در این مقاله کوتاه سعی کردم برخی از ویژگی های اینترفیس ها در دلفی را بطور ساده توضیح بدم. قبلا از اینکه قلم خوبی ندارم عذرخواهی می کنم.

Interfaces (رابط ها) در Delphi

Interface ها از نسخه ۳ به دلفی اضافه شدند. برای درک بهتر interface ها، ابتدا به کلاس های Abstract و تعریف آنها اشاره ایی خواهیم داشت:

Abstract Classes (کلاس های انتزاعی)

در دلفی، کلاسی که حداقل یکی از متدهای آن با استفاده از رهنمود abstract تعریف شده باشد، کلاس Abstract نامیده می شود.
متدهای abstract متدهایی هستند که توسط یک کلاس والد (parent) تعریف می شوند، اما پیاده سازی نمی شوند. پیاده سازی اینگونه متدها بر عهده کلاس های فرزند (child) می باشد.
در صورتی که یک نمونه (instance) [یک شی (object)] از یک کلاس abstract ایجاد کنید، با هشدار کامپایل مواجه خواهید شد و در صورتی که یکی از متدهای abstract آن شی را فراخوانی کنید، برنامه شما با یک استثنای "*EAbstractError*" متوقف خواهد شد.
در اینجا این سوال ممکن است مطرح شود که اصلا کلاس یا متدی که هیچ پیاده سازیی ندارد و نمی توان از آن یک نمونه شی ایجاد کرد، چه استفاده ایی ممکن است داشته باشد:
کلاس های Abstract در ساختار یک نرم افزار موجب می شوند که برنامه نویسی که پیاده سازی کلاس های فرزند را برعهده دارد، ملزم به پیاده سازی یکسری رفتارهای معین در هر یک از کلاس های خود شود؛ در غیر این صورت، با هشدارهای کامپایلر مواجه می شود. به این ترتیب احتمال اشتباه برنامه نویس (بخصوص در پروژه های بزرگ) کاهش می یابد.
مثال :
فرض کنید قصد داریم دو نوع پنجره طراحی کنیم (پنجره بیضوی و پنجره مستطیلی). هر دو این پنجره ها رفتارهای استانداردی مثل Minimize, Restore و غیره دارند. هر دو این پنجره ها باید ترسیم شوند، اما نحوه ترسیم هر یک از آنها فرق می کند. برای طراحی آنها می توانیم رفتارهای مشترک آنها را در کلاس TBaseWindow قرار دهیم و کلاس های دیگر از همین کلاس پایه، مشتق شوند. برای تعریف کلاس TBaseWindow دو روش می توانیم بکار بریم:
روش اول:
کد:

```
TBaseWindow = class
```

کد:

```
private
 AField : TSomeType;
public
 procedure Draw; virtual;
 procedure Minimize;
 procedure Maximize;
 procedure Restore;
end;
```

متد Draw در کلاس پایه به صورت virtual تعریف شده است، در نتیجه کلاس های TOvalWindow و TRectangularWindow می توانند این متد را override کرده و پیاده سازی خود را اعمال کنند. اگر برنامه نویسان این کلاس ها فراموش کنند که متد Draw را Override کنند، تا زمان اجرای برنامه مشکلی بوجود نخواهد آمد؛ اما در زمان اجرا با فراخوانی متد Draw پنجره ایی رسم نخواهد شد!
روش دوم:
کد:

```
TBaseWindow = class
```

کد:

```
private
```

```
 AField : TSomeType;
public
 procedure Draw; virtual; abstract;
 procedure Minimize;
 procedure Maximize;
 procedure Restore;
end;
```

متد Draw در کلاس پایه بصورت abstract تعریف شده است. اگر برنامه نویسی که کلاس های TOvalWindow و TRectangularWindow را ایجاد می کند، فراموش کند که متد Draw را پیاده سازی کند؛ با فراخوانی متد Draw هشدار زیر را دریافت می کند:

OvalWindow.Draw;
Constructing instance of TOvalWindow contains abstract method TBaseWindow.Draw

Interfaces

اینترفیس ها نوعی قرارداد محسوب می شوند که در آن یک کلاس پیاده سازی تمامی متدهای مشخص شده در قرار داد را برای استفاده سایر کلاس ها یا کاربر تضمین می کند. پیاده سازی این متدها از دید کلاس های استفاده کننده از آنها و کاربر پنهان است. اینترفیس ها شبیه به کلاسی هستند که تمامی متد های آن abstract باشد و هیچ فیلدی برای ذخیره اطلاعات نداشته باشد. اما اینترفیس ها ویژگی هایی دارند که آنها را از کلاس های Abstract جدا می کند:

1 - اینترفیس ها کلاس نیستند، بلکه یک نوع (Type) مستقل در زبان مربوطه (در اینجا Object Pascal) محسوب می شوند.

2 - اینترفیس ها ازTObject مشتق نمی شوند. IUnknown در نسخه های قبلی دلفی (قبل از دلفی ۵) بعنوان والد تمام اینترفیس ها محسوب می شد، اما از نسخه ۵ به بعد، همه اینترفیس ها از IInterface مشتق می شوند. IUnknown و IInterface یکسان هستند. از IUnknown در برنامه های مبتنی بر تکنولوژی COM مایکروسافت استفاده می شود.

۳ - اگر پیدا سازی نکردن یک متد از کلاسabstract ،در کلاس های فرزند، موجب ایجاد هشدار کامپایلر می شد؛ پیاده سازی نکردن هر یک از متدهای یک اینترفیس، توسط کلاسی که آن اینترفیس را پشتیبانی می کند، موجب کامپایل نشدن برنامه می شود.

4 - یک کلاس می تواند از چندین اینترفیس بطور همزمان پشتیبانی کند.

5 - اینترفیس ها قابلیتReference-counting (شمارش مرجع) دارند.

نکته: از اینترفیس ها هم مثل کلاس های abstract نمی توان یک نمونه (شی) ایجاد کرد.

اینترفیس ها این امکان را به شما می دهند که خارج از ساختار سلسله مراتبی کلاس ها، قابلیت هایی را به برخی از کلاس های خود اضافه کنید. برای مثال فرض کنید شما کلاسی بنام TCar دارید که کلاس هایی مثل TSUV, TTruck, TSedan, TVan و... از آن مشتق شده اند .اگر شما قصد اضافه کردن قابلیت 4WD (تقسیم نیروی محرکه بر روی هر چهار چرخ) به ماشین های SUV و Van داشته باشید، نمی توانید این قابلیت را در کلاس TCar پیاده سازی کنید، چون در این صورت تمامی ماشین ها از آن بهره خواهند برد. البته می توان این قابلیت را در هر یک ار این کلاس ها بصورت مستقل تعریف و پیاده سازی کرد، اما در صورت استفاده از اینترفیس (در اینجا I4wd)، هم می توانید از یک واسط مشترک استفاده کنید و هم از قابلیت های Polymorphic اینترفیس ها بهره ببرید:

کد:

```
I4wd = interface
```

```
['{C2816773-A5DF-4136-AC86-27E6231DB4A9}']
 procedure Initialize4WD;
end;

TCar = class(TInterfacedObject)
public
 procedure SomeCommonBehavior;
 procedure PlayCD; virtual;
end;

TSedan = class(TCar)
public
 procedure SomeSpecificBehavior;
 procedure PlayCD; override;
end;

TSUV = class(TCar, I4wd)
public
 procedure SomeSpecificBehavior;
 procedure PlayCD; override;
 procedure Initialize4WD;
end;

TVan = class(TCar, I4wd)
public
 procedure SomeSpecificBehavior;
 procedure Initialize4WD;
end;

//--------------------------------
procedure Drive4WD(A4WDCar : I4WD);
begin
 if A4WDCar <> nil then
        A4WDCar.Initialize4WD;
        {Add Code for driving the car here}
end;
```

با استفاده از رویه Drive4WD شما می توانید هر ماشینی را که از اینترفیس I4WD شتیبانی می کند، برانید. البته هر ماشین متد Initialize4WD مربوط به خود را فراخوانی می کند. برای مثال :

```
Drive4WD(TVan.Create);
```

نگران آزاد کردن حافظه اشغال شده توسط TVan.Create نباشید، شی ایجاد شده بصورت خودکار آزاد خواهد شد.

Globally Unique Identifier (GUID)

عبارت {C2816773-A5DF-4136-AC86-27E6231DB4A9} در تعریف اینترفیس بالا، یک GUID نامیده می شود. GUID عددی ۱۲۸ بیتی است (در مبنای ۱۶) که با تقریب بالایی منحصر به فرد می باشد [تعداد کل GUID هایی که می توان ساخت ۲ به توان ۱۲۸ است و عملا احتمال اینکه دو GUID یکسان تولید شود بسیار کم است.]
استفاده از GUID در تعریف یک اینترفیس موجب منحصر به فرد شدن آن اینترفیس می شود، به عبارت دیگر، حتی اگر چند اینترفیس در یک سیستم با نام مشابه وجود داشته باشند، مشکلی در استفاده از آنها بوجود نمی آید. استفاده از GUID در یک اینترفیس اجباری نیست، اما وجود آن برای استفاده از عملگر as، متد GetInterface، تابع Supports و متد QueryInterface (بطور کلی **Interface**)

Querying الزامی ست. اشیا؛ COM، اینترفیس ها و Type Library آنها نیز باید هر کدام دارای GUID باشند.
برای ساخت GUID در Editor دلفی می توانید از ترکیب سه کلید Ctrl+Shift+G استفاده کنید تا دلفی برای شما یک GUID ایجاد کند.

نکته: هیچگاه GUID یک اینترفیس را در یک اینترفیس دیگر استفاده نکنید.

استخراج یک Interface از داخل یک Class

فرد متاهلی را در نظر بگرید که ریس یک شرکت می باشد. این فرد در رابطه با کارمندان خود نقش ریس، در رابطه با همسر خود نقش شوهر و در رابطه با فرزندان خود نقش پدر را ایفا می کند. در واقع این فرد در رابطه با هر گروه از افراد مذکور طبق یک رسم یا توافق مشترک، عمل می کند. هر یک از این گروه ها نیز بر اساس همان توافق یا رسم با این فرد ارتباط برقرار می کند و فقط به نقش مربوط به خود نیاز دارد (مثلا یک کارمند نیازی به دانستن رابطه ریس خود با همسر و فرزندانش ندارد). در دنیای نرم افزار هم یک کلاس می تواند با پیاده سازی اینترفیس های مختلف، نقش های مختلفی را در ارتباط با دنیای خارج بر عهده بگیرد
حال، فرد بالا را بعنوان یک کلاس و توافق او با هر یک از گروه های فوق را به عنوان یک اینترفیس در نظر می گیریم:
هر گروه(**درخواست کننده، هر شی در دنیای خارج از آن کلاس**) باید ابتدا چک کند که آیا فرد (**کلاس پیاده سازی کننده**) مذکور به رسومات و توافق مشخص شده(**اینترفیس**) پایبند هست یا نه، اگر آن کلاس از اینترفیس مشخص شده پشتیبانی کند، درخواست کننده می تواند، بر اساس همان اینترفیس، درخواست خود را به آن کلاس ارائه دهد و جواب دریافت کند (**استخراج یک اینترفیس از کلاسی که آن را پیاده سازی می کند**.)
در دلفی برای بررسی پشتیبانی کردن یک کلاس از یک اینترفیس خاص و استخراج آن اینترفیس می توان از تابع Supports در یونیت SysUtils، متد QueryInterface از هر کلاسی که IInterface را پشتیبانی می کند (مثل TInterfaceObject) ، عملگر as، یا متد GetInterface در کلاس TObject استفاده کرد (در واقع تمامی روش های ذکر شده در داخل خود به نوعی از متد GetInterface استفاده می کنند).

به روش های مختلف استخراج اینترفیس I4WD از کلاس TSUV در مثال زیر توجه کنید:
کد:

```
SUV : TSUV;
A4wdInterface : I4WD;
----------------------------------
 //If TSuv implements I4WD (supports it), extract the interface.
 if Supports(TSUV.Create,I4WD,A4wdInterface) then
            //Invoke Initialize4WD from returned interface.
            A4wdInterface.Initialize4WD;
----------------------------------
 //Just check if TSuv supports I4WD, no interface is returned.
 if Supports(TSuv,I4WD) then
            ShowMessage('TSuv supports I4WD interface');
----------------------------------
 SUV := TSUV.Create;
 //If you are sure than SUV supports I4WD, use as operator to extract the interface and
invoke Initialize4WD
//If SUV does not support I4WD, an exception will be thrown.
 (SUV as I4wd).Initialize4WD;
----------------------------------
 SUV := TSUV.Create;
 //If Suv implements I4WD (supports it), extract the interface.
 if SUV.GetInterface(I4WD,A4wdInterface) then
            A4wdInterface.Initialize4WD;
----------------------------------
 //TSUV supports I4WD, so you can assign an instance of it to a variable of type I4WD.
```

```
A4wdInterface := TSuv.Create;
A4wdInterface.Initialize4WD;
```

تمامی توابع و عملگرهای فوق در Help دلفی توضیح داده شده اند.

(Multiple Inheritance) وراثت چندگانه

در زبانهایی مثل دلفی، #C و جاوا وراثت چندگانه امکان پذیر نیست - یعنی یک کلاس نمی تواند مثل C++ از چند کلاس والد مشتق شود. اما در این زبان ها امکان پیاده سازی چند اینترفیس توسط یک کلاس وجود دارد و می تواند به نوعی این خلاء را پر کند. مثلا کلاس TMyMobile (یک کلاس فرضی مربوط به گوشی موبایل) می تواند بطور همزمان اینترفیس های IInfraRed و IBlueTooth را پیاده سازی کند:

کد:

```
TMyMobile = class(TInterfaceObject, IInfraRed, IBlueTooth)
 //TMobile's fields and methods + IInfraRed's methods + IBlueThoot's methods
end;
```

Reference counting (شمارش مرجع)

یکی از ویژگی های مهم اینترفیس ها در دلفی (همچنین در COM) خصوصیت Reference Counting آنها ست. با استفاده از این قابلیت اشیاء ایجاد شده بصورت خودکار مدیریت می شوند و در صورتی که نیازی به آنها نباشد، بطور خودکار آزاد می شوند. متدهای مربوط به این قابلیت در اینترفیس IInterface تعریف شده اند.
هر بار که اینترفیسی از یک کلاس استخراج می شود یا نمونه ایی از یک کلاس به متغیری از یک اینترفیس اختصاص داده می شود، دلفی متد AddRef_ از اینترفیس IInterface را فراخوانی می کند. هر زمان که آن متغیر نا معتبر شود (با اختصاص مقدار nil به آن یا خارج شدن از محدوده تعریف آن متغیر)، دلفی متد Release_ را فراخوانی می کند. شما به عنوان برنامه نویس وظیفه دارید که متغیری در کلاس خود تعریف کرده (برای ذخیره شماره مرجع) و این متد ها را پیاده سازی کنید. با اجرای AddRef_ باید یک واحد به شماره مرجع اضافه شود. با اجرای Release_ باید یک واحد از شماره مرجع کم شود، اگر شماره مرجع به صفر رسید، Relase_ وظیفه آزاد کردن شی ایجاد شده را بر عهده دارد. در صورتی که تمایلی به پیاده سازی متدهای فوق ندارید، می توانید کلاس خود را از کلاس TInterfacedObject مشتق بگیرید، در این صورت کلاس TInterfacedObject قابلیت شمارش مرجع را برای کلاس شما پیاده سازی خواهد کرد. TInterfacedObject در یونیت System تعریف شده است. بهتر است که کد آن را مطالعه نمایید.
به مثالهای زیر در رابطه با کاربرد Reference Counting توجه کنید:

کد:

```
procedure TForm1.Button3Click(Sender: TObject);
var
 SUV : TSUV;
 Van : TVan;
 AnInterface,
 AnotherInterface : I4WD;
begin
 Memo1.Clear;
 SUV := TSuv.Create;
 //SUV.RefCount = 0
 AnInterface := SUV;
 //SUV.RefCount = 1
 AnotherInterface := SUV;
 //SUV.RefCount = 2
```

```
 Van := TVan.Create;
 //Van.RefCount = 0;
 AnInterface := Van;
 //Van.RefCount = 1
end;
//"AnInterface" is out of scope: SUV.RefCount = 1
//"AnotherInterface" is out of scope: SUV.RefCount = 0
//SUV is destroyed
//AnotherInterface is out of scope: Van.RefCount = 0
//Van is destroyed
```

در مثال بالا به تغییرات RefCount توجه کنید:
زمانی که SUV به یک اینترفیس اختصاص داده می شود، یک واحد به شماره مرجع آن افزوده می شود. با اختصاص SUV به اینترفیس بعدی (AnotherInterface)، شماره مرجع آن به ۲ می رسد. با اختصاص یک اینترفیس به Van، شماره مرجع این شی یک واحد افزایش پیدا می کند. در پایان این متد، هیچ کدی برای آزاد سازی متغیرهای SUV و Van وجود ندارد؛ اما با پایان یافتن کد Form1.Button3Click، متغیرهای AnInterface و AnotherInterface که بصورت محلی تعریف شده اند، نامعتبر می شوند و با نامعتبر شدن هر یک از آنها، یک واحد از شماره مرجع هر یک از اشیاء اختصاص یافته به آنها کاهش میابد. با صفر شدن شماره مرجع هر شی، آن شی آزاد می شود.

در شماره ۸۰ **Delphi Magazine** مقاله ای توسط آقای **Malcolm Groves** تحت عنوان **Interfaces: Off The Beaten Track** منتشر شد که در آن سه مورد از کاربردهای Reference Counting توضیح داده شده است. در یکی از این مثالها کلاس ساده ایی (TmtReaper) توضیح داده شده که Garbage Collection را برای شما انجام می دهد. با ارسال یک شی به متد Create این کلاس، شی مورد نظر در زمان مقتضی آزاد خواهد شد و نیازی نیست که شما شخصا آن شی را آزاد کنید:
کد:

```
unit mtReaper;
interface
type
ImtReaper = interface
['{1B321324-975F-4026-B742-E7B3AD486BB5}']
end;
TmtReaper = class(TInterfacedObject, ImtReaper)
private
        FObject : TObject;
public
        constructor Create(AObject : TObject);
        destructor Destroy; override;
end;
implementation
uses SysUtils;
constructor TmtReaper.Create(AObject: TObject);
begin
FObject := AObject;
end;
destructor TmtReaper.Destroy;
begin
FreeAndNil(FObject);
inherited;
end;
end.
-----------------------------------
مثال
var
 List : TStringList;
```

```
  mtReaper : ImtReaper;
begin
  List := TStringList.Create;
  mtReaper := TmtReaper.Create(List);

  //Use List in your code
  List.LoadFromFile('A File Name');
  List.Sort;
end;
//List will be destroyed automatically by mtReaper
```

در مثال دیگری در همین مقاله، روشی برای ذخیره وضعیت یک شی و Restore کردن خودکار آن وضعیت توضیح داده شده است. مثال سوم هم به کاربران CodeSite مربوط می شود.

نویسنده : علي كشاورز

كليه حقوق اين مقاله متعلق به نويسنده و سايت برنامه نويس مي باشد .
استفاده از مطالب اين مقاله در صورت ذكر منبع مجاز است .
www.barnamenevis.org