## ##, ##### ##  
 ##### # \_\_\_\_\_, #.#. ##  
 ##### # ##-#####. ## ##  
 (smart pointers) ##### ## ##  
 ##### (## ##).

```
CORBA ##### # #####
{$interfaces corba} ### #####, #####
#####.
```

```
Delphi ##### COM #####, ##### COM
#####, ##### # Delphi.
```

#### ##### CORBA?

```
##. ##### _AddRef / _ReleaseRef. #####
IUnknown. #####, #####
#####, ##### COM
#####.
```

### 10.3. ##### GUIDs

```
GUID ## ##### ['{ABCD1234-...}'], #####
#####. ##, ## ##
#####.
```

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

```
(#####) GUID, #####
is. #, ## true, ##

Supports(ObjectInstance, IMyInterface) ## ##-##, ##
GUID. #####
CORBA, ## # COM, ## FPC 3.0.0.
```

```
##, ## ##, ##### GUID ##
Lazarus ##### GUID (##### Ctrl
+ Shift + G #). #####
https://www.guidgenerator.com/.
```

```

CreateGUID # GUIDToString # RTL. #####-##:
```

---

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



##### - #####

```
SysUtils;

var
 MyGuid: TGUID;

begin
 Randomize;
 CreateGUID(MyGuid);
 WriteLn('[' + GUIDToString(MyGuid) + ']');
end.
```

10.4. ##### # ##### ## ##### (COM)

COM ##### ## #####:

1. ##### # COM (##### ## Windows, ##### # ## Unix ##### XPCOM, ##### ## Mozilla),
2. ##### ## ##### (##### ##### ## ##### #####, ##### ##### ##### ##### ##### ## #####).

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

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

- ##### #### ##### ## ##### ##### ##### \_AddRef, \_Release # QueryInterface. ## ## ##### ##, ##### ## ## # #####. ##### ## ## ## ## ## ## ## ##### ## ##### ## ## ##### reference-counting ## COM ##### (##### ## ##### # # ##### - ##### #####).

```
TInterfacedObject #####
#####.
```

```
TComponent
#####. # Castle
Game Engine ### ## ##### #####
TNonRefCountedInterfacedObject
TNonRefCountedInterfacedPersistent ## ##### ###, ##### https://github.com/castle-engine/castle-engine/blob/0519585abc13e8386cdae5f7dfef6f9659dc9b57/src/base/castleinterfaces.pas .
```

- #####, #####, #####  
#####. #####  
##### (##### reference-counted, ##### \_AddRef #####...), #####  
#####, #####  
##### "7.7" #  
## FPC (<http://freepascal.org/docs-html/ref/refse47.html>).

###-##### ## COM ##### #:

- ##, ## reference-counted,
- ## TInterfacedObject,
- # ##, #####  
#####  
#####.

#### # #####:

.....  
{ \$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
 WriteLn('TMyClass2.Shoot');
end;

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

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

var
 C1: IMyInterface; // COM се грижи за унищожаването
 C2: IMyInterface; // COM се грижи за унищожаването
 C3: TMyClass3; // ВМЕ трябва да се погрижите за унищожаването
begin
 C1 := TMyClass1.Create as IMyInterface;
 C2 := TMyClass2.Create as IMyInterface;
 C3 := TMyClass3.Create;
 try
 UseThroughInterface(C1); // няма нужда от оператор "as"
 UseThroughInterface(C2);
 if C3 is IMyInterface then
 UseThroughInterface(C3 as IMyInterface); // това няма да се изпълни
 finally
 { Променливи C1 и C2 излизат от обхват и тук би трябвало да се
 унищожат автоматично.

 За разлика от тях, C3 е инстанция, която не се управлява от
 интерфейс
 и трябва да се унищожи ръчно. }
 FreeAndNil(C3);
 end;
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('Shooting... ');
 I.Shoot;
end;

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

procedure UseInterfaces;
begin
 if C1 is IMyInterface then
 //if Supports(C1, IMyInterface) then // equivalent to "is" check above
 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);
FreeAndNil(C3);
end;
end.
```

## 10.6. #####

##### CORBA, ##### COM (#####  
##### CORBA).

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

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

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

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

##### CORBA #####.

2. #####:

```
UseThroughInterface(Cx);
```

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

##### #a# ##### # ##### #  
##### TMyClass,  
#####  
##### TMyClass, ##### TMyClass #####  
#####.

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

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

```


#####
```

```
Cx #
TMyClass2),

#####
```

```
#####
```

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

```
// { $interfaces corba } // забележете, че "as" конверсии за CORBA няма да
се компилират
```

```
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);

 // Това не може да с компилира, не е известно дали My е IMyInterface2.
 // UseInterface2(My);
 // UseInterface2(MyClass);

 // Това се компилира и работи.

```



```

UseInterface2(IMyInterface2(My));
// Това не може да с компилира. Преобразуването InterfaceType(ClassType)
се проверява при компилация.
// UseInterface2(IMyInterface2(MyClass));

// Това се компилира и работи.
UseInterface2(My as IMyInterface2);
// Това се компилира и работи.
UseInterface2(MyClass as IMyInterface2);

// Това се компилира но не работи при изпълнение, с грозно "Access
violation".
// UseInterface3(IMyInterface3(My));
// Това не може да с компилира. Преобразуването InterfaceType(ClassType)
се проверява при компилация.
// UseInterface3(IMyInterface3(MyClass));

// Това се компилира но не работи при изпълнение, с хубаво
"EInvalidCast: Invalid type cast".
// UseInterface3(My as IMyInterface3);
// Това се компилира но не работи при изпълнение, с хубаво
"EInvalidCast: Invalid type cast".
// UseInterface3(MyClass as IMyInterface3);

writeln('Край');
end.

```

## 11. #####

Copyright Michalis Kamburelis.

##### ## ## ##### # ## ##### AsciiDoc ## <https://github.com/michaliskambi/modern-pascal-introduction>. ##### #  
 #####, ##### # ##### ## ##### ## ##### ## ##) #####  
 ## ## ##### # ## ## ## GitHub ## ## ##### ## ## [michalis@castle-engine.io](mailto:michalis@castle-engine.io)<sup>8</sup>. ##### WEB ##### # <https://michalis.xyz/>. ##### #  
 # ##### Documentation ## Castle Game Engine website <https://castle-engine.io/>.

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

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

---

<sup>8</sup> <mailto:michalis@castle-engine.io>

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

Thank you for reading!

##### #: #####, 2023