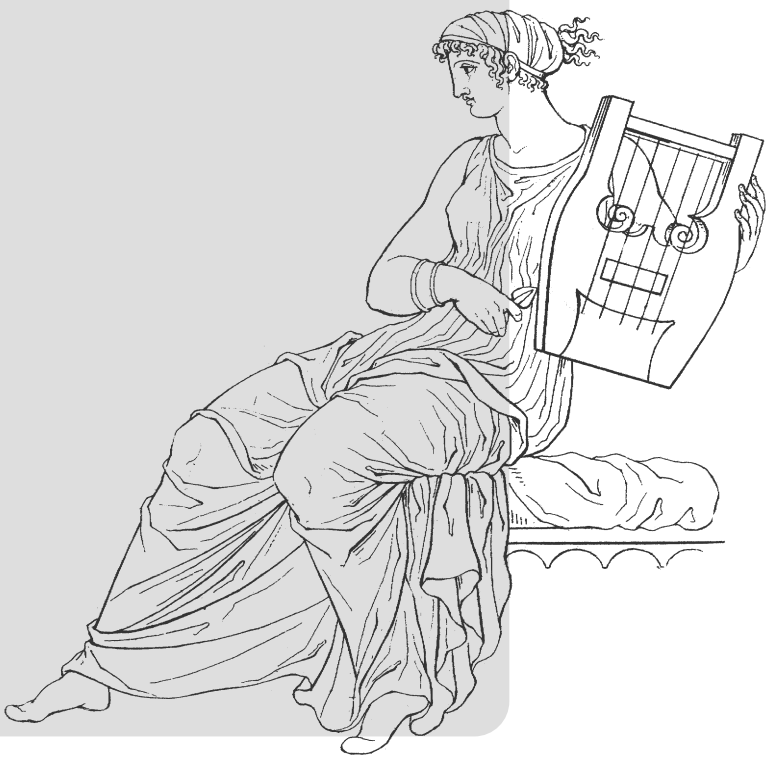
기본 언어 설명

1부에서는 프로그래밍 작업에서 일반적으로 필요한 필수 언어 요소들을 다룹니다.
각 장에서는 다음과 같은 내용들을 다룹니다.

- 1장 Delphi 언어 개요
- 2장 프로그램과 유닛
- 3장 문법 요소
- 4장 데이터 타입, 변수 및 상수
- 5장 프로시저와 함수
- 6장 클래스와 객체
- 7장 제네릭
- 8장 익명 메소드
- 9장 표준 루틴과 I/O

Part_

01



CHAPTER _1

Delphi 언어 개요

이 장에서는 델파이 언어에 대한 간단한 소개와 함께 델파이 프로젝트를구성하는 요소들에 대해 알아보고, 간단한 예제 몇 개를 살펴봄으로써 델파이 프로젝트에 대한 감각을 익힙니다.

■프로그램의 구성 ■예제 프로그램들

델파이(Delphi) 언어는 고급 언어이자 컴파일 기반 언어이며, 타입 지정이 엄격한 언어로서, 구조적 설계와 객체 지향 설계를 지원합니다. 델파이 언어는 오브젝트 파스칼을 기반으로 하고 있으며, 읽기 쉬운 코드, 빠른 컴파일, 모듈화 프로그래밍을 위한 복수의 유닛 파일을 지원하는 등의 장점이 있습니다. 델파이는 코드기어의 컴포넌트 프레임워크와 RAD 환경을 지원하는 특수한 기능들을 갖추고 있습니다. 이 책의 내용과 예제는 여러분이 코드기어 개발툴을 사용하고 있다고 가정합니다.

코드기어 개발툴을 사용하는 대부분의 개발자들은 통합 개발환경(IDE) 내에서 자신의 코드를 작성하고 컴파일합니다. 코드기어 개발툴들은 유닛들 사이의 의존성 정보들의 관리 작업 등 프로젝트와 소스 파일들을 설정하는 많은 세부적인 작업들을 처리해줍니다. 또한 코드기어 개발툴은 엄밀하게는 오브젝트 파스칼 언어 표준에 포함되지 않는 제한 사항들을 프로그램 구성에 추가합니다. 예를 들어, 코드기어 개발툴은 IDE 외부에서 프로그램을 작성하여 커맨드라인에서 컴파일한다면 필요하지 않은 파일 및 프로그램에 대한 이름 지정 규칙을 따르도록 강제합니다.

이 책에서는 여러분이 IDE에서 작업을 하며 코드기어 비주얼 컴포넌트 라이브러리 (VCL)

를 이용하여 애플리케이션을 개발한다고 가정합니다. 하지만, 때로는 델파이만의 규칙들을 모든 오브젝트 파스칼 프로그래밍에 적용되는 규칙들과는 구분할 것입니다.

프로그램의 구성

델파이 프로그램은 보통 유닛(unit)이라고 하는 여러 개의 소스 코드 모듈로 구성됩니다. 대부분의 프로그램은 가장 선두에서 program으로 프로그램의 이름을 지정합니다. 그 다음으로 선택적으로 uses 절이 올 수 있으며, 그뒤에 선언과 문장 블럭이 옵니다. uses 절은 해당 프로그램에 연관된 유닛들을 나열하는데, 이 유닛들은 다른 프로그램에서도 사용될 수도 있으며, 종종 자체적으로도 uses 절을 가지고 있습니다.
uses 절은 컴파일러에게 모듈들 사이의 종속 관계 정보를 알려줍니다. 이러한 종속 관계에 대한 정보가 모듈 자체에 저장되기 때문에, 델파이 프로그램에는 make 파일, 헤더 파일이나 "include" 지시어가 필요없습니다.

델파이 소스 파일들

델파이 컴파일러는 다음과 같은 세 가지 파일들을 델파이 소스 파일로 취급합니다.

- 유닛 소스 파일 (확장자 : .pas)
- 프로젝트 파일 (확장자 : .dpr)
- 패키지 소스 파일 (확장자 : .dpk)

일반적으로 유닛 소스 파일에는 애플리케이션 내의 코드 대부분이 들어갑니다. 각 애플리케이션은 프로젝트 파일 하나와 여러 유닛 파일들을 포함합니다. 프로젝트 파일은 전통적인 파스칼에서의 프로그램 파일에 해당하는 것으로, 여러 유닛 파일을 하나의 애플리케이션으로 구성합니다. 코드기어 개발툴은 각 애플리케이션에 대해 자동으로 하나의 프로젝트 파일을 유지합니다.
커맨드라인에서 프로그램을 컴파일할 경우, 유닛(.pas) 파일에 소스 코드 전체를 넣을 수 있습니다. IDE를 사용하여 애플리케이션을 빌드하는 경우에는 프로젝트(.dpr) 파일이 생성됩니다.
패키지 소스 파일은 프로젝트 파일과 유사하지만, '패키지' 라는 특수한 DLL을 개발할 경우에만 사용됩니다. 패키지에 대해 더 자세히 알아보려면 10장 'DLL과 패키지' 를 살펴보십시오.

애플리케이션을 구성하는 다른 파일들

소스 코드 모듈 이외에 코드기어 제품들에서는 애플리케이션을 빌드하는 데 파스칼 파일 외에 몇 가지 파일들도 사용합니다. 이러한 파일들은 IDE에서 자동으로 유지 관리되며 다음과 같은 파일들이 여기에 속합니다.

- VCL 폼 파일 (확장자: .dfm)
- 리소스 파일 (확장자: .res)
- 프로젝트 옵션 파일 (확장자: .dof)

폼 파일은 폼과 폼에 포함된 컴포넌트들의 속성 내용들을 포함하는 파일입니다. 각폼 파일은 폼 하나를 나타내는데, 일반적인 애플리케이션의 윈도우나 다이얼로그에 해당합니다. IDE에서는 폼 파일을 텍스트 형태로 보거나 편집할 수 있으며, 텍스트 파일(버전 컨트롤에 적합한 포맷) 혹은 바이너리 파일로 저장할 수 있습니다. 기본적으로 폼 파일은 텍스트로 저장되지만, 폼 파일을 편집하기 위해서는 보통 수작업으로 편집하지 않고 코드기어의 비주얼 디자인 툴을 사용합니다. 각 프로젝트는 하나 이상의 폼이 포함되며, 기본적으로 각 폼은 폼 파일과 같은 이름의 유닛(.pas) 파일에 연결되어 있습니다.

폼 파일 외에, 각 프로젝트는 리소스(.res) 파일에 애플리케이션 아이콘 비트맵과 문자열 등의 리소스들을 저장합니다. 기본적으로 이 리소스 파일은 프로젝트(.dpr) 파일과 같은 이름을 가집니다. 애플리케이션 아이콘을 변경하려면 Project Options 다이얼로그를 사용합니다.

프로젝트 옵션 파일(.dof)에는 컴파일러와 링커 설정, 서치 패스(Search Path), 버전 정보 등이 포함되어 있습니다. 각 프로젝트에는 프로젝트(.dpr) 파일과 같은 파일 이름을 가진 연결된 프로젝트 옵션 파일이 있습니다. 이 파일에 포함된 옵션 사항들은 보통 Project Options 다이얼로그에서 설정합니다.

IDE의 여러 툴들은 각각의 데이터를 저장기 위해 다른 종류의 파일들을 사용합니다. 데스크톱 설정 파일(.dsk)에는 윈도우 배치와 기타 설정 옵션들이 포함되어 있는데, 데스크톱 설정은 특정 프로젝트에만 적용할 수도 있고, 환경 전체에 적용할 수도 있습니다. 이런 파일들은 컴파일에 직접적인 영향을 미치지는 않습니다.

컴파일러 생성 파일

애플리케이션이나 패키지를 처음 빌드하면 컴파일러는 프로젝트에 포함된 각각의 새 유닛들에 대해 컴파일된 유닛 파일(.dcu)을 만듭니다. 다음으로 프로젝트의 모든 .dcu 파일들을 링크하여 실행 파일 또는 공유 패키지 파일을 생성합니다. 패키지를 처음으로 빌드하면 컴

파일러는 패키지에 포함된 유닛들의 .dcu 파일을 만든 다음 .dcp 파일과 .bpl 파일을 만듭니다. -GD 스위치를 사용할 경우, 링커는 맵 파일과 .drc 파일을 만듭니다. .drc 파일은 문자열 리소스들을 포함하고 있으며 컴파일하면 리소스 파일이 생성됩니다.
프로젝트를 다시 빌드할 때는, 마지막 컴파일 후에 소스 파일(.pas)이 변경되었거나, 유닛의 .dcu 파일이 없거나, 사용자가 명시적으로 유닛을 다시 처리하도록 지정하거나, 유닛의 인터페이스가 다른 유닛에 의존하고 그 유닛이 변경되지 않은 한, 각 유닛은 다시 컴파일되지 않습니다. 사실, 컴파일러가 .dcu을 찾을 수 있고 또 해당 유닛이 의존하는 다른 유닛들도 변경되지 않았다면 컴파일러는 유닛 소스 파일에 대해 전혀 상관하지 않습니다.

예제 프로그램들

다음의 예제들은 델파이 프로그래밍의 기본적인 특징들을 보여줍니다. 이 예제는 IDE에서는 정상적으로 컴파일 할 수 없는 간단한 애플리케이션을 보여줍니다. 커맨드라인에서 이 코드를 컴파일할 수 있습니다.

간단한 콘솔 애플리케이션

아래의 프로그램은 커맨드 프롬프트에서 컴파일 및 실행할 수 있는 간단한 콘솔 애플리케이션입니다.

```
program Greeting;

{$APPTYPE CONSOLE}

var MyMessage: string

begin
  MyMessage := 'Hello world!';
  Writeln(MyMessage);
end.
```

첫 번째 행은 Greeting이라는 프로그램을 선언합니다. {$APPTYPE CONSOLE} 지시어는 컴파일러에게 이 프로그램이 커맨드라인에서 실행할 콘솔 애플리케이션임을 알려줍니다. 다음 행에서는 MyMessage라는 변수를 선언하는데, 이 변수는 문자열 변수입니다. (델파이에는 문자열만을 위한 데이터 타입이 있습니다) 다음으로, MyMessage 변수에 "Hello

world!"라는 문자열을 지정하고, Writeln 프로시저를 사용하여 MyMessage의 내용을 표준 출력으로 보냅니다. (Writeln은 System 유닛에 암시적으로 정의되어 있으며, 컴파일러는 모든 애플리케이션에서 자동으로 이 유닛을 포함시킵니다)
이 프로그램을 Greeting.pas 또는 Greeting.dpr이라는 이름의 파일로 저장한 후, 커맨드라인에서 다음과 같이 입력하여 파일을 컴파일합니다.

```
dcc32 Greeting
```

실행 결과로 "Hello world!"라는 메시지가 나타납니다.
단순하다는 점을 제외하고, 이 예제에는 코드기어 개발툴로 개발하는 일반적인 프로그램들과는 몇 가지 중요한 차이점들이 있습니다. 첫째, 이 예제는 콘솔 애플리케이션입니다. 보통 코드기어 개발툴은 그래픽 인터페이스를 가진 애플리케이션을 개발할 목적으로 사용되기 때문에 Writeln을 호출하는 경우는 일반적이지 않습니다. 또 예제 프로그램 전체가 하나의 파일로 되어 있습니다. 일반적인 GUI 애플리케이션에서는 예제의 첫번째 라인인 프로그램 헤딩 부분은 별도의 프로젝트 파일에 포함되어 있으며, 몇가지 유닛 파일들에 정의된 루틴들의 호출들을 제외하면 실질적인 애플리케이션 로직을 갖지 않습니다.

좀 더 복잡한 예제

다음 예제는 프로젝트 파일과 유닛 파일로 나뉘어진 프로그램을 보여줍니다. 아래 내용을 Greeting.dpr이라는 이름으로 저장하시기 바랍니다.

```
program Greeting;

{$APPTYPE CONSOLE}

uses Unit1;

begin
  PrintMessage('Hello World!');
end.
```

첫 번째 줄은 Greeting이라는 프로그램을 선언하는데, 역시 콘솔 애플리케이션입니다. uses Unit1; 절은 컴파일러에게 Greeting이 Unit1이라는 유닛을 포함한다는 것을 알려줍니다. 마지막으로 PrintMessage 프로시저를 호출하여 "Hello World!" 문자열을 전달합니다. PrintMessage 프로시저는 Unit1에 정의되어 있습니다. 다음은 Unit1.pas의 소스 코드입니다.

```
unit Unit1;

interface

procedure PrintMessage(msg: string);

implementation

procedure PrintMessage(msg: string);
begin
  Writeln(msg);
end;

end.
```

Unit1에는 PrintMessage 라는 프로시저가 정의되어 있는데, 이 프로시저는 하나의 문자열 인수를 가지며 이 문자열을 표준 출력으로 내보냅니다. (파스칼에서는 값을 리턴하지 않는 루틴을 프로시저라고 합니다. 값을 리턴하는 루틴은 함수라고 합니다.) PrintMessage가 Unit1에서 두 번 선언된 점에 눈여겨 보십시오. interface 예약어 아래에 있는 첫 번째 선언은 Greeting 같이 Unit1을 사용(uses)하는 다른 모듈에서 PrintMessage를 호출할 수 있도록 합니다. implementation 예약어 아래에 있는 두 번째 선언은 실제로 PrintMessage를 정의합니다.

이제 커맨드라인에 다음과 같이 입력하여 Greeting을 컴파일할 수 있습니다.

```
dcc32 Greeting
```

커맨드라인 인수로 Unit1을 지정할 필요는 없습니다. 컴파일러는 Greeting.dpr을 처리할 때 자동으로 Greeting 프로그램에서 필요로 하는 유닛 파일을 찾습니다. 실행 결과는 첫 번째 예제와 마찬가지로 "Hello world!" 메시지를 출력합니다.

VCL 애플리케이션

다음 예제는 IDE에서 VCL 컴포넌트를 사용하여 개발하는 애플리케이션입니다. 이 프로그램은 자동으로 생성되는 폼과 리소스 파일을 사용하므로 소스 코드만 단독으로 컴파일할 수 없습니다. 하지만 이 프로그램은 델파이 언어의 중요한 특징들을 보여줍니다. 이 프로그램은 여러 개의 유닛을 사용하며 클래스와 객체(6장에서 설명)도 사용합니다.

이 프로그램에는 하나의 프로젝트 파일과 두 개의 새 유닛 파일이 포함되어 있습니다. 프로

젝트 파일은 다음과 같습니다.

```
program Greeting;

uses
  Forms, Unit1, Unit2;

{$R *.res} {이 지시어는 프로젝트의 리소스 파일을 연결합니다}

begin
  {전역 Application 객체를 호출합니다}
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

이 프로그램 이름은 역시 greeting입니다. 이 프로그램은 VCL에 포함되어 있는 Forms, 애플리케이션의 메인 폼인 Form1과 연결되어 있는 Unit1, 그리고 폼 Form2와 연결되어 있는 Unit2 유닛을 사용합니다.

이 프로그램은 Forms 유닛에 정의된 TApplication 클래스의 인스턴스인 Application 객체를 여러 번 호출합니다. (모든 프로젝트에는 자동으로 생성된 Application 객체가 있습니다.) TApplication의 메소드 CreateForm를 두 번 호출하는데, 첫 번째 CreateForm 호출은 Unit1에서 정의한 TForm1 클래스의 인스턴스인 Form1을 만듭니다. 두 번째 CreateForm 호출은 Unit2에서 정의한 TForm2 클래스의 인스턴스인 Form2를 만듭니다. Unit1은 다음과 같습니다.

```
unit Unit1;

interface

uses SysUtils, Types, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
    Button1:TButton;
    procedure Button1Click(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation
```

```
uses Unit2;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form2.ShowModal;
end;

end.
```

Unit1은 TForm1(TForm에서 파생됨)이라는 클래스와 이 클래스의 인스턴스인 Form1을 만듭니다. TForm1에는 TButton의 인스턴스인 Button1 버튼과, 사용자가 Button1을 클릭할 때 호출되는 TForm1.Button1Click이라는 프로시저가 포함되어 있습니다. Form1.Button1Click은 Form1을 숨기고 Form2.ShowModal을 호출하여 Form2를 보여줍니다.

Form2는 Unit2에 다음과 같이 정의되어 있습니다.

Note

이 예제에서, Form2.ShowModal은 자동 생성되는(auto-created) 폼을 사용합니다. 예제 코드에서 아무런 문제가 없기는 하지만, 자동 생성되는 폼을 사용하는 것은 권장되지 않습니다.

```
unit Unit2;

interface

uses SysUtils, Types, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm2 = class(TForm)
    Label1: TLabel;
    CancelButton: TButton;
    procedure CancelButtonClick(Sender: TObject);
  end;

var
  Form2: TForm2;

implementation

uses Unit1;

{$R *.dfm}

procedure TForm2.CancelButtonClick(Sender: TObject);
begin
  Form2.Close;
end;

end.
```

Unit2는 TForm2라는 클래스와 이 클래스의 인스턴스인 Form2를 만듭니다. TForm2에는 CancelButton 버튼(TButton의 인스턴스)과 Label1 레이블(TLabel의 인스턴스)이 포함되어 있습니다. 소스 코드에서는 볼 수 없지만, Label1에는 "Hello world!"라는 캡션이 지정되어 있습니다. 이 캡션은 Form2의 폼 파일인 Unit2.dfm에서 지정됩니다.

TForm2에는 TForm2.CancelButtonClick라는 프로시저가 정의됩니다. 이 프로시저는 프로그램 실행중에 사용자가 CancelButton을 클릭할 때마다 호출됩니다. 이 프로시저는 Unit1의 TForm1.Button1Click과 마찬가지로 프로그램이 실행되는 동안 발생하는 이벤트에 응답하기 때문에 이를 이벤트 핸들러라고 합니다. Form1과 Form2에 대한 폼 파일에서 특정 이벤트에 이벤트 핸들러가 지정됩니다.

greeting 프로그램을 실행하면 Form1이 나타나고 Form2는 보이지 않습니다. (기본적으로 프로젝트 파일에서 가장 먼저 생성된 폼만 자동으로 나타납니다. 이 폼을 프로젝트의 메인 폼이라고 합니다.) 사용자가 Form1에 있는 버튼을 누르면 "Hello world!"라는 메시지를 가진 Form2가 나타납니다. CancelButton이나 제목 표시줄의 Close 버튼을 클릭하면 Form2가 닫힙니다.