COM 인터페이스와 COM 객체의 활용

(Using COM Interfaces and COM Objects)

델파이에는 COM 을 지원하기 위한 여러가지 클래스가 존재한다. 이들을 이용하면 쉽게 가장 원시적인 COM 객체에서부터 액티브 폼과 같이 다소 복잡한 객체까지 구현이 가능하다. 이번 장에서는 COM 객체를 생성하고, 인터페이스를 구현하는 기본적인 방법과 이들의 활용방안에 대해서 알아보도록 한다.

인터페이스 활용하기

인터페이스에 대해서는 7 장에서 오브젝트 파스칼의 문법에 대해서 다룬 적이 있지만, COM 기술과 밀접한 관계가 있기 때문에 여기서 다시 한번 알아본다. 인터페이스는 추상 메소드(abstract method)만을 포함한 클래스와 유사한 구조를 가지고 있다. 엄격하게 말해, 인터페이스 메소드 정의에는 파라미터의 수와 데이터 형, 리턴값의 데이터 형과 기대되는 행동에 대해 기술해야 한다. 이러한 인터페이스 메소드는 의미와 논리적으로 인터페이스의 목적과 부합되어야 한다.
인터페이스의 이름은 항상 I 로 시작하는 것이 규칙이다. 예를 들어, IPersist 인터페이스는 객체의 상태를 storage, stream, file 등에 저장하거나 읽어오는 메소드를 정의하는 인터페이스 들의 기초 인터페이스가 된다. 다음의 코드는 실제 인터페이스를 선언한 것으로 편집에 공통적으로 필요한 메소드를 모아서 IEdit 라는 인터페이스를 선언하였다.

```
type
IEdit = interface
  procedure Copy; stdcall;
  procedure Cut; stdcall;
  procedure Paste; stdcall;
  function Undo: Boolean; stdcall;
end;
```

추상 클래스와 마찬가지로, 인터페이스 역시 자체적으로 인스턴스화되지 못한다. 이들을 인스턴스화 하여 사용하려면 인터페이스를 반드시 클래스로 구현해야 한다. 다음의 코드를 살펴 보자.

```
TEditor = class(TInterfacedObject, IEdit)
```

```
  procedure Copy; stdcall;
  procedure Cut; stdcall;
  procedure Paste; stdcall;
  function Undo: Boolean; stdcall;
end;
```

여기서 TEditor 클래스는 TInterfacedObject 라는 기초 클래스를 상속받은 클래스로, TInterfacedObject 는 가장 기본적인 인터페이스라고 할 수 있는 IUnknown 을 구현한 클래스이다. 그러므로, IUnknown 인터페이스의 메소드는 구현할 필요는 없고 이와 같이 추가적으로 구현하겠다고 선언한 IEdit 인터페이스의 메소드 들을 각각 구현하게 된다.

- 클래스간 인터페이스 공유

인터페이스를 이용하면 구현되는 내용과 클래스를 분리해서 디자인할 수 있다. 또한, 2 개의 클래스가 같은 인터페이스를 공유하면 쉽게 다형성(polymorphism)을 구현할 수 있다. 그림을 그리는 기능을 선언한 IPaint 인터페이스와 이 인터페이스를 구현한 2 개의 클래스를 생각해보자.

```
IPaint = interface
  procedure Paint;
end;

TSquare = class(TPolygonObject, IPaint)
  procedure Paint;
end;

TCircle = class(TCustomShape, IPaint)
  procedure Paint;
end;
```

이들은 IPaint 변수에 의해 서로 대입이 가능하며, 서로 다른 클래스의 메소드를 간단하게 하나의 메소드처럼 호출하는 것이 가능하다.

```
var
  Painter: IPaint;
begin
```

```
  Painter := TSquare.Create;
  Painter.Paint;
  Painter := TCircle.Create;
  Painter.Paint;
end;
```

여기에서, IRotate 라는 추가적인 인터페이스를 선언하고 클래스에 구현하도록 수정해보자.

```
IRotate = interface
  procedure Rotate(Degrees: Integer);
end;

TSquare = class(TRectangularObject, IPaint, IRotate)
  procedure Paint;
  procedure Rotate(Degrees: Integer);
end;

TCircle = class(TCustomShape, IPaint)
  procedure Paint;
end;
```

원은 회전해도 모양이 변하지 않으므로, IRotate 인터페이스를 구현할 필요가 없다.

- 프로시저에 인터페이스 활용하기

인터페이스를 이용하면 특정 기초 클래스에서 상속받은 객체를 요구하지 않고도 객체들을 다룰 수 있는 일반적인 프로시저를 작성할 수 있다. 앞에서 IPaint, IRotate 인터페이스를 가지고 다음과 같은 프로시저의 제작이 가능하다. 여기에서 IPaint 인터페이스를 구현한 여러 객체들의 배열을 파라미터로 사용할 수 있다.

```
procedure PaintObjects(Painters: array of IPaint);
var
  I: Integer;
begin
  for I := Low(Painters) to High(Painters) do
    Painters[I].Paint;
```

```
end;

procedure RotateObjects(Degrees: Integer; Rotaters: array of IRotate);
var
  I: Integer;
begin
  for I := Low(Rotaters) to High(Rotaters) do
    Rotaters[I].Rotate(Degrees);
end;
```

- as 연산자의 사용

인터페이스를 구현한 클래스는 as 연산자를 사용하여 인터페이스에 동적으로 바인딩할 수 있다. 다음 코드를 살펴 보자.

```
procedurePaintObjects(P: TInterfacedObject)
var
  X: IPaint;
begin
  X := P as IPaint;
  ...
end;
```

여기에서 TInterfacedObject 클래스인 변수 P 는 변수 X 에 대입될 수 있다. 이때 변수 X 는 IPaint 인터페이스의 레퍼런스가 된다. 이런 대입이 가능한 것이 동적 바인딩 때문이다. 이때 컴파일러는 P 가 IPaint 인터페이스를 지원하는지 알아내기 위해서 P 의 IUnknown 인터페이스의 QueryInterface 메소드를 호출하는 코드를 생성한다. 런타임에서 P 가 IPaint 레퍼런스 대입이 되지 않으면 예외가 발생한다. 그렇지만, 컴파일 시에는 최소한 P 가 IUnknown 만 지원한다면 에러는 발생하지 않는다.

- 포함과 대리, 통합 (Containment, Delegation and Aggregation)

COM 에서 인터페이스를 선언한 뒤에도 이런 인터페이스에 어떤 기능을 더 넣어서 확장을 하고 싶은 경우가 있을 것이다. COM 에서는 이런 경우에 포함과 통합을 이용하게 된다. 다시 말해서 포함과 통합은 어떤 객체가 다른 객체를 어떻게 사용할 것인가를 결정하는 기술이다.

1. 포함과 대리 (Containment and Delegation)

인터페이스를 이용한 코드 재사용 중에서 대표적인 방법 중의 하나가 다른 객체를 포함하거나, 다른 객체에 포함되는 포함(containment)이다. 포함에서 외부 객체가 내부 객체의 인터페이스 포인터를 포함한다. 즉, 외부 객체가 내부 객체의 인터페이스를 이용해서 자신의 인터페이스를 구현하는 것이다. VCL 은 포함과 코드 재사용을 위해서 프로퍼티를 사용한다. 이를 지원하기 위해서 델파이 4 에서 implements 키워드가 추가 되었다. 이를 이용하면 서브-객체(sub-object)에 대한 인터페이스의 구현의 일부 또는 전부를 대리(delegate)에 대한 코드로 쉽게 작성할 수 있다. Implements 키워드를 이용한 대리에 대한 내용은 7 장을 참고하기 바란다.

2. 통합 (Aggregation)

통합(aggregation)은 포함과 대리를 통한 코드 재사용과는 다르다. 즉, 외부 객체가 내부 객체를 포함하는데 이때 내부 객체에 의해 구현된 인터페이스는 외부 객체에 의해서만 노출될 수 있다. 통합은 코드 재사용을 할 때 서브-객체가 포함된 객체의 기능을 정의하지만, 구체적인 구현부는 숨겨지는 형태로 코드 재사용을 하기 때문에, 모듈화가 용이하다는 장점이 있다.

통합에서 외부 객체는 하나 이상의 인터페이스를 구현한다. 내부 객체 역시 하나 이상의 인터페이스를 구현할 수 있다. 그렇지만, 외부 객체만 인터페이스를 노출할 수 있다. 그러므로, 외부 객체에 의해 노출되는 인터페이스 중에는 외부 객체가 구현하는 것도 있고, 내부 객체가 구현하는 것도 존재하게 된다. 그렇지만, 클라이언트는 내부 객체에 대해서는 전혀 알 수가 없다. 그렇기 때문에, 외부 객체 클래스는 내부 객체 클래스 형을 같은 인터페이스를 구현하는 어떤 클래스와도 교체가 가능하다. 즉, 내부 객체 클래스에 대한 코드는 이를 사용하고자 하는 여러 클래스 들에 의해 공유될 수 있는 것이다.

통합에 대한 구현 모델은 대리(delegation)를 이용해서 IUnknown 을 구현하는 규칙을 명시적으로 정의한다. 내부 객체는 반드시 IUnknown 을 구현해야 하며, 내부 객체의 참조 계수(reference count)를 조절해야 한다. 이러한 IUnknown 의 구현은 외부와 내부 객체의 관계를 추적하게 된다. 예를 들어, 내부 객체와 같은 형의 객체가 생성될 때 요구된 인터페이스의 IUnkown 이 성공할 때에만 제대로 객체가 생성된다. 또한, 내부 객체는 구현하는 모든 인터페이스에 대한 2 번째 IUnknown 인터페이스를 구현해야 한다. 이 인터페이스들이 외부 객체에 대한 QueryInterface, AddRef, Release 메소드를 호출하는 대리(delegate) 역할을 하게 된다. 이때 외부에 노출된 IUnknown 을 Controlling Unknown 이라고 한다.

통합 클래스를 작성할 때에는 TComObject 의 IUnknown 인터페이스의 자세한 구현 부분을

참고로 하면 된다. TComObject 는 통합을 지원하는 COM 클래스이기 때문에, COM 어플리케이션을 작성할 때 TComObject 를 기초 클래스로 이용하면 통합을 쉽게 지원할 수 있다.

- 참조 계수에 대한 이해

델파이는 인터페이스 질의과 참조 계수 처리의 대부분을 구현한 클래스를 제공한다. TInterfacedObject 클래스는 이러한 역할을 하는 기초 클래스이다. 참조 계수를 사용하기 위해서는 객체를 인터페이스 레퍼런스로 사용하고, 참조 계수를 적용해야 한다. 다음의 코드를 살펴보자.

```
procedure beep(x: ITest);
function test_func()
var
  y: ITest;
begin
  y := TTest.Create;          //y 가 ITest 변수이므로, 참조 계수는 1 이 된다.
  beep(y);                    //y 를 이용한 호출이므로 참조 계수를 1 증가시킨 후, 다시 감소한다.
  y.something;                //현재 참조 계수는 1 이다.
end;
```

TInterfacedObject 를 이용하면, 이런 참조 계수의 사용이 자동으로 이루어 지게 되지만 다음과 같이 이를 제대로 다루지 않으면 객체가 갑자기 사라질 수도 있으니 주의해야 한다.

```
function test_func()
var
  x: TTest;
begin
  x := TTest.Create;                  //인터페이스를 참조하지 않으므로 계수는 0 이다.
  beep(x as ITest);                   //참조 계수가 1 증가 했다가 1 감소한다.
  x.something;                        //객체가 없어진다 !
end;
```

객체가 VCL 컴포넌트이거나 다른 컴포넌트에 의해 소유된 컨트롤인 경우에는 TComponent 에 기초한 메모리 관리 시스템을 사용하게 된다. 이때 VCL 컴포넌트의 메모리 관리 시스템과 COM 의 참조 계수를 합쳐서 작성하면 안된다. 만약, 인터페이스를 지원

하는 컴포넌트를 만들고 싶다면 IUnknown 의 AddRef, Release 메소드를 COM 참조 계수 기전을 사용하지 않도록 empty 함수로 작성하면 된다.

```
function TMyObject.AddRef: Integer;
begin
  Result := -1;
end;

function TMyObject.Release: Integer;
begin
  Result := -1;
end;
```

QueryInterface 메소드는 객체에 대한 동적 질의를 제공하기 위해 구현해야 한다. QueryInterface 를 구현하면 as 연산자를 이용하여 컴포넌트에서 인터페이스를 사용할 수 있게 된다. 또한, 통합(aggregation)을 이용할 수 있는데 외부 객체가 컴포넌트이면 내부 객체는 참조 계수를 구현해야 한다. 이때 TInterfacedObject 를 기초 클래스로 내부 객체를 구현하고, 컴포넌트를 외부 객체로 사용하여 통합을 구현하는 것이 보통이다.

- 분산 어플리케이션에서의 인터페이스 활용

인터페이스는 COM 이나 CORBA 분산 객체 모델의 핵심이다. 델파이는 이들의 공통적인 기초 클래스로 TInterfacedObject 를 제공한다. TInterfacedObject 는 단순히 IUnknown 인터페이스 메소드를 구현한다.
COM 클래스는 클래스 팩토리와 클래스 ID(CLSIDs)를 이용한 기능성이 추가된다. 클래스 팩토리는 CLSIDs 를 통해 클래스 인스턴스를 생성하게 되고, CLSIDs 는 COM 클래스를 등록하고 관리하는 기초 요소가 된다. 이때 클래스 팩토리와 클래스 ID 를 가진 COM 클래스를 CoClass 라고 한다.
참고로, CORBA 의 경우 클라이언트의 스텁(stub) 클래스와 서버의 스켈레톤(skeleton) 클래스를 통해 인터페이스를 사용하게 된다. 어플리케이션은 반드시 스텁과 스켈레톤 클래스를 사용하거나, 모든 파라미터를 가변형으로 변경하는 DII(Dynamic Invocation Interface)를 이용할 수 있다. 델파이에서는 CORBA 를 클래스 팩토리를 통해서 구현할 수 있도록 하였으며, CoClass 도 사용한다. 이렇게 COM 과 CORBA 를 구현하는 방법을 동일하게 사용함으로써 델파이는 COM/CORBA 클라이언트를 동시에 지원하는 서버를 작성할 수 있게 되었다.

델파이 COM 관련 클래스의 이해

델파이의 COM 관련 클래스에 대해서는 25 장에서 간단하게 다루었지만, 여기서는 가장 기초적인 클래스라고 할 수 있는 TInterfacedObject, TComObject, TTypedComObject 에 대해서 알아보도록 한다.

- TInterfacedObject

TInterfacedObject 클래스는 IUnknown 인터페이스를 구현한 기초 클래스이다. 이 클래스는 System.pas 유닛에 다음과 같이 선언되어 있다.

```
type
  TInterfacedObject = class(TObject, IUnknown)
  private
    FRefCount: Integer;
 protected
    function QueryInterface(const IID: TGUID; out Obj): Integer; stdcall;
    function _AddRef: Integer; stdcall;
    function _Release: Integer; stdcall;
  public
    property RefCount: Integer read FRefCount;
  end;
```

사용법은 비교적 간단하다. 그대로 상속받아서 사용하는 것인다. 다음의 코드에서 TDerived 는 IPaint 인터페이스를 구현한 클래스이다.

```
type
  TDerived = class(TInterfacedObject, IPaint)
    ...
  end;
```

IUnknown 을 구현하고 있기 때문에, 참조 계수 관리와 인터페이스 객체의 메모리 관리를 자동으로 해 주는 역할을 한다.

- TComObject

TComObject 는 IUnknown 이외에 COM 객체를 위한 추가적인 인터페이스를 지원한다. TComObject 는 반드시 클래스 ID 를 가져야 한다. CLSIDs 는 데이터베이스 레지스트리에 클래스를 등록하고, 클래스를 클래스 팩토리를 이용하여 외부에 인스턴스화할 때 필요하다. TComObject 에 대한 클래스 팩토리는 TComObjectFactory 이며, 클래스 팩토리 프로퍼티는 이름, 설명, CLSID 등의 TComObject 클래스에 대한 정보를 제공하며, 클래스 팩토리의 메소드를 이용하여 TComObject 클래스를 레지스트리에 등록하고, 이를 인스턴스화 한다.
TComObject 는 단일 COM 객체로 인스턴스화 될 수도 있고, 통합(aggregation)의 한 부분으로서 인스턴스화할 수도 있다. TComObject 의 IUnknown 메소드 들은 통합의 내부 객체로 COM 객체가 인스턴스화할 때 IUnknown 인터페이스를 조절할 수 있도록 구현되어 있다. 결과적으로 내부 COM 객체에 의해 구현된 모든 인터페이스는 내부적인 참조 계수를 가지게 되지만, 이것이 인터페이스 레퍼런스가 생성될 때 직접 영향을 받지는 않는다.
ISupportErrorInfo 인터페이스는 OLE 자동화 컨트롤러가 에러 객체를 계속 사용할 수 있는지 여부를 질의하고 에러 정보를 호출 chain 에 파급할 수 있도록 한다. 또한, IErrorInfo 를 지원하여 OLE 예외 처리와 safecall 호출 규칙을 지원한다.

- TTypedComObject

TTypedComObject 클래스는 TComObject 클래스에 IProvideClassInfo 인터페이스를 추가하여 객체에 대한 인터페이스와 데이터 형을 설명할 수 있도록 한다. 이를 위해서 타입 라이브러리를 요구하는데, 디폴트로는 듀얼 인터페이스를 지원한다.
IProvideClassInfo 는 GetClassInfo 라는 메소드를 지원하여 타입 정보를 제공할 수 있다. 즉, TTypedComObject 클래스에서 상속받은 객체가 실행될 때 클라이언트가 객체에 대한 정보를 GetClassInfo 메소드를 이용하여 얻을 수 있다.
TTypedComObject 클래스는 타입 라이브러리를 로드하지 않고, 타입 정보를 제공하고자 할 때 기초 클래스로 사용하면 된다.

TInterfacedObject 클래스를 이용한 예제의 작성

그럼 앞에서의 설명을 바탕으로 2 개의 인터페이스를 지원하는 간단한 객체를 하나 제작해 보자. 먼저 ILocation, IDimension 이라는 인터페이스를 type 선언문에 다음과 같이 선언한다. 이 인터페이스 들은 그림을 그리는 객체의 위치와 크기를 각각 지정하는 역할을 한다.

```
ILocation = interface
  function GetLeft: Integer;
  procedure SetLeft(Value: Integer);
  function GetTop: Integer;
```

```
    procedure SetTop( Value: Integer );
    property Left: Integer read GetLeft write SetLeft;
    property Top: Integer read GetTop write SetTop;
  end;

  IDimension = interface
    function GetWidth: Integer;
    procedure SetWidth(Value: Integer);
    function GetHeight: Integer;
    procedure SetHeight(Value: Integer);
    property Width: Integer read GetWidth write SetWidth;
    property Height: Integer read GetHeight write SetHeight;
  end;
```

그리고, 이 인터페이스를 구현할 객체를 TInterfacedObject 클래스에서 상속받아 다음과 같이 선언한다.

```
  TWidget = class(TInterfacedObject, ILocation, IDimension)
  private
    FLeft: Integer;
    FTop: Integer;
    FWidth: Integer;
    FHeight: Integer;
  public
    constructor Create;
    function GetLeft: Integer;
    procedure SetLeft(Value: Integer);
    function GetTop: Integer;
    procedure SetTop(Value: Integer);
    function GetWidth: Integer;
    procedure SetWidth(Value: Integer);
    function GetHeight: Integer;
    procedure SetHeight(Value: Integer);
    procedure Draw(Canvas: TCanvas);
  end;
```

이렇게 선언된 TWidget 컴포넌트에서 ILocation, IDimension 과 공통적인 메소드는 이들 인터페이스를 실제로 구현할 때 사용된다. Draw 메소드는 이들 인터페이스의 메소드가 아니라 TWidget 의 메소드로 실제 도형을 그리는 역할을 한다.
그러면, TWidget 클래스를 다음과 같이 구현하도록 한다.

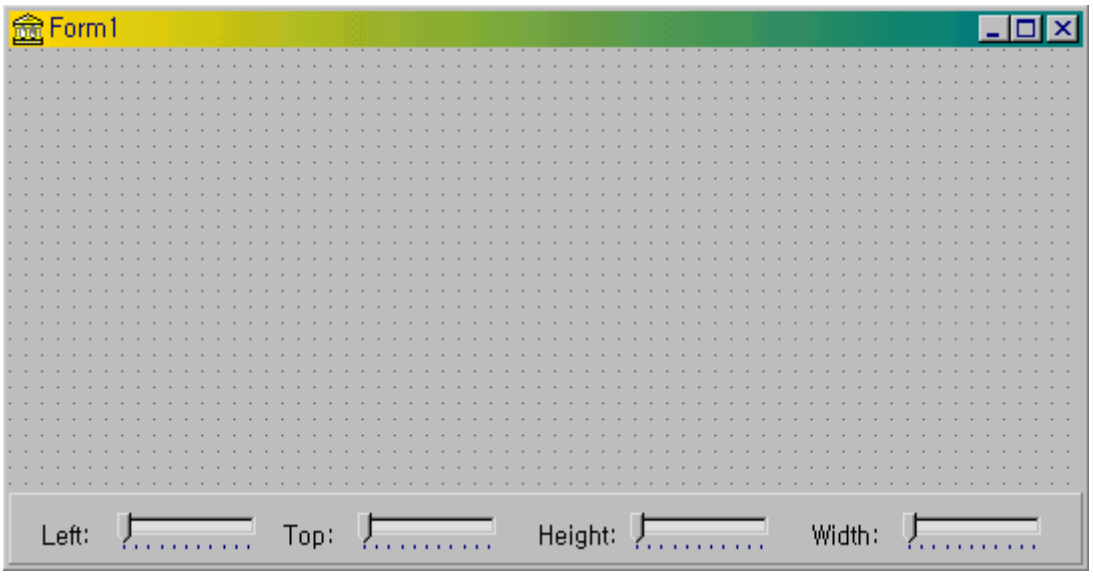
```
constructor TWidget.Create;
begin
  inherited Create;
  FLeft := 0;
  FTop := 0;
  FWidth := 100;
  FHeight := 100;
end;

function TWidget.GetLeft: Integer;
begin
  Result := FLeft;
end;

procedure TWidget.SetLeft(Value: Integer);
begin
  FLeft := Value;
end;

function TWidget.GetTop: Integer;
begin
  Result := FTop;
end;

procedure TWidget.SetTop(Value: Integer);
begin
  FTop := Value;
end;

function TWidget.GetWidth: Integer;
begin
```

```
  Result := FWidth;
end;

procedure TWidget.SetWidth(Value: Integer);
begin
  FWidth := Value;
end;

function TWidget.GetHeight: Integer;
begin
  Result := FHeight;
end;

... (중략)

procedure TWidget.Draw(Canvas: TCanvas);
begin
  with Canvas do
  begin
    Brush.Color := clBlue;
    Pen.Color := clBlack;
    Rectangle( FLeft, FTop, FLeft + FWidth, FTop + FHeight );
  end;
end;
```

그다지 어려운 내용이 아니므로, 설명은 생략하겠다. 이제 이들 인터페이스와 클래스 객체를 사용하려면 이들을 담을 변수를 선언해야 한다. 폼의 public 섹션에 다음과 같이 변수를 선언하도록 한다.

```
  AWidget: TWidget;
  LocRef: ILocation;
  DimRef: IDimension;
```

이제 준비는 모두 끝났다. 폼에 패널을 하나 얹고 alBottom 으로 Align 프로퍼티를 설정하고, Caption 을 없앤 되, 그 위에 TLabel 과 TTrackBar 컴포넌트를 각각 4 개씩 얹고 다음과 같이 디자인하도록 한다.



이들 트랙바는 각각 폭을 80, 높이는 20, LineSize 프로퍼티는 5 정도로 설정한다. 이들을 조정하면 파란색 바탕, 검은색 라인의 사각형이 폼에 그려지도록 TWidget 클래스와 2 개의 인터페이스를 활용할 것이다. Form 의 OnCreate 이벤트 핸들러를 다음과 같이 작성한다.

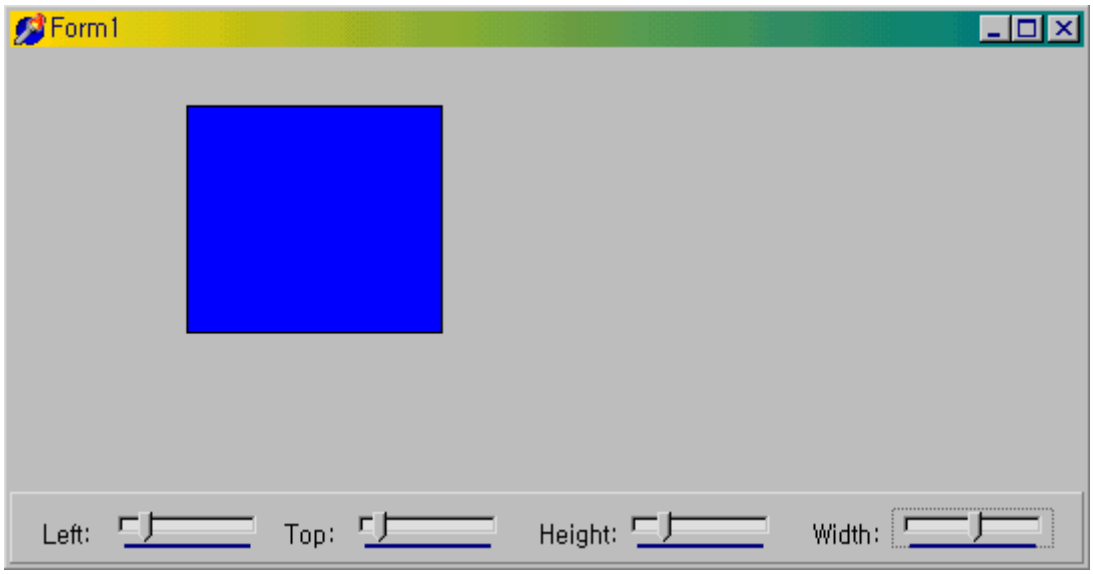
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  AWidget := TWidget.Create;
  LocRef := AWidget;
  DimRef := AWidget;
  TrackBar1.Max := Width;
  TrackBar2.Max := Height - Panel1.Height;
  TrackBar3.Max := Width - TrackBar1.Position;
  TrackBar4.Max := Height - TrackBar2.Position - Panel1.Height;
  TrackBar1.Position := 0;
  TrackBar2.Position := 0;
  TrackBar3.Position := 100;
  TrackBar4.Position := 100;
end;
```

여기서, TWidget 클래스를 생성하여 AWidget 변수에 대입하고 이 객체를 ILocation, IDimension 변수인 LocRef, DimRef 에 대입할 수 있다. 이는 AWidget 객체가 ILocation, IDimension 인터페이스를 구현하고 있기 때문에 가능하다. 그 뒤의 코드는 4 개의 트랙 바

에 대한 초기값을 폼의 크기에 맞추어 설정하는 내용이다.
이제 OnPaint 이벤트 핸들러에서 작성하고, 4 개의 트랙 바를 선택하고 이들 모두의 OnChange 이벤트 핸들러를 작성한다.

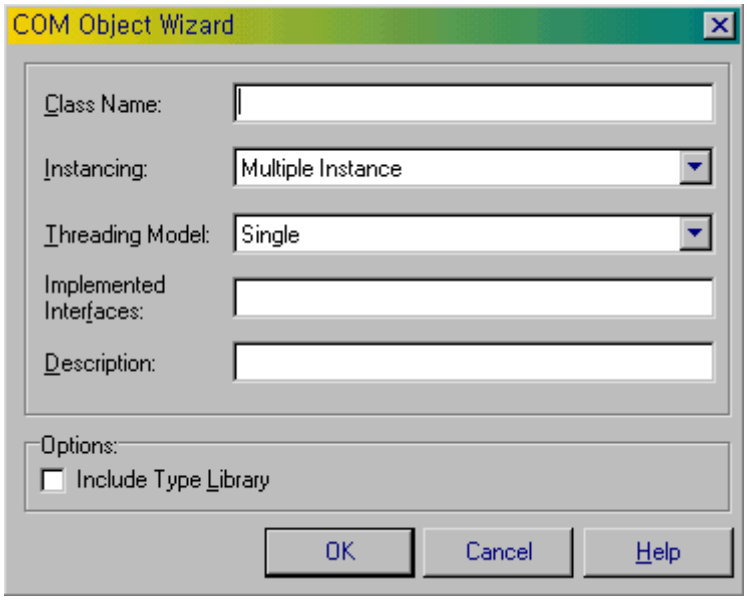
```
procedure TForm1.FormPaint(Sender: TObject);
begin
  AWidget.Draw(Canvas);
end;

procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  LocRef.Left := TrackBar1.Position;
  LocRef.Top := TrackBar2.Position;
  DimRef.Height := TrackBar3.Position;
  DimRef.Width := TrackBar4.Position;
  Invalidate;
end;
```

이 어플리케이션을 실행하면 다음과 같이 크기와 위치를 트랙 바를 가지고 마음대로 조절할 수 있는 파란 사각형을 볼 수 있을 것이다.



COM 객체 위저드의 사용

델파이는 여러가지 COM 객체를 생성하는데 도움을 주는 위저드를 많이 제공한다. 이 중에서 COM 객체 위저드를 이용하면 간단하고, 작은 COM 객체를 쉽게 만들 수 있다.
COM 객체를 생성하기 위해서는 먼저 File|New 메뉴를 선택하여 객체 저장소를 띄운뒤, ActiveX 탭을 선택하고 여기에서 COM Object 아이콘을 더블 클릭하면 다음과 같은 대화상자가 나타날 것이다.



COM 객체는 in-process, out-of-process 또는 원격 서버로 구현할 수 있다. COM 객체 위저드는 다음과 같은 작업을 해 준다.

1. 새로운 유닛을 생성한다.
2. TComObject 에서 상속받은 새로운 클래스를 정의하고, 클래스 팩토리 constructor 를 설정한다.

앞의 대화 상자에서 클래스 이름과 구현할 인터페이스를 적어주면 되지만, 그 밖에도 인스턴싱 타입과 쓰레딩 모델을 선택하여야 한다. 이들의 의미는 다음과 같다.

- COM 객체 인스턴싱 타입 (instancing type)

COM 객체가 in-process 서버로만 사용된다면 인스턴싱은 무시된다. 다른 경우에 COM 어플리케이션이 새로운 COM 객체를 생성하게 되면, 다음의 인스턴싱 타입 중 하나를 가질 수 있다.

인스턴싱	의 미

Multiple	여러 개의 어플리케이션들이 객체에 접속할 수 있다. 클라이언트가 서비스를 요구하면, 독립된 서버의 인스턴스가 생성된다.
Single	어플리케이션이 객체에 접속하면, 다른 어플리케이션은 접속할 수 없다. 이 옵션은 SDI 어플리케이션에서 자주 사용되는데, 클라이언트가 단일 인스턴스 객체에서 서비스를 요구하면, 모든 요구가 같은 서버에 의해 다루어진다. 예를 들어, 사용자가 워드 프로세서에서 새로운 문서를 열면, 이 문서는 어플리케이션 프로세스와 같은 프로세스에서 열린다.
Internal	이 옵션으로 지정된 객체는 내부적으로 생성된다. 외부의 어플리케이션은 객체의 인스턴스를 직접 생성할 수 없다.

- 쓰레딩 모델의 선택

위저드에서 객체를 생성할 때에 객체가 지원하게될 쓰레딩 모델을 지정해야 한다. 어플리케이션의 클라이언트와 서버 양측에서 COM 을 초기화할 때 쓰레드를 이용하는 규칙을 지정하는데, COM 은 이를 비교하여 같은 규칙을 지원하면 COM 은 양측을 직접 연결하고 규칙을 지킬 것이라고 신뢰할 수 있다. 만약 서로의 규칙이 다르면 COM 은 마샬링을 설정하여 서로가 자신의 규칙을 지킬 수 있도록 해준다. 물론 마샬링은 수행 속도를 감소시키는 것이 단점이다.
위저드에서 선택하게 되는 쓰레딩 모델은 레지스트리에 기록되는 정보이다. 그러므로, 개발자는 쓰레딩 모델에 걸맞게 객체를 구현하여야 한다. 쓰레딩 모델은 in-process 서버에서만 유효하다. Out-of-process 서버는 EXE 로 등록되며, 쓰레딩 모델은 자신이 직접 구현해야 한다.
선택할 수 있는 쓰레딩 모델로는 다음과 같은 것들이 있다.

쓰레딩 모델	설 명
Single	쓰레드를 지원하지 않는다. 클라이언트의 요구는 호출 기전에 의해 나열되고, 클라이언트는 한번에 하나씩 처리하므로 쓰레드 지원이 필요하지 않다. 대신 수행 성능의 향상은 없다.
Apartment (Single-thread apartment)	클라이언트가 객체의 메소드를 객체가 생성된 쓰레드에서만 호출할 수 있는 모델이다. 같은 서버의 서로 다른 객체 들은 서로 다른 쓰레드에서 호출될 수 있으나, 각각의 객체 자체는 하나의 쓰레드에서만 호출될 수 있다. 각 인스턴스의 데이터는 안전하고, 전역 데이터는 반드시 임계 섹션이나 다른 동기화 기법을 이용하여 보호해야 한다. 약간의 수행 성능 향상이 있으며, 객체도 작성하기 쉬운 장점이 있다. 웹 브라우저 컨트롤로 가장 많이 사용되는 모델이다.
Free (Multi-thread apartment)	클라이언트가 어느 때, 어느 쓰레드에서든 객체의 메소드를 호출할 수 있는 모델이다. 객체는 반드시 모든 인스턴스와 전역 데이터를 임계 섹션이나 다른 동기화 기법을 이용하여 보호해야 한다. 클라이언트는 쉽게 작성할 수 있지만, 서버

	객체는 구현하기 어렵다. 분산 DCOM 환경에서 많이 사용되는 모델이다.
Both	객체가 클라이언트를 지원할 때 apartment 와 free 쓰레딩 모델을 모두 지원하는 모델이다. 최고의 수행 능력과 유연함을 가지게 된다.

참고로 지역 변수는 쓰레딩 모델에 관계 없이 언제나 안전하다. 이는 지역 변수가 각 쓰레드의 스택에 변수의 값을 저장하기 때문이다.

1. Free 쓰레딩 모델을 지원하는 객체의 제작

거의 언제나 하나 이상의 쓰레드에서 객체에 접근해야 하는 경우에는 Free 쓰레딩 모델을 사용하는 것이 좋다. 가장 흔한 예로는 원격 기계에 있는 객체에 접속하는 클라이언트 어플리케이션이다. 원격 클라이언트가 객체의 메소드를 호출하면, 서버는 서버 기계의 쓰레드 pool 에서 쓰레드의 호출을 받게 된다. 이 쓰레드는 실제 객체에 로컬로 호출하지만, 객체가 free 쓰레딩 모델을 지원하기 때문에 쓰레드는 객체를 직접 호출할 수 있다.
만약 객체가 apartment 쓰레딩 모델을 지원하면 호출은 객체를 생성한 쓰레드로 이전되어야 하며, 결과가 클라이언트로 반환되기 전에 호출을 받은 쓰레드로 전송해야 한다. 이런 작업을 위해서는 마샬링이 필요하다.
Free 쓰레딩을 지원하려면, 개발자는 반드시 각각의 메소드에 의해 인스턴스의 데이터가 어떻게 접근되는지 고려해야 한다. 만약 메소드가 인스턴스 데이터를 기록한다면, 개발자는 임계 섹션(critical section) 등의 여러가지 동기화 기법을 이용하여 인스턴스 데이터를 보호해야 한다. 그래도, 이런 데이터 보호 기법이 COM 의 마샬링 보다는 훨씬 효율적이다. 만약 데이터가 읽기 전용이면 이런 문제는 생기지 않는다.

2. Apartment 쓰레딩 모델을 지원하는 객체의 제작

Apartment 쓰레딩 모델을 구현하려면 다음의 몇가지 규칙을 따라야 한다.

- 어플리케이션의 첫번째 쓰레드는 COM 의 메인 쓰레드가 된다. 또한, 이 쓰레드는 COM 을 uninitialize 하는 마지막 쓰레드가 된다.
- Apartment 쓰레딩 모델의 각각의 쓰레드는 반드시 메시지 루프를 가져야 하며, 메시지 큐를 자주 검사해야 한다.
- 쓰레드가 COM 인터페이스에 대한 포인터를 얻으면, 인터페이스의 메소드는 그 쓰레드에서만 호출할 수 있다.

이 쓰레딩 모델은 쓰레딩을 지원하지 않는 모델과 Free 쓰레딩 모델의 중간 정도의 모델이므로, 서버는 전역 데이터는 보호해야 하지만 객체의 인스턴스 데이터는 언제나 같은 쓰레

드의 메소드에 의해 호출되므로 안전하다.
보통 웹 브라우저의 컨트롤에 이 모델이 많이 사용되는데, 이는 브라우저 어플리케이션이 각 쓰레드를 apartment 로 초기화하기 때문이다.

COM 객체의 활용

그러면 이번에는 COM 객체 위저드를 이용해서 COM 객체를 생성해서, 이를 등록하고 클라이언트 어플리케이션에서 COM 객체를 사용하는 방법에 대해서 알아보자.

- COM 객체 서버의 제작

먼저 COM 객체를 구현할 프로젝트 파일을 만들어야 한다. 여기서는 DLL 의 형태로 만들 것이다. 일단 File|New 메뉴를 선택하고 여기서 DLL 아이콘을 더블 클릭해서 DLL 프로젝트 파일을 하나 생성한다. 물론, 여기에서 ActiveX 탭에 있는 ActiveX Library 를 더블 클릭하면 더욱 쉽게 구현이 가능하지만 프로젝트 파일을 일반적인 DLL 로 선택한 것은 기초적인 구현 방법을 설명하기 위한 것임을 미리 밝혀 둔다. 그리고, 다시 File|New 메뉴를 선택한 후 객체 저장소에서 ActiveX 탭을 클릭하고 Com Object 아이콘을 더블 클릭하여 앞에서 설명한 구현할 TComObject 의 이름과, 지원할 인터페이스, 인스턴싱 type, 쓰레딩 모델 등을 대화 상자에 적어 넣는다. 인스턴싱 type 과 쓰레딩 모델은 디폴트 값인 Single, Multiple Instance 로 선택한다. 참고로 이들의 종류를 결정할 때에는 앞에서 설명한 대로 COM 객체의 용도에 따라서 결정해야 하는 것임을 명심하도록 하자. 참고로 Description 부분에 기록하는 내용은 나중에 액티브 X 서버에 대한 정보를 레지스트리에 기록하고, 이를 조회할 때 나타나는 설명이 되므로 적당한 설명을 적어넣는 것도 좋을 것이다. 여기서는 COM 객체의 이름으로는 Math, 구현할 인터페이스로는 IMath 라고 적어넣고 OK 버튼을 클릭하면 다음과 같은 뼈대 코드가 생성될 것이다.

```
unit Unit2;

interface

uses
  Windows, ActiveX, ComObj;

type
  TMath = class(TComObject, IMath)
  protected
```

```
    {Declare IMath methods here}
  end;

const
  Class_Math: TGUID = '{ED309D88-2732-11D2-9774-0000E838052E}';

implementation

uses ComServ;

initialization
  TComObjectFactory.Create(ComServer, TMath, Class_Math,
    'Math', '', ciMultiInstance, tmSingle);
end.
```

이렇게 COM 객체 위저드는 해당되는 클래스 이름과 인터페이스 이름에 대한 기본적인 선언과 클래스 팩토리의 생성 코드를 자동으로 만들어 준다. initialization 섹션의 클래스 팩토리 생성 코드가 COM 객체를 생성하는 핵심이 된다.
그러면, 실제로 사용할 만한 COM 객체를 생성해보자. IMath 인터페이스는 델파이의 Math.pas 유닛에서 지원되지 않는 몇가지 중요한 수학 함수를 선언하고, 이를 델파이 외에 다른 언어에서도 사용할 수도 있게할 것이다. 앞에서 설명한 TInterfaced 객체에서의 인터페이스와는 달리 COM 객체에서 사용하는 인터페이스에는 반드시 GUID 형식의 IID 를 가져야 한다. 그러므로, 먼저 interface 이름을 적어 넣고 Shift+Ctrl+G 키를 누르면 GUID 형식의 IID 가 추가된다. 그리고, 인터페이스의 메소드를 다음과 같이 선언한다.

```
  IMath = interface
    ['{ED309D86-2732-11D2-9774-0000E838052E}']
    function Distance (const X1, Y1, X2, Y2: Extended): Extended; stdcall;
    procedure Polar2XY (const Rho, Theta: Extended; var X, Y: Extended); stdcall;
    procedure XY2Polar (const X, Y: Extended; var Rho, Theta: Extended); stdcall;
    function FactorialX (A: Cardinal): Extended; stdcall;
  end;
```

간단히 메소드를 설명하면 Distance 메소드는 두개의 좌표의 거리를 계산하는 함수이며, Polar2XY 와 XY2Polar 메소드는 극좌표계와 XY 좌표계의 좌표를 전환하여 계산하는 함수이다. FactorialX 는 잘 아는 Factorial 함수이다 (n! = 1*2*...*(n-1)*n).

이 인터페이스를 구현하기 위해 TMath 클래스를 다음과 같이 변경하고 이들을 구현한다.

```
  TMath = class(TComObject, IMath)
  private
    function Sgn (const X: Extended): ShortInt;
  public
    function Distance (const X1, Y1, X2, Y2: Extended): Extended; stdcall;
    procedure Polar2XY (const Rho, Theta: Extended; var X, Y: Extended); stdcall;
    procedure XY2Polar (const X, Y: Extended; var Rho, Theta: Extended); stdcall;
    function FactorialX (A: Cardinal): Extended; stdcall;
  end;
```

여기서 Sgn 메소드는 극좌표 계산을 할 때 사용되는 유틸리티 함수이다.
이들을 다음과 같이 구현한다. 구현 방법 자체는 수학적인 내용이므로 설명을 생략하겠다.

```
function TMath.Sgn(const X: Extended): ShortInt;
begin
  if X < 0.0 then Result := -1
  else if X = 0.0 then Result := 0
  else Result := 1
end;

function TMath.Distance(const X1, Y1, X2, Y2: Extended): Extended;
var
  X, Y: Extended;
begin
  X := Abs (X1 - X2);
  Y := Abs (Y1 - Y2);
  if X > Y then       Result := X * Sqrt (1 + Sqr (Y / X))
  else if Y <> 0 then Result := Y * Sqrt (1 + Sqr (X / Y))
  else Result := 0
end;

function TMath.FactorialX(A: Cardinal): Extended;
var
  I: Integer;
```

```
begin
  if A   > 1547 then
  begin
    Result := 0.0;
    Exit;
  end;
  Result := 1.0;
  for I := 2 to A do Result := Result * I;
end;

procedure TMath.Polar2XY(const Rho, Theta: Extended; var X, Y: Extended);
begin
  SinCos (Theta, X, Y);
  X := Rho * X;
  Y := Rho * Y;
end;

procedure TMath.XY2Polar(const X, Y: Extended; var Rho, Theta: Extended);
begin
  Rho := Sqrt (Sqr (X) + Sqr (Y));
  if Abs (X) > 0.00001 then Theta := ArcTan (Y / X)
  else Theta := Sgn (Y) * Pi / 2.0
end;
```

기본적인 구현은 거의 끝난 셈이다. 이제 File|Save All 메뉴를 선택하여 유닛 파일과 프로젝트 파일을 각각 적당한 이름으로 저장하도록 한다. 이 책의 부록으로 제공되는 CD-ROM 에서는 U_ExamSvr1.pas, ExamSvr1.dpr 로 저장하였다. 마지막으로, 프로젝트 파일을 다음과 같이 수정한다.

```
library ExamSvr1;

uses
  ComServ,
  U_ExamSvr1 in 'U_ExamSvr1.pas';

exports
```

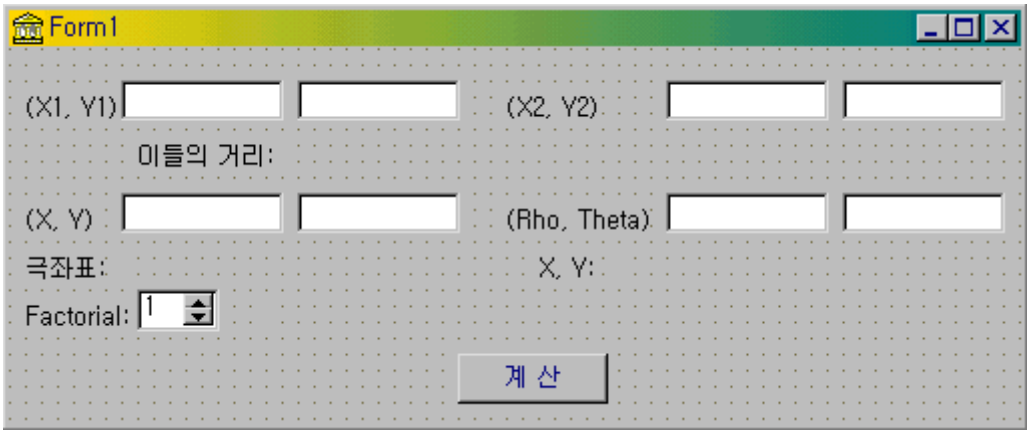
```
  DllGetClassObject, DllCanUnloadNow,
  DllRegisterServer, DllUnRegisterServer;
end.
```

이 코드의 의미는 기본적인 구현 부분은 U_ExamSvr1 에 위치하고 있으며, 프로젝트 파일에서는 COM 서버를 등록할 때 필요한 4 개의 함수를 export 하게 된다. 이들은 모두 ComServ.pas 유닛에 선언되어 있으므로 uses 절에 ComServ.pas 유닛을 추가한 것이다.
DllGetClassObject 는 액티브 X 객체에 대한 클래스 팩토리를 얻으려고 할 때, OLE 엔진이 액티브 X 서버 DLL 을 메모리에 불러들인 후 호출하는 함수이다 .
DllRegisterServer 는 in-process 액티브 X 서버(DLL)에 의해 export 되는 함수로 타입 라이브러리와 서버 모듈에서 지원하는 모든 클래스에 대한 레지스트리 엔트리를 생성한다. DLLRegisterServer 는 일반적으로 액티브 X 서버가 사용자의 시스템에 인스톨될 때 호출된다. DllUnregisterServer 은 in-process 자동화 서버(DLL)에 의해 export 되며, 서버가 언이스톨될 때 DLL 이 DllRegisterServer 에 의해 등록한 엔트리를 제거하는 역할을 한다.
DllCanUnloadNow 는 액티브 X 서버(DLL)에 의해 export 되며, Ole 엔진에 의해 호출되어 서버가 더 이상 사용되지 않을 때 메모리에서 언로드할 수 있는지 알아낸다.
이들 4 개의 함수는 액티브 X/COM 서버가 설치될 때에는 기본적으로 export 되어야 하는 것들이다. 이런 작업을 프로젝트 파일을 선택할 때 표준 DLL 대신 ActiveX Library 를 선택하면 하지 않아도 된다.
이제 몇 가지 수학 함수를 지원하는 COM 객체가 완성되었다. 컴파일을 하고, COM 서버를 등록하면 다른 어플리케이션에서 이를 사용할 수 있게 된다.
등록하는 방법은 델파이를 사용하고 있다면 Run|Register ActiveX Server 명령을 선택하면 즉시 등록된다. 델파이가 깔려 있지 않은 컴퓨터에 ExamSvr1.DLL 파일을 액티브 X 서버로 등록하려면, 마이크로소프트의 REGSVR32.EXE 파일을 이용하여 직접 등록하거나 확장자가 REG 인 파일을 텍스트 파일로 작성하여 이를 실행하도록 하면 자동으로 등록되게 할 수 있다. REG 파일을 작성하는 방법과 REGSVR32.EXE 파일의 사용 방법에 대한 것은 MSDN 이나 마이크로소프트의 웹 사이트를 찾아보면 자세한 내용이 있으므로 이를 참고하기 바란다. 그 밖에, 런타임에서 코드로 작성한 액티브 X 서버 DLL 을 등록할 수 있다. 이 방법은 클라이언트 어플리케이션을 작성하는 부분에서 다루게 될 것이므로 이를 참고하기 바란다.

- COM 객체 클라이언트의 제작

이제 등록된 COM 객체를 이용하여 이를 활용하는 클라이언트 어플리케이션을 만들어 보자. 일단 새로운 어플리케이션을 시작하고 폼을 다음과 같이 디자인한다. 우리가 사용할 IMath 인터페이스의 Distance, Polar2XY, XY2Polar, FactorialX 등의 계산을 마음대로 할 수 있도

록 8 개의 TEdit 컴포넌트와 1 개의 TSpinEdit 컴포넌트를 배치하고, 계산을 실행할 때 사용할 버튼도 하나 올려 넣도록 한다. 그리고, 각 컴포넌트의 역할을 설명하기 위해 여러 개의 라벨을 올려 놓고, 결과를 보여주기 위해 라벨 컴포넌트를 위치시킨뒤 Caption 을 지우도록 한다.



폼의 디자인이 끝났으면, IMath 인터페이스를 사용할 수 있도록 하기 위해 인터페이스 선언부와 클래스 ID 부분을 COM 객체 서버의 소스에서 복사해서 type 선언부에 추가하도록 한다. 그리고, IMath 인터페이스를 저장할 전역 변수를 하나 선언한다.

```
type
  IMath = interface
    ['{ED309D86-2732-11D2-9774-0000E838052E}']
    ...
  end;

var
  Form1: TForm1;
  AMath: IMath;

const
  Class_Math: TGUID = '{ED309D85-2732-11D2-9774-0000E838052E}';
```

uses 절에 ComObj.pas 유닛만 추가하면, COM 객체를 사용할 준비는 모두 끝난 셈이다. 먼저 폼의 OnCreate 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
  AMath := CreateComObject(Class_Math) as IMath;
end;
```

CreateComObject 의 파라미터로는 COM 객체의 CLSID 가 사용된다. 그렇기 때문에, const 절의 내용을 복사해와서 사용한 것이다. 또한 생성된 COM 객체의 인터페이스를 사용하기 위해서 as 연산자를 이용하여 IMath 인터페이스를 AMath 변수에 저장한다.
그런데, 앞에서도 간단히 설명한 바 있지만 액티브 X 서버 DLL 이 등록되어 있지 않다면 여기에서 클래스가 등록되지 않았다는 에러 메시지를 보게 될 것이다. 물론, 액티브 X 서버 DLL 을 컴파일하고 델파이에서 Run|Register ActiveX Server DLL 명령을 선택해서 레지스트리에 등록했거나, REGSVR.EXE 프로그램을 사용하여 등록했다면 문제가 없겠지만 어플리케이션을 배포할 때 이를 모두 처리해주기는 쉽지 않다. 물론 InstallShield 와 같은 설치 프로그램의 최근 버전에는 이들을 처리할 수 있는 내용이 있지만, 이보다 더 간단하게 처리할 수 있는 방법이 있으면 좋을 것이다.
이를 위해서 런타임에서 직접 등록해서 사용할 수 있도록 코드를 수정하도록 하자. 먼저 uses 절에 ActiveX.pas 유닛을 추가하고, 다음과 같이 OnCreate 이벤트 핸들러를 다시 작성한다.

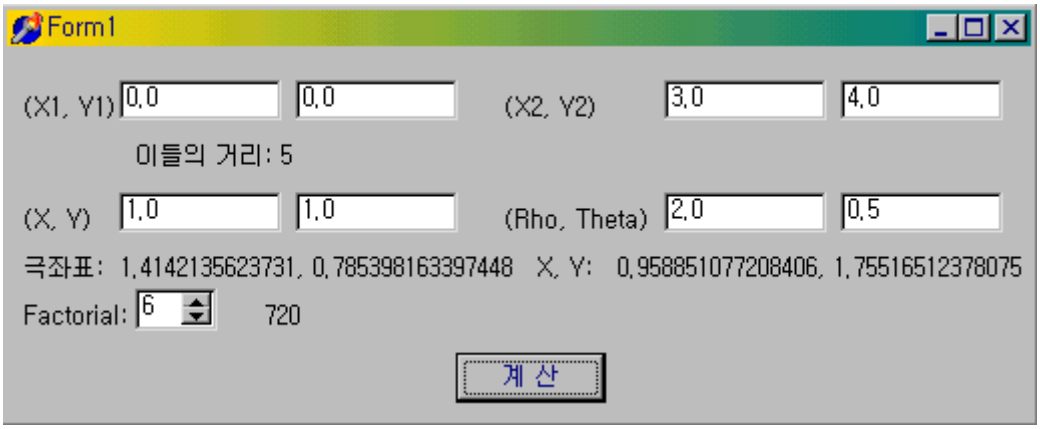
```
procedure TForm1.FormCreate(Sender: TObject);
var
  OCXHandle: THandle;
  RegFunc: TDllRegisterServer;
begin
  try
    AMath := CreateComObject(Class_Math) as IMath;
  except
    OCXHandle
      := LoadLibrary(PChar(ExtractFilePath(Application.ExeName) + '₩ExamSvr1.Dll'));
    RegFunc := GetProcAddress(OCXHandle, 'DllRegisterServer');
    if RegFunc > 0 then ShowMessage('등록에 실패했습니다 !')
    else AMath := CreateComObject(Class_Math) as IMath;
    FreeLibrary(OCXHandle);
  end;
end;
```

우선 CreateComObject 메소드를 이용해서 COM 객체의 생성을 시도한다. 이때 레지스트

리에 COM 객체가 등록되어 있지 않았다면, 예외가 발생할 것이다. 그러므로, try...except 구문을 이용하여 우리가 사용하고자 하는 액티브 X 서버 DLL 을 직접 등록한다.
등록할 때에는 액티브 X 서버 DLL 에 구현된 DllRegisterServer 함수의 주소를 이용하게 된다. 즉, TDllRegisterServer 형(ActiveX.pas 유닛에 정의되어 있다)의 변수를 선언하고 이 변수에 DllRegisterServer 함수의 주소를 얻어온 뒤에 COM 객체를 생성하면 된다.
이제 AMath 변수를 이용해서 IMath 인터페이스의 메소드를 호출할 수 있다.
그러면, Button1 의 OnClick 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X1, Y1, X2, Y2: Extended;
begin
  X1 := StrToFloat(Edit1.Text);
  Y1 := StrToFloat(Edit2.Text);
  X2 := StrToFloat(Edit3.Text);
  Y2 := StrToFloat(Edit4.Text);
  Label4.Caption := FloatToStr(AMath.Distance(X1, Y1, X2, Y2));
  AMath.XY2Polar(StrToFloat(Edit5.Text), StrToFloat(Edit6.Text), X1, Y1);
  Label8.Caption := FloatToStr(X1) + ', ' + FloatToStr(Y1);
  AMath.Polar2XY(StrToFloat(Edit7.Text), StrToFloat(Edit8.Text), X1, Y1);
  Label10.Caption := FloatToStr(X1) + ', ' + FloatToStr(Y1);
  Label12.Caption := FloatToStr(AMath.FactorialX(SpinEdit1.Value));
end;
```

내용이 평이하므로, 자세한 설명은 생략하도록 한다. 이것으로 클라이언트 프로그램이 완성되었다. 프로그램을 실행하고 에디트 박스와 SpinEdit 박스에 적당한 값을 입력하고 '계산' 버튼을 클릭하면 다음과 같은 계산 결과를 볼 수 있을 것이다.



- Initialize, Destory 메소드

TComObject 클래스의 constructor는 가상 함수가 아니다. 그렇지만, constructor는 항상 가상 메소드인 Initialize 메소드를 호출한다. 그러므로, 객체를 생성할 때 내부 필드 값을 가지고 있고, 그 값을 초기화하는 등의 작업이 필요한 경우에는 Initialize 메소드를 오버라이드해서 사용하면 된다. Destroy 함수는 가상 함수이기 때문에 직접 오버라이드가 가능하다. 여기서는 COM 객체를 구현하기 위해 사용한 여러가지 리소스 등을 해제하는 코드 등을 추가하게 된다. 간단하게 사용방법을 소개하면 다음과 같다.

```
TSampleObject(TComObject, ISampleInterface)
private
  FSample: string;
public
  function GetSample: string; stdcall;
  procedure SetSample(Value: string); stdcall;
  procedure Initialize; override;
  destructor Destroy; override;
end;

...

procedure TSampleObject.Initialize;
begin
  inherited;
  FSample := 'Default Value';
end;

procedure TSampleObject.Destroy;
begin
  inherited;
  ShowMessage('Object Destroyed !');
end;
```

정 리 (Summary)

이번 장에서는 COM 을 활용할 때 가장 기본이라고 할 수 있는 인터페이스의 사용 방법과 가장 원시적인 TInterfacedObject 클래스를 사용한 간단한 예제와 COM 객체 위저드를 이용하여 COM 객체 서버를 제작하고, 이렇게 제작된 서버를 사용하는 클라이언트 프로그램을 만들어서 활용하는 방법을 익혀 보았다.

COM 객체 서버는 기본적인 제작 방법만 알면 제작이 쉽기 때문에, 델파이 VCL 소스 코드로 된 여러가지 내용을 COM 객체 서버로 제작해서 DLL 로 만든 뒤에 이를 등록하여 비주얼 베이직이나 비주얼 C++과 같은 다른 언어에서 그대로 활용하게 할 수 있으므로 장점이 무척 많다. 자칫 수박 겉핥기처럼 읽고만 넘어갈 수도 있는 내용이지만, 위저드를 이용해서 액티브 X 컨트롤을 만들어 사용하는 것보다 훨씬 강력하고도 활용도가 높을 수 있는 내용이므로 반드시 자신의 것으로 만들고 넘어가기 바란다.

다음 장에서는 TOleContainer 컴포넌트를 이용하여 액티브 도큐먼트를 활용하는 방법과 OLE 자동화 컨트롤러를 만들어서 DAO, 엑셀 등을 직접 제어하는 방법, 마지막으로 OLE 자동화 서버를 제작하는 방법에 대해서 알아볼 것이다.