7장 ActiveX 컨트롤 활용

* ActiveX 컨트롤이란 무엇인가?
* 언제 ActiveX 컨트롤을 사용하는가?
* ActiveX 컨트롤을 컴포넌트 팔레트에 추가하려면
* 델파이 컴포넌트 래퍼
* 애플리케이션에서 ActiveX 컨트롤을 사용하려면
* ActiveX 컨트롤을 사용하는 애플리케이션의 배포
* OCX 예제 애플리케이션 “블랙잭”

델파이에서는 업계 표준이라고 할 수 있는 ActiveX 컨트롤(이전에는 OLE 컨트롤이라고 불렀다)도 원활하게 사용할 수 있다. 델파이 고유의 컴포넌트와는 달리, ActiveX 컨트롤은 개발 도구의 종류에 상관 없이 사용할 수 있도록 설계된 것이다. 다시 말해, 같은 ActiveX 컨트롤을 델파이에서도 사용할 수 있고, 비주얼 베이직이나 C++ 빌더에서도 사용할 수 있다는 이야기이다. 그런 이점 때문에 상당히 다양한 ActiveX 컨트롤들이 시장에 나와 있고, ActiveX 컨트롤만을 전문으로 하는 소프트웨어 개발사들도 많이 있다.

32비트 델파이에서 ActiveX 컨트롤을 사용하는 방법은 16비트 델파이 1에서 VBX를 사용했던 방법과 비슷하다. 즉, 델파이 IDE의 메뉴나 패키지 편집기(Package Editor)를 통해서 새로운 ActiveX 컨트롤을 추가하면 델파이는 그 ActiveX 컨트롤에 대한 오브젝트 파스칼 래퍼(wrapper)를 생성시킨다. 래퍼가 만들어진 후 ActiveX 컨트롤을 패키지에 컴파일해서 넣으면 최종적으로 델파이 컴포넌트 형태의 ActiveX 컨트롤이 델파이의 컴포넌트 팔레트에 추가된다. 일단 팔레트에 추가되면 그때부터는 다른 VCL 컴포넌트와 동일한 방식으로 다룰 수 있다. 즉, ActiveX 컨트롤을 폼에 추가하거나, 애플리케이션의 코드에서 ActiveX 컨트롤의 속성, 메소드들을 사용할 수 있는 것이다. 이번 장은 델파이에 ActiveX 컨트롤을 추가하는 방법과 애플리케이션에서 ActiveX 컨트롤을 사용하는 방법, 그리고 ActiveX 컨트롤을 사용하는 애플리케이션을 배포하는 방법에 대해 이야기하겠다.



델파이 5는 VBX(Visual Basic Extension) 컨트롤을 지원하지 않는다. VBX 컨트롤이 꼭 필요한 델파이 1 프로젝트를 델파이 5로 옮겨야 한다면 그 VBX에 해당하는 ActiveX 컨트롤을 구해야만 한다.

ActiveX 컨트롤이란 무엇인가?

ActiveX 컨트롤은 OLE(Object Linking and Embedding, 객체 연결 및 내장) 기술과 ActiveX 컨트롤 기술을 사용하는 16비트/32비트 윈도우즈 애플리케이션을 위한 커스텀 컨트롤이다. 16비트 비주얼 베이직을 위해 만들어진(그래서 비주얼 베이직의 한계를 그대로 이어받은) VBX와는 달리, ActiveX 컨트롤은 애플리케이션에 비의존적인 컨트롤로 작동하도록 설계되었다. 이번 장에서는 OLE와 ActiveX 기술을 기본적으로 동일한 것으로 간주한다. OLE와 ActiveX의 핵심적인 차이점에 대해서는 23장, "COM과 ActiveX"를 보기 바란다.

ActiveX 컨트롤을 한꺼풀 벗겨보면 ActiveX 컨트롤이란, 하나의 패키지로 된 ActiveX 서버이다. ActiveX 컨트롤은 모든 OLE 기능이나 서비스들을 포함하고 있으며, 비주얼한 편집이나 드래그 앤 드롭, OLE 자동화를 지원하는 특별한 종류의 ActiveX 서버라고 할 수 있다. 다른 모든 ActiveX 서버들과 마찬가지로, ActiveX 컨트롤은 시스템 레지스트리에 등록된다. ActiveX 컨트롤을 만들 수 있는 개발 도구에는 델파이, 볼랜드 C++ 빌더, 비주얼 C++, 비주얼 베이직 등이 있다.

마이크로소프트는 ActiveX 컨트롤을 애플리케이션에 비의존적인 사용자 컨트롤 시장의 대표 주자로 밀고 있다. 마이크로소프트는 VBX 기술이 Win32 운영체제를 직접적으로 지원하는 일은 없을 것이라고 말했다. 다시 말하면, 32비트 애플리케이션 개발자는 VBX에 대해서는 잊어버리고, 적당한 ActiveX 컨트롤을 골라야 하는 상황이 된 것이다.



ActiveX 컨트롤 기술에 대한 자세한 내용은 25장, "ActiveX 컨트롤 만들기"에 나온다.

언제 ActiveX 컨트롤을 사용하는가?

델파이 고유의 컴포넌트 대신 ActiveX 컨트롤을 사용해야 하는 이유는 크게 두 가지이다. 첫 번째 이유는, 특별한 용도로 사용할 델파이 컴포넌트가 항상 준비되어 있는 것이 아니라는 점이다. ActiveX 컨트롤 시장은 VCL 컴포넌트 시장보다 훨씬 크기 때문에, 예를 들어서 웹 브라우저나 스프레드시트 기능이 필요한 경우에 VCL 컴포넌트를 구하는 것보다 ActiveX 컨트롤을 구하는 편이 훨씬 더 쉽다. 두 번째 이유는, 델파이와 델파이 이외의 개발 도구를 동시에 사용해서 애플리케이션을 개발하는 경우, ActiveX 컨트롤을 쓰는 쪽이 훨씬 더 편리하기 때문이다. 즉, 하나의 ActiveX 컨트롤을 공부해서 두 개발 도구에서 공통적으로 써먹을 수 있게 된다.

ActiveX 컨트롤은 델파이 IDE에 거의 완벽하게 통합되지만, 그래도 델파이 컴포넌트를 사용하는 것에 비하면 단점이 있을 수밖에 없다. 무엇보다도 델파이 컴포넌트들은 실행 파일에 직접 링크시킬 수 있지만, ActiveX 컨트롤을 사용할 경우에는 그 ActiveX 컨트롤(추가적인 런타임 DLL이 필요할 수도 있다)을 실행 파일과 함께 배포해야 한다. 또한, 델파이 컴포넌트는 애플리케이션이나 다른 컴포넌트들과 직접 통신할 수 있지만, ActiveX 컨트롤은 애플리케이션과 COM 계층을 통해서만 통신을 할 수 있다. 즉, 수행 속도 측면에서 "잘 만든 델파이 컴포넌트 하나 열 ActiveX 안부럽다"라는 의미가 된다. 결론적으로 말해서, "ActiveX 컨트롤은 개발 도구나 애플리케이션에 비의존적이다"라는 특성은 특정 개발 도구만의 고유한 기능을 최대한 발휘하지 못한다는 면에서 단점이라고 할 수 있다.

ActiveX 컨트롤을 컴포넌트 팔레트에 추가하려면

델파이 애플리케이션에 어떤 ActiveX 컨트롤을 사용하려면, 우선 그 컨트롤을 델파이 IDE의 컴포넌트 팔레트에 추가해야 한다. ActiveX 컨트롤을 컴포넌트 팔레트에 추가하면 그 컨트롤을 뜻하는 아이콘이 컴포넌트 팔레트에 나타난다. 일단 ActiveX 컨트롤이 팔레트에 추가된 다음부터는 다른 델파이 컴포넌트들과 마찬가지 방식으로 컨트롤을 폼에 추가해서 사용하면 된다.

ActiveX 컨트롤을 컴포넌트 팔레트에 추가하는 과정은 다음과 같다.

1. 메인 메뉴에서 Component | Import ActiveX Control을 선택해서 Import ActiveX Control 대화상자를 띄운다(그림 7.1).

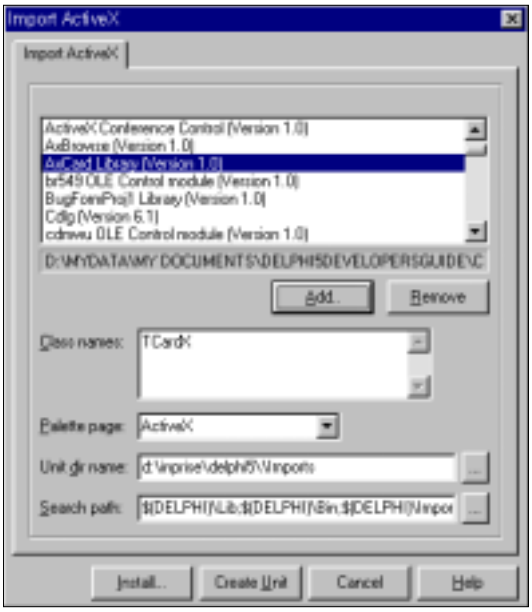


그림 7.1
Import ActiveX Control 대화상자

2 Import ActiveX Control 대화상자는 두 부분으로 나뉘어져 있다. 윗부분에는 레지스트리에 등록된 ActiveX 컨트롤들이 있는 목록 상자가 있으며, 또 ActiveX 컨트롤을 등록하거나 등록 해제할 수 있는 Add와 Remove 버튼이 있다. 아랫부분에는 ActiveX 컨트롤을 캡슐화하는 델파이 컴포넌트와 유닛에 관련된 컨트롤들이 있다.

3 사용하고자 하는 ActiveX 컨트롤이 윗부분의 목록에 있으면 4로 넘어간다. 만일 없으면 Add 버튼을 클릭해서 컨트롤을 시스템에 등록해야 한다. Add 버튼을 클릭하면 그림 7.2와 같은 Register OLE Control 대화상자가 나타난다. 등록할 ActiveX 컨트롤에 해당하는 OCX나 DLL 파일을 선택한 후 열기 버튼을 클릭하면 그 ActiveX 컨트롤이 시스템 레지스트리에 등록되며, Register OLE Control 대화상자는 닫힌다.

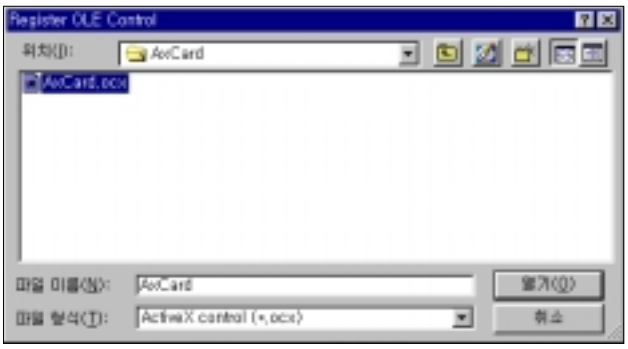


그림 7.2
Register OLE Control 대화상자

4 Import ActiveX Control 대화상자의 윗부분 목록에서 컴포넌트 팔레트에 추가하고자 하는 ActiveX 컨트롤을 선택한다. 그리고 아랫부분에는 유닛 파일의 이름과 팔레트 페이지, 검색 경로를 입력하는 텍스트 상자들과 OCX 파일에 포함된 클래스들이 나열되는 메모 컨트롤이 있다. Unit File Name 텍스트 상자는 ActiveX 컨트롤을 캡슐화하는 델파이 래퍼 컴포넌트의 경로와 파일 이름을 뜻한다. 기본 파일 이름은 OCX 파일의 이름과 같다(확장자는 .pas). 기본 경로는 \Delphi5\Imports이다. 기본 설정을 그대로 써도 되지만, 원한다면 다른 파일 이름이나 경로를 입력해도 된다.

5 Palette Page 텍스트 상자에는 컴포넌트 팔레트의 페이지들 중 ActiveX 컨트롤이 들어갈 페이지의 이름을 설정한다. 기본 페이지는 ActiveX 페이지이다. 다른 페이지를 선택해도 되고, 새로운 페이지 이름을 입력하면 컴포넌트 팔레트에 그 이름으로 된 새 페이지가 생긴다.

6 Class Names 메모 컨트롤에는 이 ActiveX 컨트롤에 생기는 새 객체들의 이름이 나타난다. 특별한 이유가 없는 한 그 이름들은 그대로 놔두는 것이 좋다. 단, IDE에 이미 설치되어 있는 컴포넌트의 클래스 이름과 중복되는 경우에는 이름을 적절히 변경해야 한다.

7 여기까지 왔으면 Install이나 Create Unit 버튼을 클릭할 준비가 다 된 것이다. Create Unit 버튼은 ActiveX 컴포넌트 래퍼 유닛을 만드는 것이다. Install 버튼은 래퍼를 만든 후 Install 대화상자를 띄우는데, 그 대화상자에서 컴포넌트를 패키지에 추가한다(그림 7.3).

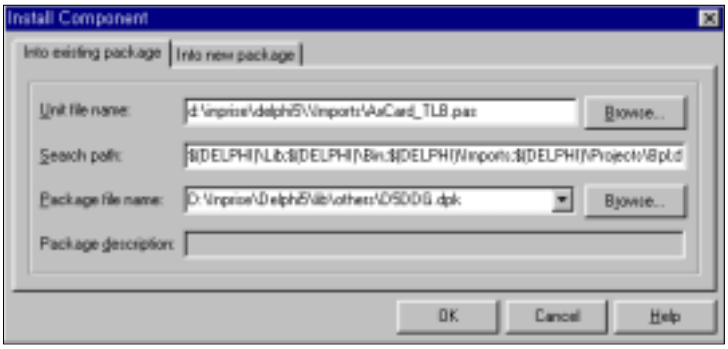


그림 7.3
Install 대화상자

8 컨트롤을 기존의 패키지에 추가할 때에는 Into existing package 페이지에서 원하는 패키지를 선택한다. 새로운 패키지를 만들어서 추가하려면 Into new package 페이지를 사용하면 된다. OK 버튼을 클릭하면 컨트롤(컴포넌트 형태로 변환되었음)이 팔레트에 설치된다.

위와 같은 과정을 거쳐서 ActiveX 컨트롤을 일단 컴포넌트 팔레트에 추가한 다음부터는 델파이 컴포넌트를 사용하는 것과 마찬가지 방식으로 사용하면 된다.

델파이 컴포넌트 래퍼

그럼, ActiveX 컨트롤을 캡슐화하는 오브젝트 파스칼 래퍼(wrapper)를 살펴보자. 래퍼를 들여다 보면 델파이가 ActiveX 기술을 어떻게 지원하는지 알 수 있으며, 또한 ActiveX 컨트롤의 능력과 한계도 더욱 잘 알 수 있다. 목록 7.1은 델파이가 만들어낸 Card_TBL.pas 유닛의 코드이다. 이 유닛은 AxCard.ocx라는 ActiveX 컨트롤의 래퍼이다.

참고

AxCard.ocx는 25장에서 만든 ActiveX 컨트롤이다.

목록 7.1 AxCard.ocx의 델파이 컴포넌트 래퍼 유닛(역주 : 한글로 된 주석은 델파이가 만들어낸 것이 아니라, 독자의 이해를 돕기 위해서 역자가 델파이의 영문 주석을 번역한 것임).

```
unit AxCard_TLB;

interface

uses Windows, ActiveX, Classes, Graphics, OleServer, OleCtrls, StdVCL;

const
  // TypeLibrary Major와 minor 버전
  AxCardMajorVersion = 1;
  AxCardMinorVersion = 0;

  LIBID_AxCard: TGUID = '{7B33D940-0A2C-11D2-AE5C-04640BC10000}';

  IID_ICardX: TGUID = '{7B33D941-0A2C-11D2-AE5C-04640BC10000}';
  DIID_ICardXEvents: TGUID = '{7B33D943-0A2C-11D2-AE5C-04640BC10000}';
  CLASS_CardX: TGUID = '{7B33D945-0A2C-11D2-AE5C-04640BC10000}';
// *********************************************************************//
// 타입 라이브러리에 정의된 열거형들의 선언
// *********************************************************************//
// TxDragMode 상수들
type
  TxDragMode = TOleEnum;
const
  dmManual = $00000000;
  dmAutomatic = $00000001;

// TxCardSuit 상수들
type
  TxCardSuit = TOleEnum;
const
  csClub = $00000000;
  csDiamond = $00000001;
  csHeart = $00000002;
  csSpade = $00000003;
```

```
// TxCardValue 상수들
type
  TxCardValue = TOleEnum;
const
  cvAce = $00000000;
  cvTwo = $00000001;
  cvThree = $00000002;
  cvFour = $00000003;
  cvFive = $00000004;
  cvSix = $00000005;
  cvSeven = $00000006;
  cvEight = $00000007;
  cvNine = $00000008;
  cvTen = $00000009;
  cvJack = $0000000A;
  cvQueen = $0000000B;
  cvKing = $0000000C;

// TxMouseButton 상수들
type
 TxMouseButton = TOleEnum;
const
 mbLeft = $00000000;
 mbRight = $00000001;
 mbMiddle = $00000002;

// TxBevelCut 상수들
type
 TxBevelCut = TOleEnum;
const
 bvNone = $00000000;
 bvLowered = $00000001;
 bvRaised = $00000002;
 bvSpace = $00000003;

// TxBevelKind 상수들
type
TxBevelKind = TOleEnum;
const
```

```
 bkNone = $00000000;
 bkTile = $00000001;
 bkSoft = $00000002;
 bkFlat = $00000003;

// TxAlignment 상수들
type
 TxAlignment = TOleEnum;
const
 taLeftJustify = $00000000;
 taRightJustify = $00000001;
 taCenter = $00000002;

// TxBiDiMode 상수들
type
 TxBiDiMode = TOleEnum;
const
 bdLeftToRight = $00000000;
 bdRightToLeft = $00000001;
 bdRightToLeftNoAlign = $00000002;
 bdRightToLeftReadingOnly = $00000003;

// TxDockOrientation 상수들
type
 TxDockOrientation = TOleEnum;
const
 doNoOrient = $00000000;
 doHorizontal = $00000001;
 doVertical = $00000002;

type

//*********************************************************//
// 타입 라이브러리에 정의된 인터페이스들의 선행 선언
//*********************************************************//
ICardX = interface;
ICardXDisp = dispinterface;
ICardXEvents = dispinterface;
```

```
//******************************************************************//
// 타입 라이브러리에 정의된 COM 클래스(CoClass)들의 선언
// (주의 : 각 COM 클래스를 자신의 기본 인터페이스에 연결시켰음.)
//******************************************************************//
CardX = ICardX;

//******************************************************************//
// Interface: ICardX
// Flags: (4416) Dual OleAutomation Dispatchable
// GUID:  {7B33D941-0A2C-11D2-AE5C-04640BC10000}
//******************************************************************//
ICardX = interface(IDispatch)
    ['{7B33D941-0A2C-11D2-AE5C-04640BC10000}']
 function Get_BackColor: OLE_COLOR; safecall;
 procedure Set_BackColor(Value: OLE_COLOR); safecall;
 function Get_Color: OLE_COLOR; safecall;
 procedure Set_Color(Value: OLE_COLOR); safecall;
 function Get_DragCursor: Smallint; safecall;
 procedure Set_DragCursor(Value: Smallint); safecall;
 function Get_DragMode: TxDragMode; safecall;
 procedure Set_DragMode(Value: TxDragMode); safecall;
 function Get_FaceUp: WordBool; safecall;
 procedure Set_FaceUp(Value: WordBool); safecall;
 function Get_ParentColor: WordBool; safecall;
 procedure Set_ParentColor(Value: WordBool); safecall;
 function Get_Suit: TxCardSuit; safecall;
 procedure Set_Suit(Value: TxCardSuit); safecall;
 function Get_Value: TxCardValue; safecall;
 procedure Set_Value(Value: TxCardValue); safecall;
 function Get_DockClientCount: Integer; safecall;
 function Get_DoubleBuffered: WordBool; safecall;
 procedure Set_DoubleBuffered(Value: WordBool); safecall;
 procedure FlipChildren(AllLevels: WordBool); safecall;
 function DrawTextBiDiModeFlags(Flags: Integer): Integer; safecall;
 function DrawTextBiDiModeFlagsReadingOnly: Integer; safecall;
 function Get_Enabled: WordBool; safecall;
 procedure Set_Enabled(Value: WordBool); safecall;
 function GetControlsAlignment: TxAlignment; safecall;
 procedure InitiateAction; safecall;
```

```
    function IsRightToLeft: WordBool; safecall;
    function UseRightToLeftAlignment: WordBool; safecall;
    function UseRightToLeftReading: WordBool; safecall;
    function UseRightToLeftScrollBar: WordBool; safecall;
    function Get_BiDiMode: TxBiDiMode; safecall;
    procedure Set_BiDiMode(Value: TxBiDiMode); safecall;
    function Get_DockOrientation: TxDockOrientation; safecall;
    procedure Set_DockOrientation(Value: TxDockOrientation); safecall;
    function Get_Floating: WordBool; safecall;
    function Get_LRDockWidth: Integer; safecall;
    procedure Set_LRDockWidth(Value: Integer); safecall;
    function Get_TBDockHeight: Integer; safecall;
    procedure Set_TBDockHeight(Value: Integer); safecall;
    function Get_UndockHeight: Integer; safecall;
    procedure Set_UndockHeight(Value: Integer); safecall;
    function Get_UndockWidth: Integer; safecall;
    procedure Set_UndockWidth(Value: Integer); safecall;
    function Get_Visible: WordBool; safecall;
    procedure Set_Visible(Value: WordBool); safecall;
    function Get_Cursor: Smallint; safecall;
    procedure Set_Cursor(Value: Smallint); safecall;
    function ClassNameIs(const Name: WideString): WordBool; safecall;
    procedure AboutBox; safecall;
    property BackColor: OLE_COLOR read Get_BackColor write Set_BackColor;
    property Color: OLE_COLOR read Get_Color write Set_Color;
    property DragCursor: Smallint read Get_DragCursor write
        Set_DragCursor;
    property DragMode: TxDragMode read Get_DragMode write Set_DragMode;
    property FaceUp: WordBool read Get_FaceUp write Set_FaceUp;
    property ParentColor: WordBool read Get_ParentColor write
        Set_ParentColor;
    property Suit: TxCardSuit read Get_Suit write Set_Suit;
    property Value: TxCardValue read Get_Value write Set_Value;
    property DockClientCount: Integer read Get_DockClientCount;
    property DoubleBuffered: WordBool read Get_DoubleBuffered
        write Set_DoubleBuffered;
    property Enabled: WordBool read Get_Enabled write Set_Enabled;
    property BiDiMode: TxBiDiMode read Get_BiDiMode write Set_BiDiMode;
    property DockOrientation: TxDockOrientation read Get_DockOrientation
```

```
 write Set_DockOrientation;
 property Floating: WordBool read Get_Floating;
 property LRDockWidth: Integer read Get_LRDockWidth write
     Set_LRDockWidth;
 property TBDockHeight: Integer read Get_TBDockHeight
     write Set_TBDockHeight;
 property UndockHeight: Integer read Get_UndockHeight
     write Set_UndockHeight;
 property UndockWidth: Integer read Get_UndockWidth
     write Set_UndockWidth;
 property Visible: WordBool read Get_Visible write Set_Visible;
 property Cursor: Smallint read Get_Cursor write Set_Cursor;
end;

//******************************************************************//
// DispIntf: ICardXDisp
// Flags: (4416) Dual OleAutomation Dispatchable
// GUID: {7B33D941-0A2C-11D2-AE5C-04640BC10000}
//******************************************************************//
ICardXDisp = dispinterface
    ['{7B33D941-0A2C-11D2-AE5C-04640BC10000}']
    property BackColor: OLE_COLOR dispid 1;
    property Color: OLE_COLOR dispid -501;
    property DragCursor: Smallint dispid 2;
    property DragMode: TxDragMode dispid 3;
    property FaceUp: WordBool dispid 4;
    property ParentColor: WordBool dispid 5;
    property Suit: TxCardSuit dispid 6;
    property Value: TxCardValue dispid 7;
    property DoubleBuffered: WordBool dispid 10;
    procedure FlipChildren(AllLevels: WordBool); dispid 11;
    function  DrawTextBiDiModeFlags(Flags: Integer): Integer; dispid 14;
    function  DrawTextBiDiModeFlagsReadingOnly: Integer; dispid 15;
    property Enabled: WordBool dispid -514;
    function  GetControlsAlignment: TxAlignment; dispid 16;
    procedure InitiateAction; dispid 18;
    function  IsRightToLeft: WordBool; dispid 19;
    function  UseRightToLeftAlignment: WordBool; dispid 24;
    function  UseRightToLeftReading: WordBool; dispid 25;
```

```
    function  UseRightToLeftScrollBar: WordBool; dispid 26;
    property BiDiMode: TxBiDiMode dispid 27;
    property Visible: WordBool dispid 28;
    property Cursor: Smallint dispid 29;
    function  ClassNameIs(const Name: WideString): WordBool; dispid 33;
    procedure AboutBox; dispid -552;
  end;

//*********************************************************//
// DispIntf: ICardXEvents
// Flags: (4096) Dispatchable
// GUID:      {7B33D943-0A2C-11D2-AE5C-04640BC10000}
//*********************************************************************//
  ICardXEvents = dispinterface
    ['{7B33D943-0A2C-11D2-AE5C-04640BC10000}']
    procedure OnClick; dispid 1;
    procedure OnDblClick; dispid 2;
    procedure OnKeyPress(var Key: Smallint); dispid 7;
  end;


//*********************************************************//
// OLE Control Proxy class declaration
// Control Name : TCardX
// Help String : CardX Control
// Default Interface: ICardX
// Def. Intf. DISP? : No
// Event Interface: ICardXEvents
// TypeFlags : (34) CanCreate Control
//*********************************************************//
  TCardXOnKeyPress = procedure(Sender: TObject; var Key: Smallint) of
object;

  TCardX = class(TOleControl)
  private
    FOnClick: TNotifyEvent;
    FOnDblClick: TNotifyEvent;
    FOnKeyPress: TCardXOnKeyPress;
    FIntf: ICardX;
```

```
    function  GetControlInterface: ICardX;
  protected
    procedure CreateControl;
    procedure InitControlData; override;
  public
    procedure FlipChildren(AllLevels: WordBool);
    function  DrawTextBiDiModeFlags(Flags: Integer): Integer;
    function  DrawTextBiDiModeFlagsReadingOnly: Integer;
    function  GetControlsAlignment: TxAlignment;
    procedure InitiateAction;
    function  IsRightToLeft: WordBool;
    function  UseRightToLeftAlignment: WordBool;
    function  UseRightToLeftReading: WordBool;
    function  UseRightToLeftScrollBar: WordBool;
    function  ClassNameIs(const Name: WideString): WordBool;
    procedure AboutBox;
    property  ControlInterface: ICardX read GetControlInterface;
    property  DefaultInterface: ICardX read GetControlInterface;
    property DoubleBuffered: WordBool index 10 read GetWordBoolProp write
SetWordBoolProp;
    property Enabled: WordBool index -514 read GetWordBoolProp write
SetWordBoolProp;
    property BiDiMode: TOleEnum index 27 read GetTOleEnumProp write
SetTOleEnumProp;
    property Visible: WordBool index 28 read GetWordBoolProp write
SetWordBoolProp;
  published
    property  TabStop;
    property  Align;
    property  ParentShowHint;
    property  PopupMenu;
    property  ShowHint;
    property  TabOrder;
    property  OnDragDrop;
    property  OnDragOver;
    property  OnEndDrag;
    property  OnEnter;
    property  OnExit;
    property  OnStartDrag;
```

```
    property BackColor: TColor index 1 read GetTColorProp write
SetTColorProp stored False;
    property Color: TColor index -501 read GetTColorProp write
SetTColorProp stored False;
    property DragCursor: Smallint index 2 read GetSmallintProp write
SetSmallintProp stored False;
    property DragMode: TOleEnum index 3 read GetTOleEnumProp write
SetTOleEnumProp stored False;
    property FaceUp: WordBool index 4 read GetWordBoolProp write
SetWordBoolProp stored False;
    property ParentColor: WordBool index 5 read GetWordBoolProp write
SetWordBoolProp stored False;
    property Suit: TOleEnum index 6 read GetTOleEnumProp write
SetTOleEnumProp stored False;
    property Value: TOleEnum index 7 read GetTOleEnumProp write
SetTOleEnumProp stored False;
    property Cursor: Smallint index 29 read GetSmallintProp write
SetSmallintProp stored False;
    property OnClick: TNotifyEvent read FOnClick write FOnClick;
    property OnDblClick: TNotifyEvent read FOnDblClick write FOnDblClick;
    property OnKeyPress: TCardXOnKeyPress read FOnKeyPress write
FOnKeyPress;
  end;

procedure Register;

implementation

uses ComObj;

procedure TCardX.InitControlData;
const
  CEventDispIDs: array [0..2] of DWORD = (
    $00000001, $00000002, $00000007);
  CControlData: TControlData2 = (
    ClassID: '{7B33D945-0A2C-11D2-AE5C-04640BC10000}';
    EventIID: '{7B33D943-0A2C-11D2-AE5C-04640BC10000}';
    EventCount: 3;
    EventDispIDs: @CEventDispIDs;
```

```
    LicenseKey: nil (*HR:$00000000*);
    Flags: $00000009;
    Version: 401);
begin
  ControlData := @CControlData;
  TControlData2(CControlData).FirstEventOfs := Cardinal(@@FOnClick) -
Cardinal(Self);
end;

procedure TCardX.CreateControl;

  procedure DoCreate;
  begin
    FIntf := IUnknown(OleObject) as ICardX;
  end;

begin
  if FIntf = nil then DoCreate;
end;

function TCardX.GetControlInterface: ICardX;
begin
  CreateControl;
  Result := FIntf;
end;

procedure TCardX.FlipChildren(AllLevels: WordBool);
begin
  DefaultInterface.FlipChildren(AllLevels);
end;

function  TCardX.DrawTextBiDiModeFlags(Flags: Integer): Integer;
begin
  Result := DefaultInterface.DrawTextBiDiModeFlags(Flags);
end;

function  TCardX.DrawTextBiDiModeFlagsReadingOnly: Integer;
begin
  Result := DefaultInterface.DrawTextBiDiModeFlagsReadingOnly;
```

```
end;

function  TCardX.GetControlsAlignment: TxAlignment;
begin
  Result := DefaultInterface.GetControlsAlignment;
end;

procedure TCardX.InitiateAction;
begin
  DefaultInterface.InitiateAction;
end;

function  TCardX.IsRightToLeft: WordBool;
begin
  Result := DefaultInterface.IsRightToLeft;
end;

function  TCardX.UseRightToLeftAlignment: WordBool;
begin
  Result := DefaultInterface.UseRightToLeftAlignment;
end;

function  TCardX.UseRightToLeftReading: WordBool;
begin
  Result := DefaultInterface.UseRightToLeftReading;
end;

function  TCardX.UseRightToLeftScrollBar: WordBool;
begin
  Result := DefaultInterface.UseRightToLeftScrollBar;
end;

function  TCardX.ClassNameIs(const Name: WideString): WordBool;
begin
  Result := DefaultInterface.ClassNameIs(Name);
end;

procedure TCardX.AboutBox;
begin
```

```
  DefaultInterface.AboutBox;
end;

procedure Register;
begin
  RegisterComponents('ActiveX',[TCardX]);
end;

 end.
```

래퍼 파일이 만들어지는 과정

우선, 앞 예제의 파일 이름이 _TLB로 끝난다는 점을 주목한 독자가 있을 것이다. 좀 더 날카로운 독자라면 생성된 소스 코드 안에서 "라이브러리"를 참조하는 부분이 많다는 사실도 알아챘을 것이다. 래퍼 파일은 컨트롤의 타입 라이브러리(type library)로부터 만들어진다. ActiveX 컨트롤의 타입 라이브러리는 그 ActiveX 컨트롤을 구성하고 있는 여러 요소들에 대한 정보가 모여 있는 것인데, 시스템 레지스트리를 통해서 ActiveX 컨트롤과 연결된다. 특히, 타입 라이브러리에는 컨트롤이 지원하는 인터페이스와 컨트롤의 속성, 메소드, 이벤트, 컨트롤이 사용하는 열거형 등 프로그래밍에 필수적인 정보들이 모여 있다. 래퍼 파일의 첫 번째 항목은 컨트롤의 타입 라이브러리의 GUID이다.



타입 라이브러리는 ActiveX 컨트롤만을 위한 것은 아니다. 자동화 객체라면 모두 타입 라이브러리를 사용한다. 타입 라이브러리와 그의 활용에 대해서는 23장에서 좀 더 자세히 이야기하겠다.

열거형

생성된 래퍼 유닛 코드의 타입 라이브러리 GUID 부분 바로 아래를 보면 컨트롤이 사용하는 열거형이 선언되어 있다. 그런데 그 열거형들은 정식 열거형으로 선언되지 않고, 그냥 상수로 선언되어 있음을 알 수 있다. 타입 라이브러리의 열거형은 0부터 시작하지 않을 수도 있고, 또 숫자가 순서대로 이어지지도 않는 경우가 있다. C 언어에서는 그런 열거형을 선언하는 것이 허용되지만, 델파이에서는 그렇지 않으므로 위의 코드처럼 형만 선언하고, 나머지는 상수로 선언하는 것이다.

컨트롤 인터페이스

열거형 상수 선언 다음 부분에는 컨트롤의 기본 인터페이스가 선언되어 있다. 여기에서 ActiveX 컨트롤의 모든 속성과 메소드들이 나타난다. 속성들은 dispinterface로도 다시 선언되는데, 이는 컨트롤을 이중 인터페이스(dual interface)로 사용할 수 있게 하기 위한 것이다. 이벤트들은 dispinterface 부분 다음에 개별적으로 선언되어 있다. 인터페이스에 대해 몰라도 애플리케이션에서 ActiveX 컨트롤을 사용하는 데에는 아무런 어려움이 없으며, 사실 좀 어려운 주제이므로 이번 장에서는 더 이상 자세히 다루지 않겠다. 인터페이스에 대한 전반적인 내용은 23장에 나오며, ActiveX 컨트롤의 인터페이스에 대해서는 25장에 나온다.

TOleControl의 상속

그 다음 부분은 컨트롤 래퍼를 위한 클래스 정의이다. 기본적으로, ActiveX 컨트롤 래퍼 객체의 이름은 TXX 형태이다. 여기서 XX는 타입 라이브러리에 있는 컨트롤의 coclass 이름이다. 래퍼 객체는 TOleControl 클래스로부터 상속받은 객체이다. TOleControl 클래스는 TWinControl을 상속받기 때문에 윈도우 핸들을 가진다. TOleControl은 ActiveX 컨트롤의 기능을 델파이 컴포넌트로 연결하는 데 필요한 복잡한 처리를 캡슐화하는 클래스인데, 이 클래스 덕분에 ActiveX 컨트롤을 델파이 IDE 안에서 델파이 컴포넌트와 같은 방식으로 사용할 수 있는 것이다. TOleControl은 추상 클래스이다. 즉, TOleClass의 인스턴스를 직접 생성할 수는 없으며, 오직 다른 클래스에게 상속하는 용도로만 쓰인다.

메소드들

처음으로 나타나는 프로시저는 InitControlData() 프로시저이다. 이 프로시저는 TOleControl 객체의 InitControlData()를 재정의한 것이다. 이 프로시저는 고유한 OLE 클래스와 이벤트 식별 번호, 그리고 기타 컨트롤 관련 정보를 설정한다. 클래스 ID나 컨트롤의 잡다한 플래그들, 그리고 컨트롤의 사용 권한에 관련된 라이센스 키 등 ActiveX 컨트롤의 중요한 여러 정보들이 여기서 설정된다. 이 프로시저는 클래스 내부에서만 필요하고, 클래스 외부에서는 필요하지 않으므로 클래스 선언의 protected 부분에서 선언되어 있다.

CreateControl() 메소드는 private 인터페이스 필드인 FIntf를 컨트롤의 ICardX 인터페이스로 설정한다.

이 CardX ActiveX 컨트롤은 AboutBox()라는 메소드도 공개한다. 대부분의 ActiveX 컨트롤들, 특히 써드파티 제품들은 About 상자를 제공한다(개발 환경에서 ActiveX 컨트롤을 마우스 오른쪽 버튼으로 클릭하면 나오는 About 메뉴로 확인할 수 있다). CardX ActiveX 컨트롤 역시 About 상자를 제공한다. 이 메소드는 vTable 인터페이스(ControlInterface 속성을 이용)를 통해 호출된다.



ControlInterface 속성을 이용한 vTable 호출과 더불어, IOleControl의 OleObject 속성을 이용한 자동화(OLE Automation) 호출도 가능하다. 그러나 일반적으로, vTable을 통한 호출이 자동화를 통한 호출보다 더 효율적이다. 이에 대해서는 23장에서 다시 이야기하겠다.

속성들

TCardX 클래스의 속성들은 두 부류로 나누어져 있다. 한 부류는 read나 write 메소드가 지정되지 않은 것들이다. 이들은 TWinControl과 TComponent 클래스로부터 상속받은 표준 델파이 컴포넌트 속성들이다. 또 한 부류는 Get이나 Set으로 시작하는 read, write 메소드가 지정되어 있는 것들이다. 그 속성들은 IOleControl을 이용해서 캡슐화되는 ActiveX 컨트롤 자체의 속성들이다.

Get, Set 메소드들은 ActiveX 컨트롤 속성들과 델파이 컴포넌트 속성들을 연결시키는 역할을 한다. 델파이의 여러 데이터형들에 해당하는 Get과 Set 메소드들이 지정되어 있음을 주목하기 바란다(GetSmallIntProp(), SetSmallIntProp(), GetTColorProp(), SetTColorProp() 등등). 그런데 각각의 형들마다 Get과 Set 메소드들이 있지만, 그것들의 작동 방식은 모두 동일하다. X라고 하는 형에 대한 TOleControl의 Get_, Set_ 메소드들은 대부분 다음과 같은 형태를 가지고 있다.

```
function TOleControl.Get_XProp(Index: Integer): X;
var
  Temp: TVarData;

begin
  GetProperty(Index, Temp);
  Result := Temp.V X;
end;

procedure TOleControl.Set_XProp(Index: Integer; Value: X);
var
  Temp: TVarData;
begin
  Temp.VType := var X;
  Temp.V X := Value;
  SetProperty(Index, Temp);
end;
```

위의 코드에서 Index 인자는 속성의 색인 번호(TCardsCtrl 컴포넌트에서 속성들을 선언할 때 index 지시자로 지정한 값)인데, 내부적으로 프로시저에게 전달된다. 넘겨받은 인자 Value는 TVarData(Variant를 담는 레코드) 레코드에 저장되며, 그 TVarData와 Index를 TOleControl의 GetProperty()나 SetProperty() 메소드에 넘긴다. ActiveX 컨트롤의 각 속성들마다 고유한 색인(index)이 있으며, 그 색인은 속성들을 식별하는 데 사용한다. 색인값과 TVarData 변수인 Temp를 넘겨 받은 GetProperty()나 SetProperty()는 OLE 자동화를 이용해서 ActiveX 컨트롤 안에 있는 속성값을 얻거나 설정한다. 다시 말하면, 색인을 이용해서 해당 속성을 찾고, 그 속성에 해당하는 TVarData 레코드에 속성을 저장하는 것이다.

다른 개발 도구들과 비교해 볼 때 델파이에서는 ActiveX 컨트롤 자체의 속성들도 쉽게 사용할 수 있을 뿐만 아니라, 표준적인 TWinControl의 속성이나 메소드들도 사용할 수 있다는 장점이 있다. 따라서, ActiveX 같은 핸들 기반 컨트롤을 사용할 수 있다는 차원을 넘어서 오브젝트 파스칼의 객체 지향적인 성격도 최대한 살릴 수 있다. 즉, 델파이 개발 환경 안에서 ActiveX 컨트롤을 상속한 새로운 컴포넌트도 만들 수 있다는 뜻이다.

애플리케이션에서 ActiveX 컨트롤을 사용하려면

ActiveX 컨트롤 래퍼를 컴포넌트 라이브러리에 연결시킨 후에는 ActiveX 컨트롤에 대해서 더 이상 신경쓰지 않아도 된다. 앞에서도 이야기했지만, 일단 컴포넌트 팔레트에 올라온 것이면 그냥 델파이 컴포넌트라고 생각하고 사용하면 되는 것이다. 그림 7.4는 폼 디자이너의 TCardX 컴포넌트를 마우스 오른쪽 버튼으로 클릭해서 팝업 메뉴를 띄운 모습이다. 객체 조사기(Object Inspector)에 TCardX의 속성들이 나와 있음을 주목하기 바란다.

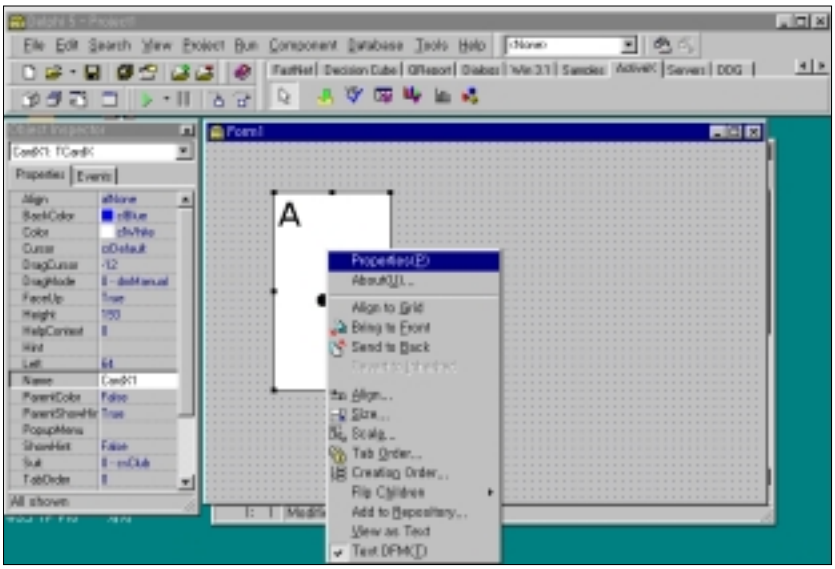


그림 7.4
델파이 개발 환경에서 액티브X 컨트롤을 사용

이미 델파이 컴포넌트가 된 이상 객체 조사기로 속성들을 설정할 수 있는 것은 당연하다. 그와 더불어서, 델파이는 ActiveX 컨트롤의 속성 페이지(property page, 속성을 쉽게 설정할 수 있는 대화상자. 이하 속성 대화상자)도 지원한다. 물론, 모든 ActiveX 컨트롤들이 속성 페이지를 제공하지는 않는다. TCardX는 속성 페이지를 지원하는데, 그림 7.4에서처럼 폼 디자이너 위의 컨트롤을 마우스 오른쪽 버튼으로 클릭하고 Properties를 선택하면 그림 7.5와 같은 속성 대화상자가 나타난다. ActiveX 컨트롤의 속성 대화상자 역시 델파이 컴포넌트의 속성 대화상자와 다를 것이 없다.



그림 7.5
CardX의 속성 대화상자

TCardX의 모습을 보면 짐작할 수 있을테지만, 이 컨트롤은 트럼프 카드의 종류나 숫자, 색, 카드 뒷면 그림 등을 지정할 수 있는 속성들을 가지고 있다. 또한, 컨트롤의 위치나 탭 순서 같은 표준적인 속성들도 가지고 있다. 그림 7.5에 나타난 카드는 Value 속성이 2이고, Suit 속성이 1(클로버)이다.

ActiveX 컨트롤을 사용하는 애플리케이션의 배포

다음은 ActiveX 컨트롤을 사용하는 애플리케이션을 배포할 때 주의해야 할 사항들이다.

- 애플리케이션에서 사용하는 ActiveX 컨트롤의 실제 파일인 OCX나 DLL 파일도 반드시 배포해야 한다. OCX 파일은 일종의 DLL 파일이며, 애플리케이션의 실행 파일 안에 링크되지 않는다. 또한, 사용자의 컴퓨터에서 애플리케이션이 제대로 실행되려면 먼저 ActiveX 컨트롤이 사용자 컴퓨터의 레지스트리에 등록되어야 한다. ActiveX 컨트롤의 등록에 대해서는 잠시 후에 이야기하겠다.

- 어떤 ActiveX 컨트롤들은 하나나 그 이상의 개별적인 DLL이나 기타 파일들이 있어야 작동을 한다. 사용하고자 하는 ActiveX 컨트롤의 도움말이나 문서를 참고해서 어떤 파일을 함께 배포해야 하는지 점검해야 한다. 델파이로 직접 작성한 ActiveX 컨트롤을 배포할 때 필요한 파일들에 대해서는 25장을 참고하기 바란다.
- ActiveX 컨트롤을 설계 시점에서 사용하려면 라이센스 파일이 필요한 경우가 많다. 라이선스 파일은 해당 ActiveX 컨트롤의 개발사로부터 얻는다. 라이센스 파일은 개발 권한을 가지지 않은 사용자가 함부로 ActiveX 컨트롤을 애플리케이션 개발에 사용하는 일을 방지하기 위한 것이다. 만일 ActiveX 컨트롤을 개발에도 사용할 수 있도록 배포하려면 라이센스 파일도 배포해야 되나, 라이센스 파일의 배포는 ActiveX 컨트롤의 개발사로부터 적절한 배포 권한을 얻은 경우에만 가능하다(개발사가 제공한 문서의 redistribution 항목을 참고하기 바란다).

ActiveX 컨트롤의 등록

ActiveX 컨트롤은 시스템 레지스트리에 등록된 이후에만 사용할 수 있다. 일반적으로, 윈도우즈에 기본적으로 포함되어 있는 RegSvr32.exe 프로그램을 통해 ActiveX 컨트롤을 등록한다. 델파이 사용자라면 bin 디렉토리에 있는 명령행 유틸리티 TRegSvr.exe를 사용할 수도 있다. 그러나 외부 프로그램을 사용하지 않는 방법도 있다. ActiveX 컨트롤의 등록(또는 해제)을 애플리케이션 안에서 수행할 수도 있는 것이다. 델파이 예제 중에는 TRegSvr이라는 유틸리티가 있는데, 그 프로그램의 소스 코드를 보면 ActiveX 서버나 타입 라이브러리를 등록하는 방법을 배울 수 있다.

OCX 예제 애플리케이션 "블랙잭"

백문이 불여일견이라고, ActiveX 컨트롤의 사용법을 배우려면 실제 애플리케이션을 만들어보는 것이 가장 좋다. 이번 예제는 TCardX ActiveX 컨트롤을 사용한다. 카드를 다루는 컨트롤인 만큼 규칙이 간단하면서도 재미 있는 블랙잭 게임을 만들어보자. 설명을 간단하게 하기 위해서 블랙잭이 어떤 게임인지는 알고 있다고 가정하겠다.

블랙잭 게임 프로그램에서 가장 중요한 부분은 물론 카드를 선택하고 승패를 판정하는 알고리즘이다. 그러나 일단은 ActiveX 컨트롤을 다루는 데 관련된 코드들을 부분적으로 설명하겠다. 이 프로젝트의 전체 코드는 설명이 끝난 다음에 나올 것이다. 이번 프로젝트의 이름은 BJ이다. 이 프로젝트의 실행 모습이 그림 7.6에 나와 있는데, 폼의 제목은 'DDG 블랙잭'으로 되어 있다. DDG는 물론 델파이 5 개발자 가이드의 약자이다.

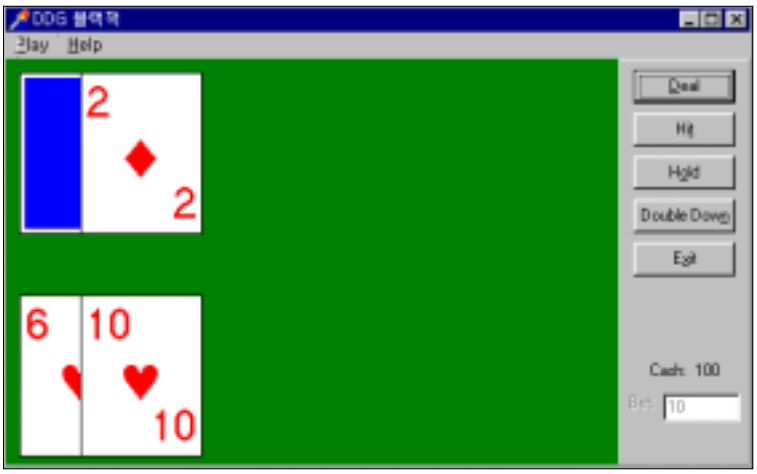


그림 7.6
BJ의 실행 모습

카드 덱

객체 지향적 방법론에 따라서 우선 게임을 구성하는 요소들을 객체화시켜 보자. 가장 먼저 떠오르는 것이 카드 덱(카드 더미)이다. 하나의 덱은 중복되지 않는 52장의 카드로 되어 있으므로(조커는 쓰지 않겠다) 한 장의 카드를 뜻하는 객체의 배열로 카드 덱 객체를 만들면 된다. 물론, 카드를 초기화하고 섞거나, 한 장의 카드를 꺼내는 메소드들도 제공해야 할 것이다. 다음의 목록 7.2는 TCardDeck 객체가 담긴 PlayCard.Pas의 코드이다.

목록 7.2 PlayCard.pas 유닛

```
unit PlayCard;

interface

uses SysUtils, Cards;

type
 ECardError = class(Exception); // 범용 카드 예외

 TPlayingCard = record // 한 장의 카드를 뜻하는 레코드
```

```
    Face: TCardValue; // 카드의 번호
    Suit: TCardSuit; // 카드의 종류
 end;

{ 하나의 덱을 구성하는 52장의 카드 배열 }
 TCardArray = array[1..52] of TPlayingCard;

 { 다음은 중복되지 않는 52장의 카드로 구성된 하나의 덱을 뜻하는 객체이다. }
 { 덱에서 몇 장의 카드가 나갔는지를 기억하기 위한 속성도 제공한다. }
 TCardDeck = class
    private
    FCardArray: TCardArray;
    FTop: integer;
    procedure InitCards;
    function GetCount: integer;
 public
    property Count: integer read GetCount;
    constructor Create; virtual;
    procedure Shuffle;
    function Draw: TPlayingCard;
 end;

{ GetCardValue는 인자로 지정된 카드의 번호를 돌려준다. }
function GetCardValue(C: TPlayingCard): Integer;

implementation

function GetCardValue(C: TPlayingCard): Integer;
{ 카드의 번호를 돌려준다. }
begin
 Result := Ord(C.Face) + 1;
 if Result > 10 then Result := 10;
end;

procedure TCardDeck.InitCards;
{ 덱의 각 카드에 고유한 숫자/종류 쌍을 지정한다. }
var
 i: integer;
 AFace: TCardValue;
```

```
 ASuit: TCardSuit;
begin
 AFace := cvAce; // 에이스부터 시작한다.
 ASuit := csClub; // 클로버부터 시작한다.
 for i := 1 to 52 do // 루프를 돌려서 각각의 카드를 초기화한다.
 begin
     FCardArray[i].Face := AFace; // 카드 숫자를 설정
     FCardArray[i].Suit := ASuit; // 카드 종류를 설정
     if (i mod 4 = 0) and (i <> 52) then // 네 장의 카드마다
     inc(AFace); // 숫자를 하나 증가한다.
     if ASuit <> High(TCardSuit) then // 다음 종류로 넘어간다.
     inc(ASuit)
 else
     ASuit := Low(TCardSuit);
 end;
end;

constructor TCardDeck.Create;
{ TCardDeck 객체의 생성자 }
begin
 inherited Create;
 InitCards;
 Shuffle;
end;

function TCardDeck.GetCount: integer;
{ 아직 사용되지 않은 카드 매수를 돌려준다.}
begin
 Result := 52 - FTop;
end;

procedure TCardDeck.Shuffle;
{ 카드를 다시 섞고, 맨 위의 카드를 0으로 한다. }
var
 i: integer;
 RandCard: TPlayingCard;
 RandNum: integer;
begin
 for i := 1 to 52 do
```

```
 begin
      RandNum := Random(51) + 1; // 난수를 얻는다.
      RandCard := FCardArray[RandNum]; // 난수로 지정된 카드와
      FCardArray[RandNum] := FCardArray[i]; // 현재 카드의 위치를 교환한다.
      FCardArray[i] := RandCard;
 end;
 FTop := 0;
end;

function TCardDeck.Draw: TPlayingCard;
{ 덱에서 다음 카드를 꺼낸다. }
begin
 inc(FTop);
 if FTop = 53 then
 raise ECardError.Create('Deck is empty');
 Result := FCardArray[FTop];
end;

initialization
 Randomize; // 난수 발생을 초기화한다.
 end.
```

게임

블랙잭 게임은 두 플레이어(플레이어와 딜러)가 번갈아서 한 장씩의 카드를 덱에서 가져오는(이를 히트라고 한다) 방식으로 진행된다. 플레이어가 한 장의 카드를 가져오는 동작에 해당하는 것이 Hit() 프로시저이고, 딜러가 한 장의 카드를 가져오는 동작에 해당하는 것이 DealerHit()이다.

마지막으로, 게임이 끝나서 승패가 판정되면 카드를 다시 덱에 넣는데, 그에 해당하는 것이 FreeCards()이다. 이 세 프로시저 모두 TCardX 객체를 사용해서 카드를 화면에 표시한다.

Hit() 프로시저는 다른 프로시저와는 달리, 컴포넌트 팔레트에서 폼에 놓은 컨트롤을 사용하지 않고, 동적으로 컨트롤을 생성한다. 또한, TCardX의 인스턴스 변수를 사용하지 않고, with..do 구조를 사용해서 한번에 컨트롤을 생성하고 사용한다. 다음은 Hit() 프로시저의 코드이다.

```
procedure TMainForm.Hit;
{ 플레이어의 히트 }
begin
  CurCard := CardDeck.Draw; // 카드를 그린다.
  with TCardX.Create(Self) do // TCardX OCX를 생성
  begin
      Left := NextPlayerPos; // 위치를 설정한다.
      Top := PYPos;
      Suit := Ord(CurCard.Suit); // 카드 종류를 설정
      Value := Ord(CurCard.Face); // 카드 숫자를 설정
      Parent := Self; // 부모를 설정한다.
      Inc(NextPlayerPos, Width div 2); // 위치를 조정
      Update; // 카드를 표시한다.
  end;
  DblBtn.Enabled := False; // 히트를 하면 더블 다운을 할 수 없으므로(블랙잭 규칙),
                           // 더블 버튼을 비활성화시킨다.
  if CurCard.Face = cvAce then PAceFlag := True; // 에이스 카드이면 플래그를 설정.
  Inc(PlayerTotal, GetCardValue(CurCard)); // 총점을 갱신한다.
  PlayLbl.Caption := IntToStr(PlayerTotal); // 총점을 표시한다.
  if PlayerTotal > 21 then // 21점이 넘으면 진 것이다.
  begin
      ShowMessage('Busted!');
      ShowFirstCard;
      ShowWinner;
  end;
end;
```

이 프로시저는 CardDeck이라는 TCardDeck 객체에서 임의의 카드를 한 장 가져온다(CurCard). 그런 다음, TCardX ActiveX 컨트롤을 생성하고 속성들을 적절히 설정한다. NextPlayerPos는 다음 카드의 X좌표를 담는 변수이다. PYPos는 플레이어의 카드의 Y좌표가 담긴 상수이다. 그 두 값을 TCardX 컨트롤의 Left와 Top에 넣어서 카드의 위치를 설정한다. 그리고 CurCard의 Suit와 Face값을 TCardX 컨트롤의 Suit와 Face 속성에 각각 넣는다. 그리고 메인 폼(Self)을 Parent 속성에 넣는다. 다음 카드의 위치인 NextPlayerPos는 카드 한 장의 반만큼 증가시킨다. 그런 다음에는 플레이어 카드의 점수 총합을 뜻하는 PlayerTotal 변수를 갱신한다.

DealerHit() 프로시저도 Hit() 프로시저와 거의 같다. 다음은 DealerHit() 프로시저의 코드이다.

```
procedure TMainForm.DealerHit(CardVisible: Boolean);
{ 딜러의 히트 }
begin
  CurCard := CardDeck.Draw;          // 딜러가 카드 한 장을 가져온다.
  with TCardX.Create(Self) do        // ActiveX 컨트롤을 생성한다.
  begin
    Left := NextDealerPos;           // 폼에 카드를 놓는다.
    Top := DYPos;
    Suit := Ord(CurCard.Suit);       // 카드 종류를 설정한다.
    FaceUp := CardVisible;
    Value := Ord(CurCard.Face);      // 카드 숫자를 설정한다.
    Parent := Self;                 // ActiveX 컨트롤의 부모 객체를 설정한다.
    Inc(NextDealerPos, Width div 2);  // 다음 카드의 위치를 설정한다.
    Update;                          // 카드를 표시한다.
  end;
  if CurCard.Face = cvAce then DAceFlag := True;   // 에이스 카드이면 플래그를 설정
  Inc(DealerTotal, GetCardValue(CurCard));         // 총점을 갱신
  DealLbl.Caption := IntToStr(DealerTotal);        // 총점을 표시한다.
  if DealerTotal > 21 then                        // 21점이 넘으면 딜러의 패배
    ShowMessage('Dealer Busted!');
end;
```

이 메소드는 CardVisible이라는 인자를 넘겨 받는다. 이 인자는 딜러의 카드 앞면을 보여줄 것인지를 결정하는 Boolean값이다. 블랙잭 규칙에 의하면 첫 번째 카드는 앞면을 보여주지 않게 되어 있으므로 DealerHit()를 처음 호출할 때에는 False를 넘겨주어야 한다.

FreeCards() 프로시저는 메인 폼에 있는 모든 TCardX 컨트롤들을 제거한다. ActiveX 컨트롤을 생성할 때 인스턴스 변수를 사용하지 않았으므로 폼에 생성되어 있는 컨트롤들에 접근하려면 폼의 Controls 배열 속성을 이용해야 한다. 루프를 돌려서 Controls 배열의 각 컨트롤들을 검사해서 만일 TCardX 컨트롤이면 그 컨트롤의 Free 메소드를 호출해서 메모리로부터 제거한다. 단, 루프를 돌릴 때는 배열의 첨자가 큰 쪽부터 작은 쪽으로 돌려야 한다. 그렇게 하지 않으면 루프 안에서 배열의 첨자가 계속 변하기 때문에 에러가 날 수 있다. 다음은 FreeCardX 프로시저의 코드이다.

```
procedure TMainForm.FreeCards;
{ 화면에 있는 모든 TCardX 컨트롤들을 해제한다. }
var
  i: integer;
begin
  for i := ControlCount - 1 downto 0 do // 루프를 거꾸로 돌려야 한다.
      if Controls[i] is TCardX then
          Controls[i].Free;
end;
```

여기까지가 ActiveX 컨트롤을 다루는 코드들이다. 애플리케이션의 메인 유닛인 Main.pas의 전체 코드는 목록 7.3에 나와 있다.

목록 7.3 BJ 프로젝트의 Main.pas 유닛

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  OleCtrls, AxCard_TLB, Cards, PlayCard, StdCtrls, ExtCtrls, Menus;

type
  TMainForm = class(TForm)
    Panel1: TPanel;
    MainMenu1: TMainMenu;
    Play1: TMenuItem;
    Deal1: TMenuItem;
    Hit1: TMenuItem;
    Hold1: TMenuItem;
    DoubleDown1: TMenuItem;
    N1: TMenuItem;
    Close1: TMenuItem;
    Help1: TMenuItem;
    About1: TMenuItem;
    Panel2: TPanel;
```

```
    Label3: TLabel;
    CashLabel: TLabel;
    BetLabel: TLabel;
    HitBtn: TButton;
    DealBtn: TButton;
    HoldBtn: TButton;
    ExitBtn: TButton;
    BetEdit: TEdit;
    DblBtn: TButton;
    CheatPanel: TPanel;
    DealLbl: TLabel;
    PlayLbl: TLabel;
    Label4: TLabel;
    Label6: TLabel;
    Cheat1: TMenuItem;
    N2: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure Cheat1Click(Sender: TObject);
    procedure ExitBtnClick(Sender: TObject);
    procedure DblBtnClick(Sender: TObject);
    procedure DealBtnClick(Sender: TObject);
    procedure HitBtnClick(Sender: TObject);
    procedure HoldBtnClick(Sender: TObject);
  private
    CardDeck: TCardDeck;
    CurCard: TPlayingCard;
    NextPlayerPos: integer;
    NextDealerPos: integer;
    PlayerTotal: integer;
    DealerTotal: integer;
    PAceFlag: Boolean;
    DAceFlag: Boolean;
    PBJFlag: Boolean;
    DBJFlag: Boolean;
    DDFlag: Boolean;
    Procedure Deal;
    procedure DealerHit(CardVisible: Boolean);
```

```
    procedure DoubleDown;
    procedure EnableMoves(Enable: Boolean);
    procedure FreeCards;
    procedure Hit;
    procedure Hold;
    procedure ShowFirstCard;
    procedure ShowWinner;
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

uses AboutU;

const
  PYPos = 175;        // 플레이어 카드의 시작 Y좌표
  DYPos = 10;         // 딜러 카드의 시작 Y좌표

procedure TMainForm.FormCreate(Sender: TObject);
begin
  CardDeck := TCardDeck.Create;
end;

procedure TMainForm.FormDestroy(Sender: TObject);
begin
  CardDeck.Free;
end;

procedure TMainForm.About1Click(Sender: TObject);
{ About 상자를 만들고 띄운다. }
begin
  with TAboutBox.Create(Self) do
    try
      ShowModal;
    finally
```

```
      Free;
    end;
end;

procedure TMainForm.Cheat1Click(Sender: TObject);
begin
  Cheat1.Checked := not Cheat1.Checked;
  CheatPanel.Visible := Cheat1.Checked;
end;

procedure TMainForm.ExitBtnClick(Sender: TObject);
begin
  Close;
end;

procedure TMainForm.DblBtnClick(Sender: TObject);
begin
  DoubleDown;
end;

procedure TMainForm.DealBtnClick(Sender: TObject);
begin
  Deal;
end;

procedure TMainForm.HitBtnClick(Sender: TObject);
begin
  Hit;
end;

procedure TMainForm.HoldBtnClick(Sender: TObject);
begin
  Hold;
end;

procedure TMainForm.Deal;
{ 딜러와 플레이어에게 카드를 나누어 준다. }
```

```
begin
  FreeCards;                        // 화면에서 카드들을 모두 제거한다.
  BetEdit.Enabled := False;         // 베팅 에디트 컨트롤을 비활성화
  BetLabel.Enabled := False;        // 베팅 레이블을 비활성화
  if CardDeck.Count < 11 then       // 덱의 카드가 11보다 적으면 다시 섞는다.
  begin
    Panel1.Caption := 'Reshuffling and dealing...';
    CardDeck.Shuffle;
  end
  else
    Panel1.Caption := 'Dealing...';
  Panel1.Show;                      // "딜링" 패널을 표시한다.
  Panel1.Update;                    // 확실하게 표시되도록 한다.
  NextPlayerPos := 10;              // 카드들의 X좌표를 설정
  NextDealerPos := 10;
  PlayerTotal := 0;                 // 카드 총점을 초기화
  DealerTotal := 0;
  PAceFlag := False;                // 플래그들을 초기화
  DAceFlag := False;
  PBJFlag := False;
  DBJFlag := False;
  DDFlag := False;
  Hit;                              // 플레이어의 히트
  DealerHit(False);                 // 딜러의 히트
  Hit;                              // 플레이어의 히트
  DealerHit(True);                  // 딜러의 히트
  Panel1.Hide;                      // 패널을 숨긴다.
  if (PlayerTotal = 11) and PAceFlag then
    PBJFlag := True;                // 플레이어가 블랙잭을 했는지 판단
  if (DealerTotal = 11) and DAceFlag then
    DBJFlag := True;                // 딜러가 블랙잭을 했는지 판단
  if PBJFlag or DBJFlag then        // 블랙잭이 나왔으면
  begin
    ShowFirstCard;                  // 딜러의 카드를 뒤집는다.
    ShowMessage('Blackjack!');
    ShowWinner;                     // 승패 판정
  end
  else
    EnableMoves(True);              // 버튼과 메뉴 항목들을 활성화
```

```
end;

procedure TMainForm.DealerHit(CardVisible: Boolean);
{ 딜러의 히트 }
begin
  CurCard := CardDeck.Draw;          // 딜러가 카드 한 장을 가져온다.
  with TCardX.Create(Self) do        // ActiveX 컨트롤을 생성한다.
  begin
    Left := NextDealerPos;           // 폼에 카드를 놓는다.
    Top := DYPos;
    Suit := Ord(CurCard.Suit);       // 카드 종류를 설정한다.
    FaceUp := CardVisible;
    Value := Ord(CurCard.Face);      // 카드 숫자를 설정한다.
    Parent := Self;                 // ActiveX 컨트롤의 부모 객체를 설정한다.
    Inc(NextDealerPos, Width div 2);  // 다음 카드의 위치를 설정한다.
    Update;                          // 카드를 표시한다.
  end;
  if CurCard.Face = cvAce then DAceFlag := True;   // 에이스 카드이면 플래그를 설정
  Inc(DealerTotal, GetCardValue(CurCard));         // 총점을 갱신
  DealLbl.Caption := IntToStr(DealerTotal);        // 총점을 표시한다.
  if DealerTotal > 21 then                        // 21점이 넘으면 딜러의 패배
    ShowMessage('Dealer Busted!');
end;

procedure TMainForm.DoubleDown;
{ 더블 다운 처리 }
begin
  DDFlag := True;               // 베팅을 조정하기 위해서 더블 다운 플래그를 설정
  Hit;                          // 한 장을 가져온다.
  Hold;                         // 딜러가 카드들을 가질 수 있도록 한다.
end;

procedure TMainForm.EnableMoves(Enable: Boolean);
{ 버튼과 메뉴 항목들을 적절히 활성화/비활성화 }
begin
  HitBtn.Enabled := Enable;             // Hit 버튼
  HoldBtn.Enabled := Enable;            // Hold 버튼
  DblBtn.Enabled := Enable;             // Double Down 버튼
  Hit1.Enabled := Enable;               // Hit 메뉴 항목
```

```
  Hold1.Enabled := Enable;              // Hold 메뉴 항목
  DoubleDown1.Enabled := Enable;        // Double 메뉴 항목
end;

procedure TMainForm.FreeCards;
{ 화면에 있는 모든 TCardX 컨트롤들을 해제한다. }
var
  i: integer;
begin
 for i := ControlCount - 1 downto 0 do // 루프를 거꾸로 돌려야 한다.
    if Controls[i] is TCardX then
      Controls[i].Free;
end;

procedure TMainForm.Hit;
{ 플레이어 히트 }
begin
 CurCard := CardDeck.Draw; // 카드를 그린다.
 with TCardX.Create(Self) do // TCardX OCX를 생성
 begin
     Left := NextPlayerPos; // 위치를 설정한다.
     Top := PYPos;
     Suit := Ord(CurCard.Suit); // 카드 종류를 설정
     Value := Ord(CurCard.Face); // 카드 숫자를 설정
     Parent := Self; // assign parent
     Inc(NextPlayerPos, Width div 2); // 위치를 조정
     Update; // 카드를 표시한다.
 end;
 DblBtn.Enabled := False; // 히트를 하면 더블 다운을 할 수 없으므로(블랙잭 규칙).
                      // 더블 버튼을 비활성화시킨다.
 if CurCard.Face = cvAce then PAceFlag := True; // 에이스 카드 소지 여부를 설정
 Inc(PlayerTotal, GetCardValue(CurCard)); // 총점을 갱신한다.
 PlayLbl.Caption := IntToStr(PlayerTotal); // 총점을 표시한다.
 if PlayerTotal > 21 then // 21점이 넘으면 진 것이다.
 begin
     ShowMessage('Busted!');
     ShowFirstCard;
     ShowWinner;
 end;
```

```
end;

procedure TMainForm.Hold;
{ 딜러의 선택을 처리하는 프로시저 }
begin
  EnableMoves(False);
  ShowFirstCard;                // 딜러의 카드를 표시한다.
  if PlayerTotal <= 21 then      // 딜러가 21점을 넘지 않았으면
  begin
    if DAceFlag then            // 딜러가 에이스를 가지고 있으면
    begin
      { 17점이 안되면 카드를 한 장 더 받는다. }
      while (DealerTotal <= 7) or ((DealerTotal >= 11) and (DealerTotal
              < 17)) do
        DealerHit(True);
    end
    else
      // 에이스가 없으면 17점이 될 때까지 카드들을 받는다.
      while DealerTotal < 17 do DealerHit(True);
  end;
  ShowWinner;                   // 승패 판정
end;

procedure TMainForm.ShowFirstCard;
var
  i: integer;
begin
  // 모든 카드들을 뒤집는다.
  for i := 0 to ControlCount - 1 do
    if Controls[i] is TCardX then
    begin
      TCardX(Controls[i]).FaceUp := True;
      TCardX(Controls[i]).Update;
    end;
end;

procedure TMainForm.ShowWinner;
{ 승패 판정 }
var
```

```
  S: string;
begin
  if DAceFlag then                    // 딜러가 에이스를 가지고 있으면
  begin
    if DealerTotal + 10 <= 21 then     // 최선의 수를 판단
      inc(DealerTotal, 10);
  end;
  if PACeFlag then                   // 플레이어가 에이스를 가지고 있으면
  begin
    if PlayerTotal + 10 <= 21 then    // 최선의 수를 판단
      inc(PlayerTotal, 10);
  end;
  if DealerTotal > 21 then           // 21점이 넘으면 총점을 0으로 설정
    DealerTotal := 0;
  if PlayerTotal > 21 then
    PlayerTotal := 0;
  if PlayerTotal > DealerTotal then   // 플레이어의 승리
  begin
    S := 'You win!';
    if DDFlag then                   // 더블 다운이면 베팅액의 두 배를 지불
      CashLabel.Caption := IntToStr(StrToInt(CashLabel.Caption) +
        StrToInt(BetEdit.Text) * 2)
    else                             // 보통이면 베팅액만큼만 지불
      CashLabel.Caption := IntToStr(StrToInt(CashLabel.Caption) +
        StrToInt(BetEdit.Text));
    if PBJFlag then                  // 블랙잭이면 1.5배를 지불
      CashLabel.Caption := IntToStr(StrToInt(CashLabel.Caption) +
        StrToInt(BetEdit.Text) div 2)
  end
  else if DealerTotal > PlayerTotal then   // 딜러의 승리
  begin
    S := 'Dealer wins!';
    if DDFlag then                   // 더블 다운이면 베팅액의 2배를 잃음.
      CashLabel.Caption := IntToStr(StrToInt(CashLabel.Caption) -
        StrToInt(BetEdit.Text) * 2)
    else                             // 보통이면 베팅액만 잃음.
      CashLabel.Caption := IntToStr(StrToInt(CashLabel.Caption) -
        StrToInt(BetEdit.Text));
  end
```

```
    else
      S := 'Push!';                    // 비김.
    if MessageDlg(S + #13#10'Do you want to play again with the same bet?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      Deal;
    BetEdit.Enabled := True;           // 베팅액을 변경할 수 있도록 함.
    BetLabel.Enabled := True;
  end;

  end.
```

ActiveX 컨트롤의 메소드 호출

목록 7.3을 보면 About 대화상자를 생성하고 표시하는 부분이 있다. 그림 7.7은 그렇게 해서 띄운 About 대화상자의 모습이다.



그림 7.7
BJ의 About 대화상자

이 About 대화상자에 있는 About Cards 버튼을 클릭하면 Cards ActiveX 컨트롤의 AboutBox() 메소드가 호출되어서 ActiveX 컨트롤의 About 대화상자가 나타난다.

다음은 Cards ActiveX 컨트롤의 About 대화상자를 띄우는 코드이다. 내부적으로는 vTable이 OLE 자동화 서버를 호출하는데, 그에 대해서는 23장에서 설명하겠다.

```
begin
  Card.AboutBox;
end;
```

요약

이번 장에서는 델파이 5 개발 환경 안에서 ActiveX 컨트롤을 사용하는 데 필요한 여러 중요한 개념이나 기법들을 소개했다. ActiveX 컨트롤을 델파이에 통합하는 방법과 오브젝트 파스칼 ActiveX 컨트롤 래퍼의 작동 방식, 그리고 ActiveX 컨트롤을 사용하는 애플리케이션을 배포할 때 주의해야 할 사항들이나 ActiveX 컨트롤의 등록에 대해 설명했으며, ActiveX 컨트롤을 사용하는 애플리케이션도 하나 만들어보았다. 양적인 면에서는 ActiveX 컨트롤의 시장이 델파이 컴포넌트 시장보다 훨씬 더 뛰어나다. 그러나 ActiveX 컨트롤은 델파이의 기능을 최대한 활용하지 못하므로 특별한 기능의 컴포넌트가 필요하다면 일단 델파이 VCL 컴포넌트부터 찾아보는 것이 좋다.