# Delphi 概説

高木英至

## 1 Delphi とは

### 1．1 Object Pascal としての Delphi

本章では Delphi（デルファイと読む）を使ったプログラミングを解説する。Delphi はボーランド(Borland)社が提供する Windows 用のプログラミング言語である。同社はＤＯＳ時代に Turbo Pascal というＰＣ用の Pascal コンパイラを安価に供給し、世界的に普及させた。特に研究者の間での普及率が高かった。Delphi はこの Turbo Pascal を Windows アプリケーション開発用に発展させた言語である。

Pascal はＣ言語や FORTRAN、BASIC、Java などと並ぶ汎用的なプログラミング言語の１つである。歴史的にいえば、アルゴリズムの記述を重視した言語である Algol から派生し、「構造化」（分かりやすいプログラムを書くこと）を実装して普及させた言語である。Algol から派生した経緯を考えれば、Ｃ言語とも同根であり、Ｃと Pascal の間の書換えは比較的に容易である。Pascal には、コードが直感的に把握しやすい、その厳密な形式性のために誤ったコードを書きにくい、それゆえに教育・研究用として優れている、などの利点がある。世界的に見れば 、社会心理学を含め、Pascal を用いる研究者人口は多い。著名な政治学者アクセルロッドもシミュレーションのプログラムを Pascal で記述している。

むろん Delphi は Turbo Pascal のような、単純な Pascal コンパイラではない。「Windows アプリケーション開発のための Object Pascal」というべきだろう。Object の語を付けるのは「オブジェクト指向」（後述）を備えているためである。また、Delphi は Windows アプリケーションを作るのに便利な、様々な機能を装備している。つまり、Delphi は業務用の巨大プログラムの開発に対応することを想定している。

本書の執筆段階における Delphi の最新ヴァージョンは Delphi 8（あるいは Delphi for the Microsoft .NET Framework）である。Delphi 8 は新たに .NET でのアプリケーション開発の機能を持つに至っている。本書では Delphi の最新ヴァージョンを前提に解説を書くことにする。ただし、本書が扱う範囲では、Delphi 8 は以前のヴァージョン（Delphi 7 など）と同じと考えて差し支えない。Delphi 8 を持っていない読者は、ボーランド社のサイト（http://www.borland.co.jp/）から１つ前のヴァージョン（Delphi 7）の試用版を無料でダウンロードして試すことができる。Delphi 7 と Delphi 8 では使い勝手に若干の相違があるけれども、相違点については解説を付すことにする。

### 1．2 Pascal のプログラム

Delphi も基本的には Pascal である。まず Pascal の大まかな概念を述べておこう。Pascal のプログラムは次のような構造を持つ。

```
program プログラム名;
宣言部
実行部
```

簡単な例がリスト１にある。このプログラムは整数の変数 x に１０を読み込み、２倍にして

表示するだけのプログラムである。最初の行はプログラムのヘッダであり、var で始まる２行目は変数 x を整数として宣言している。この行が宣言部にあたる。実行部とはその下の、begin と end. にはさまれた第４～８行である。Pascal ではこの実行部のステートメントを上から下に順次実行してゆく。

リスト１

```
program chap8_1;
var   x   : integer;
begin
      x := 10;
      writeln('x=', x);
      x := x * 2;
      writeln('x*2=',x:2);
      readln;
end.
```

元来の Pascal はこうしたプログラムのコードをエディタで書くことを想定するものだった。そのプログラムをコンパイルして実行するのである。プログラミング言語としての Pascal を解説した良い参考書は数が多く、リスト１のような形式のプログラム例を掲載している。

リスト１のプログラムは標準 Pascal で書かれており、そうした Pascal プログラムをそのまま使うことを Delphi は想定していない。ただし Delphi には標準 Pascal の範囲のプログラムをコンソールアプリケーションとして実行する機能が付いている。まずリスト１の内容のテキストファイルに拡張子 dpr （Delphi のプロジェクトファイル（後述）であることを指す）をつけて Delphi に読み込む。次にメニューの「プロジェクト」→「オプション」→「リンカ」を選択し、「コンソールアプリケーションの作成」にチェックを入れてから「実行」ボタンをクリックする。するとリスト１のプログラムが実行され、図１のように、Windows のデスクトップにおけるコンソールウィンドウに結果を表示してくれる。

しかし Delphi はこのようなコンソールアプリケーションのプログラムを作るために設計されたソフトではない。Windows の機能をフルに使えるように設計されたソフトである。したがって Delphi のプログラムは標準 Pascal とは異なった形式を備えることになる。

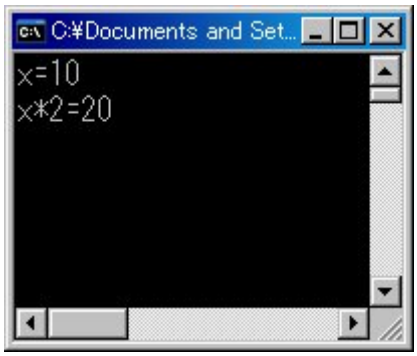


**図１：コンソールアプリケーションの実行画面**

プログラムのファイルも従来の Pascal とは異なってくる。例えば Turbo Pascal の場合であれば、１つのプログラムに対して Pascal で記述した１つのソースファイル（拡張子 .pas）があればよかった。しかし以下で見るように、機能を増した Delphi では１つのプログラムに対するファイルの数も増えることになる。

## １．３ Delphi を使う

まず Delphi を起動してみよう。Delphi 8 を起動して「新規作成」－「VCL フォームアプリケーション」を選択すると図２のような画面になる（画面の表示は設定したオプションによって異なる）。Delphi 7 までなら最初から図２に対応する画面が現れる。図２の中央に位置するのがフォームである。フォームとはプログラムの画面上での表現体とでも呼ぶべきもので、そのプログラムをコンパイルして実行したときにデスクトップに現れるのはこのフォームである。

図２の画面をデザイナ画面と呼ぶ。プログラマはデザイナ画面でフォームのデザインをする。例えばフォームの大きさを変えたり、フォームに Windows アプリケーションのためのコンポーネントを付加するといった作業を、グラフィカルにマウスなどで行うのである。図２の右側のツールパレットにはアプリケーション作成に便利な数々の部品、つまりコンポーネントが並んでいる。よく使うコンポーネントは、テキストを表示する memo や edit、図を描くための

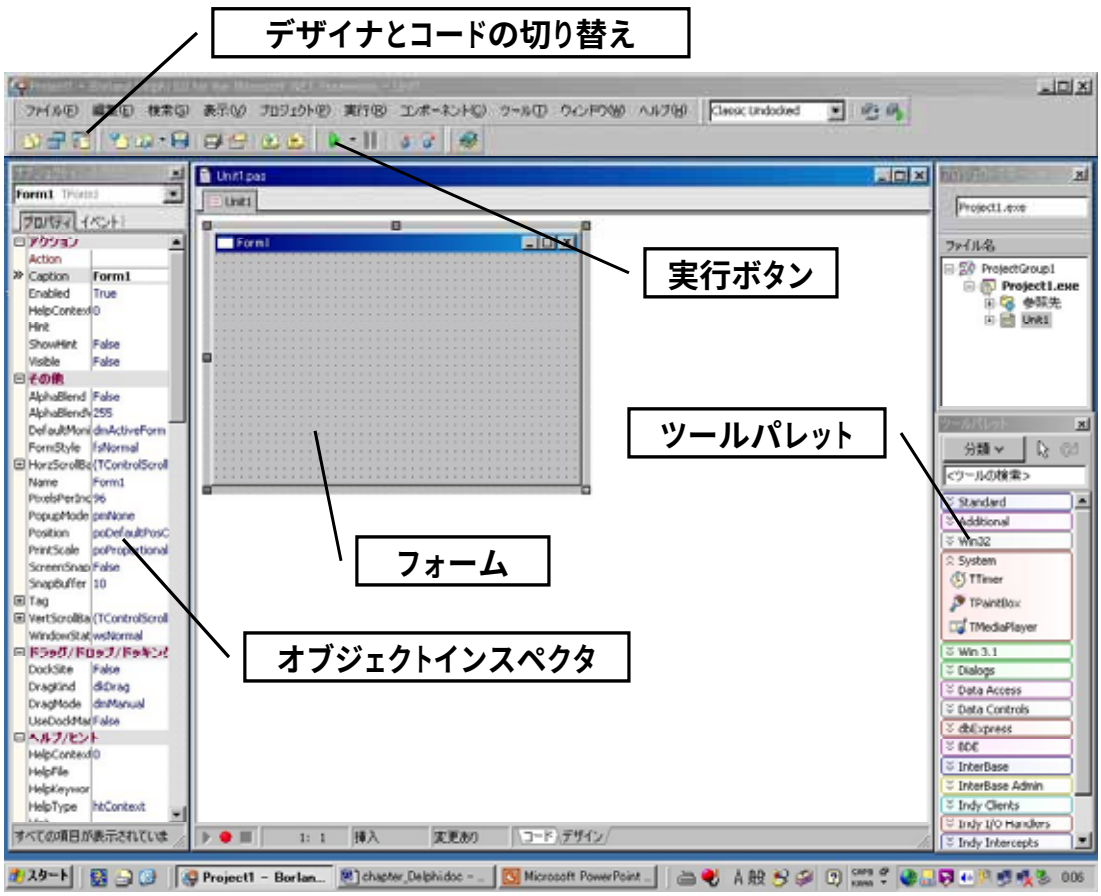


**図２：Delphi のデザイナ画面（新規作成時）**

PaintBox、実行を指定するボタン、タイマーなどである。これらのコンポーネントのいくつかをどう使うかは後に解説する。

図２の「デザイナとコードの切り替えボタン」（Delphi 7 ではフォーム／ユニットの切り替えボタン）を押すと図３のように中央にプログラムのコードが現れる。画面中央がコードのエディタになっており、ここでソースコードを書きながらプログラムすることになる。

リスト１のプログラム chap8_1 と同じ動作をするプログラムを Delphi で作成してみよう。まず、図１のコンソール画面を、テキスト文字を表示するためのメモ（memo）というコンポーネントで作ることを考えよう。そこで、図２のデザイナ画面でツールパレットの Standard に入っている Tmemo という項目をダブルクリックすると、memo コンポーネントがフォームの上に現れる。これでフォームにメモが付いたことになる。このメモにはデフォールトで memo1 という名前が付いている。デザイナ画面でフォーム上のメモをマウスでポイントすると、画面左側のオブジェクトインスペクタでこの memo1 のプロパティ（属性）を指定ないし変更することができる。ここは単に、memo1 に現れた'memo1'という文字列を削除するだけにしておこう。オブジェクトインスペクタのプロパティの中にある lines をクリックすると「文字列リストエディタ」が表示される。そこに書いてある'memo1'という文字列を削除すればよい。

次に、ツールパレットにある Tbutton という項目を同じようにダブルクリックし、ボタン（button）を２つ、フォームに付けてみる。２つのボタンにはそれぞれ、デフォールトで

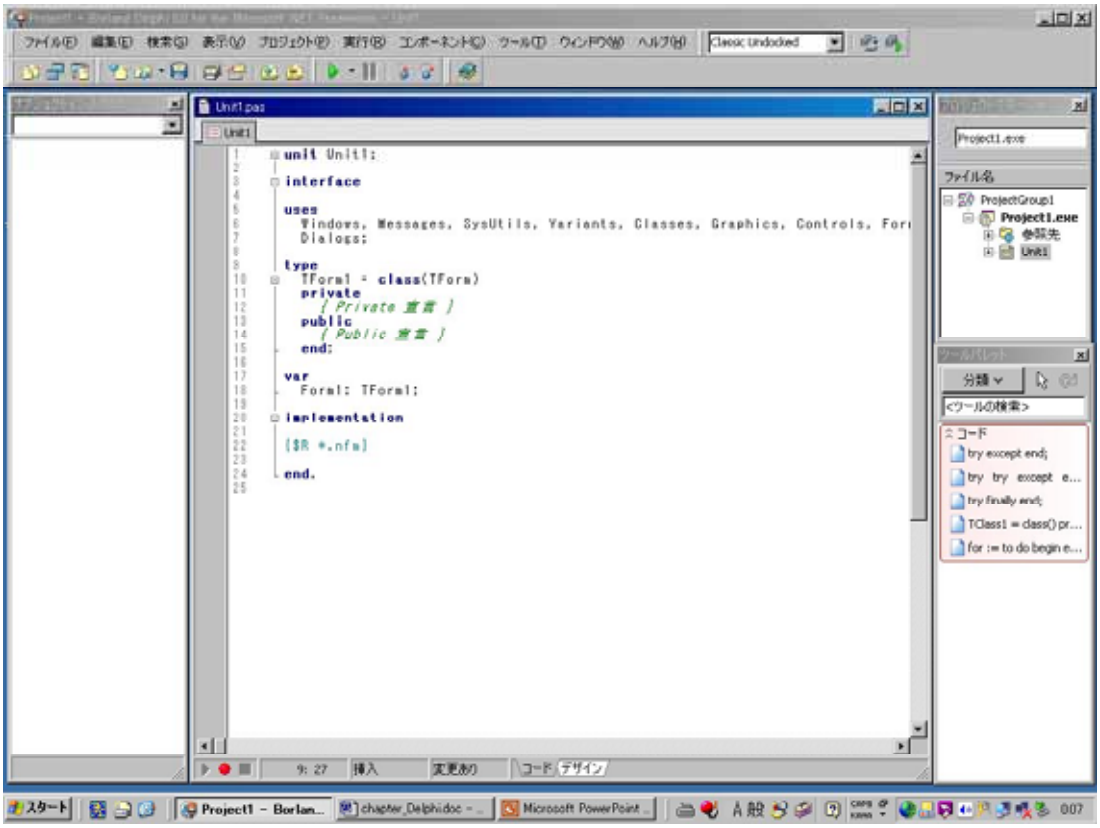


**図３：Delphi 8 のコード画面（新規作成時）**

buttun1、button2 という名前が付いている。名前を取り替えることができるけれども、いまはそのままにしておこう。それぞれのボタンをマウスでポイントしてオブジェクトインスペクタの Caption を、button1 は 'Start'、button2 は'End'にしてみる。このプログラムの動作を、Start ボタンをクリックすると作業が始まり、End ボタンをクリックするとプログラムが終了するようにデザインするためである。

ここでフォームにある Start ボタンをダブルクリックすると、画面が自動的にコード画面に切り替わり、Start ボタンをクリックするという「イベント」（後述）が生じたときに行う作業を記述する手続きの行（procedure TForm1.Button1Click(Sender: TObject);）が現れる。その箇所にリスト１のプログラムヘッダを除く部分に対応するコードを書けばよい。ただし書き込むコードは若干異なる。Delphi のメモへの表示は原則として文字列で行うからである。具体的には、後に見るリスト３の３０～３２行の変数定義を書いてみる。x を整数型の変数、s を（半角で）１０の文字を含み得る文字列の変数と定義するのである。その下の begin と end; の行の間に、リスト３の３４～３９行の６行を書いてみる。変数 s に代入した文字列（数字を表す）を form1.memo1.lines.add(s)で表示する。この Button1Click という手続きで、Start ボタンを押したときのプログラムの作業内容が書けたことになる。次に「デザイナとコードの切り替えボタン」でデザイナ画面に戻り、End ボタンをダブルクリックしてみる。するとまたコード画面に切り替わり、同様に End ボタンをクリックしたときの動作を決める手続きの行が現れる。その begin と end;の行の間に、プログラムの終了を宣言する close; という文を入れてみる。

これで chap8_1 と同じ動作をする Delphi のプログラム(chap8_2)ができたのである。メニューの実行ボタンをクリックするとプログラムが実行され、いま作ったフォームが画面に現れる（図４）。さらに Start ボタンをクリックすると memo の画面に計算結果が表示される。End ボタンをクリックするとプログラムは終了する。

ここまで読めば読者は、Delphi が Windows アプリケーションを開発する言語であることを実感できたのではないかと思う。例えば、Delphi に memo やファイル入出力のコンポーネントを貼り付ければ、簡単なテキストエディタならすぐに作ることができる。ツールパレットで TMediaPlayer と表記されるコンポーネントをつければ簡単なＣＤプレイヤーをすぐに作ることができる。そうしたアプリケーションの作り方は Delphi の参考書の中に見出すことがで

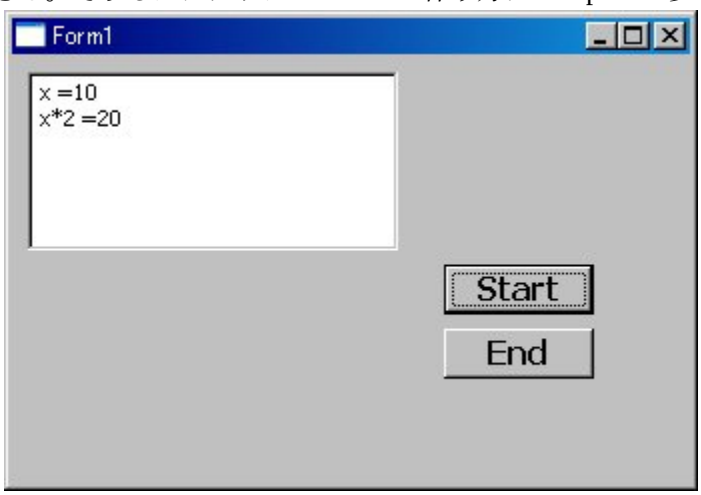


**図４：chap09_2 の実行結果**

きる。

## ２　Delphi の基本特性

この節では Delphi がどのような特性を持っているかを簡単に説明しよう。なお、ここに述べる Delphi の特性は Windows 用の言語であればほぼ共通に有するものであり、Delphi に固有という訳ではない。

### ２．１　統合開発環境

図２、３に見るようなグラフィカルなプログラミング環境を統合開発環境（IDE）と呼ぶ。統合開発環境のキーワードは２つである。第１は Rad(Rapid Application Development)、つまりアプリケーション開発の簡便性である。例えば chap8_2 程度の簡単なアプリケーションであっても、ボタンやメモを自前でプログラムしようとするとえらく面倒なことになる。統合開発環境はコンポーネントを貼り付けるなどの単純な操作で、この面倒な作業を代替しているのである。第２は 2-Way Tool、つまりデザイナ画面の作業とコード画面の作業の連携である。１．３で見たように、デザイナ画面で作業すればそれに対応するソースコードがある程度自動生成、自動変更される。つまりプログラマは、プログラムのソースコードのすべてを自分で書く必要はない。逆にソースコードの指定によってグラフィカルなフォームの形態を操作することもできる。

### ２．２　プロジェクトファイルとユニットファイル

Delphi のプログラムは複数のファイルで構成されている。それぞれが異なった拡張子を持つ。ファイルが何個になるかはプログラムの条件や Delphi のヴァージョンによって異なるけれども、Delphi 8 では１０個前後のファイルで１つのプログラムが成り立っている。ファイルの数が多いのは、異なった種類の設定を別々のファイルに書き込んでいるからである。

しかしファイルの多くは統合環境の中で自動生成されるため、プログラマが注意を向ける必要があるのはプロジェクトファイル（.dpr）とユニットファイル(.pas)の２つであり、プログラマが自分でソースコードを書くのは通常はユニットファイルだけである。なお最新ヴァージョンの Delphi 8 の統合開発環境では、プロジェクトファイルは拡張子.bdsproj のファイル名で表示される。

リスト２は上記の chap8_2 のプロジェクトファイル(chap8_2.dpr)のソースコードである（このリストでは見やすさのため、削除しても実行できる行は削除してある）。一見するとこのプロジェクトファイルは Pascal アプリケーションのメインプログラムであることがわかる。最初の行がプログラムヘッダであり、最後の５行がプログラムの実行部に当る。begin と end.の間のステートメントはアプリケーションの初期化・生成・実行を指示するメソッドである。ヘッダと実行部の間は宣言部に当り、アプリケーションで使用するユニットを宣言する uses 節などを含んでいる。この宣言部の中で、chap8_2a.pas のユニットファイルに収められたユニットを使うことを宣言している。

### リスト２

```
program chap8_2;

{%DelphiDotNetAssemblyCompiler 'c:¥borland¥common files¥borland shared¥bds¥sha
```

```
red assemblies¥2.0¥Borland.Vcl.dll'}

uses
  System.Reflection,
  System.Runtime.CompilerServices,
  SysUtils,
  Forms,
  chap8_2a in 'chap8_2a.pas' {Form1};

{$R *.res}

[STAThread]
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

元来の Pascal プログラムでは、メインプログラムを含むすべてのソースコードを拡張子 .pas のファイルに収めていた。Delphi ではメインプログラムをプロジェクトファイルとして記述し、他の Pascal のソースコードのほとんどをユニットファイル（.pas）に収めている。

プロジェクトファイルとユニットファイルが分離していることは、１つのプロジェクトファイルが複数のユニットファイルを扱えることを意味している。ここでは分かりやすさのために、１つのユニットファイルを使う場合だけを考えよう。なお、プロジェクトファイルは統合開発環境において自動生成されるため、プロジェクトファイルの中でソースコードを自分で書く必要はほとんどない。

リスト３はユニットファイル(chap8_2a.pas)の中身である。プロジェクトファイルとユニットファイルは拡張子の前のファイル名を同一にできないため、拡張子の前のファイル名を少し違えてある。

リスト３

```
unit chap8_2a;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Borland.Vcl.StdCtrls, System.ComponentModel;

type
```

```
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.nfm}

procedure TForm1.Button1Click(Sender: TObject);
var
       x : integer;
       s : string[10];
begin
      x := 10;
      s := 'x =' + IntToStr(x);
      form1.memo1.lines.add(s);
      x := x * 2;
      s := 'x*2 =' + IntToStr(x);
      form1.memo1.lines.add(s)
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
      close
end;

end.
```

unit で始まる最初の行がユニットヘッダである（unit ユニット名;）。ユニット名はユニ

ットファイル名の拡張子(.pas)の前の部分を表す。次の interface（行３）から implementation（行２５）の前までをインターフェース部と呼ぶ。インターフェース部ではこのユニットで使う他のユニットや変数・定数、手続き・関数（の中身のではなくヘッダ）の宣言、などを行う。

まず uses から始まる３行は、このユニットで呼び出す他のユニットの指定である。どれだけのユニットを呼び出す必要があるかはプログラムの作業内容によるし、Delphi のヴァージョンによっても異なる。統合開発環境を使えばこのユニットの指定部分は自動的に生成され、プログラマが自分で書く必要はない。

type（行９）、および TForm1 = class(TForm)から end;（行２０）の間で、このユニットで使うフォームのクラス（２．３で説明する）を定義している。フォームに付加した Memo1 と Button1、Button2 もコード上はここで宣言される。行１４、１５の procedure で始まる行１４、１５は、フォームにおいて定義される２つの手続きのヘッダの宣言である。インターフェース部で宣言するのは手続きのヘッダだけであり、手続きの具体的な内容は下の実現部で記述される。なお、'private' と 'public' はこれらの指定の可視性を宣言するのに使うが、簡単のためにここでは無視しよう。

var で始まる行２２、２３はこのユニットの全域で利用する（つまりユニット内ではグローバルな）変数を定義している。ここではこのプログラムで用いるフォーム（Form1）を定義しただけである。フォーム自体もコードでは変数として扱われることに注意を要する。また、例えばこのユニット全域で有効な a = 10 という定数を宣言するなら、この箇所に「const　a = 10;」という行を挿入すればよい。

implementation（行２５）と end.（最終行）までを実現部と呼ぶ。最初にある{$R *.nfm}はコンパイラ指令である。このコンパイラ指令も自動的に生成されるものであり、ここではスキップしよう。このコンパイラ指令を除くと、実現部はインターフェース部でヘッダを宣言した２つの手続きから成り立っている。これらの手続きは TForm1 というクラスのフォームにおいて定義されているため、実現部の手続き名には冒頭に'TForm1.'を付ける。ただし'TForm1.'の付いた手続きの内部で手続きを定義するときは、'TForm1.'は付けない。

このプログラム全体がどのように動くかは、実現部で記述した２つの手続きの中身で決まってくる。その中身については１．３で述べた通りである。

## ２．３　オブジェクト指向

Delphi は Object Pascal であり、現在使われている多くの開発言語と同様に、オブジェクト指向を備えた言語である。しかしオブジェクト指向そのものは抽象的かつ難解な議論であり、またオブジェクト指向性をフルに使うとすると長い解説を要する。ここではリスト３を説明するのに必要な事項に限定して解説しておこう。

大雑把にいえば、Delphi などのオブジェクト指向言語ではプログラミング上の要素を「オブジェクト」と捉える。実体的にはオブジェクトはデータの集まりであるが、視覚的にはフォームやコンポーネントも一種のオブジェクト（ないしその視覚的な表現体）である。

このオブジェクトの構造を指定する情報がクラス（ないしオブジェクト型）である。クラスはフィールド、メソッド、プロパティなどを備えており、クラスから定義されるオブジェクトもフィールド、メソッド、プロパティを有している。フィールドとはそのクラスで定義したオブジェクトで使う変数などを指す。メソッドとはオブジェクトで使う手続きや関数である。プロパティとはフィールドやメソッドにアクセスするための、オブジェクトの持つ様々な属性で

ある。実際上は、プロパティは Delphi の中で予め決められているクラス属性と思えばよい。Caption や Color がプロパティの例であり、Delphi ではオブジェクトインスペクタで視覚的に指定したりコードによって指定することができる。

同じクラスで定義した複数のオブジェクトは同じ構造を持つことになる。例えばリスト３の行１２、１３では Button1 と Button2 を TButton という同じクラスで定義している。つまりこの２つのボタンは同じ構造を持つオブジェクトである。にもかかわらずこの２つは同じ実体ではない。両者はともに Caption というプロパティを持つが、Button1 の Caption の中身は'Start'であり、Button2 の Caption は'End'である。ミケとタマが同じ猫でありながら別の猫であるのと同じである。

type で始まるリスト３の行９～２０は、このプログラムで用いるフォーム＝オブジェクト（Form1）のクラス(TForm1)を定義する箇所である。この箇所ではまず、TForm1 を TForm という「上位クラス」から派生させている。また、TForm1 を１つの Memo と２つの Button を持つものと定義する。さらに、ここでヘッダを宣言した２つの手続き（行１４、１５）が TForm1 が持つメソッドである。TForm1 というクラスのオブジェクトとして Form1 を宣言しているのが行２２、２３である。このプログラムの動作は、TForm1 で定義した Form1 を２つのメソッドを使うことによって決まることになる。

Delphi でコードを書くときにも基本的な命令文がクラスの制約を受けていることを理解する必要がある。例えば行３６の form1.memo1.lines.add という手続きは()内の文字列を memo に１行として追加する。この手続きは Form1 に付けた Memo1 というオブジェクトの lines（テキスト行）というプロパティで使える add というメソッドである。lines はさらに TString というクラスで定義され、その TString が add というメソッドを持っている。add のメソッドを持つプロパティを定義していないクラスのオブジェクトでは add のメソッドは使えない。同様に、例えば何らかのオブジェクトで描画をするときにはそのオブジェクトの Canvas というプロパティを使い、Canvas で定義されたメソッド（例えば線を引いたり円を描いたり）を使うことになる。

## ２．４　イベント駆動

Delphi の特徴の１つはプログラムの動作がイベントによって開始される点である。イベントの最も分かりやすい例は、chap8_2 で用いた、ボタンを「クリックする」(OnClick)というイベントである。

標準 Pascal に準拠した Pascal の教本を読めば、プログラムは begin と end.の行の間のメインプログラム内のステートメントを上から順に実行し、その間に出会う手続きや関数をその都度呼び出すように書かれているのが普通である。このようなプログラム書法をすることもむろん Delphi では可能である。メインプログラムにあたるコードをユニットファイルの FormCreate という手続きに書けばよい。メインプログラムで呼び出す関数や手続きは FormCreate 内部の関数、手続きとして定義すればよい。しかし Delphi では原則として、何らかのイベントによってそのイベントに対応する手続きが実行される。実は FormCreate という手続きも、フォームが作成されるというイベント(OnCreate)に対応した手続きである。

リスト３では、Form1 のメソッドとして定義した Button1Click と Button2Click の２つの手続きが書いてある。Start ボタンをクリックすれば手続き Button1Click が、End ボタンをクリックすれば Button2Click が実行されるのである。このようにあるイベントが生起したとき

に実行される手続きをイベントハンドラと呼ぶ。リスト３にあるように、イベントハンドラには引数（パラメータ）として Sender を入れる必要がある。パラメータ Sender は定められた上位クラス TObject で定義され、何がイベントを受け取ったかを伝える情報を含んでいる。

むろんユニットファイルの実現部にはイベントハンドラ以外の手続きも書くことができる。具体例は次章のサンプルを参照して欲しい。

## ３　文字列呈示のプログラム

それでは実験用のプログラムを Delphi で作ってみよう。ここでは第１章の最初に登場した、文字列呈示用の PowerPoint ファイル(char.ppt)と同じ動作をするプログラムを作ってみる。

### ３．１　プログラムのデザイン

char.ppt は次のような動作をするものだった。画面全体を黒くして中央に「Enter キーを押すと実験が始まります」という文字列を表示する。次に Enter キーを押すと一定の時間間隔で凝視点（+)、刺激語（ミカンなど)、空白（背景と同じ黒）が呈示される。最後に実験が終了したことを示す文字列を表示する。通常は、Esc キーを押せばプログラムは終了する。

この char.ppt と同じ動作をするプログラムの作り方にはいくつの方法があるだろう。ここでは次の方針をとってみる。まず、画面全体を黒くするためには、プログラムを実行して表示されるフォームを画面より大きく表示して、フォームの色を黒く塗ればよい。プログラムを起動したときにこのフォームが画面に現れ、最初の指示の文字列をフォームに書くのである。次に Enter キーを押すと刺激語などの呈示が開始するように、「キーを押す」というイベントに対応したイベントハンドラ手続きをフォームで定義する。さらにタイマー(Timer)という Delphi のコンポーネントを使い、一定時間が経つというイベントで駆動する手続きで刺激語などを表示すればよいだろう。

### ３．２　Delphi プログラム

このデザインにしたがって作った Delphi プログラムが付録ＣＤ－ＲＯＭに収録した chap8_3 である。chap8_3 は画面が 1024×768 であることを前提にしている。そのユニットファイル chap8_3a.pas の中身がリスト４である。まずインタフェース部ではフォームのクラスを指定している。このフォームに付けるコンポーネントは TTimer をクラスとする Timer1 というタイマーだけである。フォームのメソッドとして OnKeyDown、Timer1Timer、FormCreate の３つの手続きを用いる。

### リスト４

```
unit chap8_3a;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
```

```
  Dialogs, ExtCtrls;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure OnKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

  const
     pre_order = '121323123';//呈示刺激を指定する文字列
var
     Form1       : TForm1;
     iStep       : byte;//呈示のステップのカウンタ
     iStm        : byte;//刺激語のカウンタ
     n_present : byte;//呈示する刺激語数

implementation

{$R *.dfm}

procedure TForm1.OnKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);//キー入力で呼び出す手続き
begin
     if   Key = VK_ESCAPE then close;{Esc キーのとき}
     if (Key = VK_RETURN) and (iStep = 0) then{Enter キーのとき}
     begin
          timer1.enabled   := true;
          Form1.Canvas.Rectangle(Left, Top, Width, Height);
     end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);//タイマーで呼び出す手続き
  const
```

```
        s_focus     : string[6] = '    ＋    ';//凝視点
        s_blank     : string[6] = '          ';//空白
        s_stimuli : array[1..3] of string[6] = ('リンゴ','ミカン','ブドウ');
        s_ending    : array[1..2] of string[26]
                    = ('      実験が終了しました','実験者の指示をお待ち下さい');
  var
        p_ord         : byte;//呈示手順のＩＤ
        s_id          : byte;//刺激のＩＤ

begin
      timer1.enabled   := false;
      iStep := iStep + 1;//ステップカウンタの更新
      if iStep mod 3 = 1 then iStm := iStm + 1;//刺激語カウンタの更新
      p_ord := (iStep-1) mod 3;
      if iStm <= n_present then begin
          s_id   := StrToInt(copy(pre_order,iStm,1));
          with Canvas do begin
              Rectangle(0, 0, Width, Height);
              case p_ord of
                0: TextOut(350, 300, s_blank);
                1: TextOut(350, 300, s_focus);
                2: TextOut(350, 300, s_stimuli[s_id]);
              end; { of case }
          end;{of with Canvas }
      end; { of if iStm <= }
      if iStm > n_present then
          with Canvas do begin
              if p_ord = 1 then begin
                  Rectangle(0, 0, Width, Height);
                  Font.Size := 28;
                  TextOut(300, 350, s_ending[1]);
                  TextOut(300, 450, s_ending[2]);
              end
              else
                  TextOut(350, 300, s_blank);
          end;{of with Canvas }
      if (iStm <= n_present) or (p_ord = 0) then timer1.enabled   := true;
end;

procedure TForm1.FormCreate(Sender: TObject);//起動時に実行する手続き
```

```
begin
    Left := -5; Top := -5;//フォームの左端の位置
    Width := 1036; Height := 778;//フォームの縦横の大きさ
    with Canvas do begin//Canvas の使用開始
      Brush.Color := clBlack;//ブラシの色
      Rectangle(Left, Top, Width, Height);//矩形をブラシの色で塗る
      Pen.Color := clWhite;//ペンの色
      Font.Size := 36; Font.Style := [fsBold];//フォントのサイズとスタイル
      TextOut(100, 350, 'Enter キーを押すと実験が始まります');//文字列を表示
      Font.Size := 96;//フォントサイズの再定義
    end;
    iStep := 0; iStm := 0;//カウンタの初期設定
    n_present := Length(pre_order);//文字列定数 pre_order の文字数
    with timer1 do begin {タイマーの設定開始}
      enabled := false;//タイマーをオフにする
      interval := 1200;//タイマーの呼び出し間隔をミリ秒で指定
    end; { of with timer1 }//タイマーの設定終了
end;

end.
```

このプログラムを実行するとフォームの生成に対応する FormCreate の手続きが実行される。この手続きはまず、フォームを置く画面上の位置、大きさを指定する。Left、Top、Width、Length は何れもフォームのプロパティであり、これらはデザイナ画面のオブジェクトインスペクタでも指定することができる。例えば Left は正確には Form1.Left であるが、Form1 内部の作業なので Form1.を省略することができる。次に with Canvas do begin（行９０）と end;（行９７）までで、フォームの Canvas プロパティを使ってフォームにおける描画を行う。ブラシの色を指定し、そのブラシの色（黒）でフォームを塗りつぶす（フォームいっぱいに黒い矩形を描く）。次にペンの色を白に指定し、フォントを設定して教示の文字列を表示し、後に使うフォントサイズを指定する。ここに登場する変数や手続きは Canvas のプロパティ、メソッドである。例えば Brush.Color は正確には Canvas.Brush.Color であるが、with Canvas do の内部での作業であるため、Canvas.を省略している。この手続きではさらに、初期値の設定を行い（行９８、９９）、タイマーをオフにしてタイマーの呼び出し間隔を 1.2 秒に設定して終わる。

この FormCreate の実行が終わった段階では、画面は教示の文字を表示したまま静止している。ここで Enter キーを押すと、キーイベントが生じたことになり、手続き OnKeyDown が呼び出される。キーは Enter キーであり、iStep は初期設定のままのゼロであるから、タイマーがオンになり（行４０）、画面が再び塗りつぶされて（行４１）文字は消える。なお、OnKeyDown は Enter などの特殊なキーが押されたというイベントに反応する手続きである。通常の文字キーを押すときには手続き OnKeyDownPress を使う。

1.2 秒が経過するとタイマーイベントが生じる。そのときに呼び出されるのが手続き Timer1Timer である。この手続きの冒頭ではタイマーがオフになる（行５７）。次にカウンタの iStep の値は１増える。呈示する刺激語を指定する iStm も、iStep が３つ増えるごとに１増えることになる。

１つの刺激語の呈示は３つのステップからなっている。第１のステップで凝視点を表示し、第２ステップで刺激語を表示し、第３ステップで空白を表示する。この表示を指定するのが 0～2 の値をとる p_ord である（行６０）。刺激語のカウンタ iStm が呈示刺激を指定する文字列（pre_order）の範囲内であれば（行６１の if 文）、i_ord の値に応じて、case 文を用いて表示する文字列を選択する（行６５－６９）。呈示すべき刺激語を呈示し終えたとすれば（行７２の if 文）、刺激呈示後の教示を画面に表示する（行７７、７８）。そして、次のステップがある限りはタイマーをオンに戻す（行８３）。

すべての刺激語を呈示し終えた段階では、画面は教示文を表示したまま静止している。この段階でもう一度 Enter キーを押しても、iStep はゼロより大きくなっているので、プログラムは何もしない。しかし Esc キーを押せば close が実行され（行３７）、プログラムが終了する。

## ３．３　デザイナ画面での操作

chap8_3 を作るにあたっては、リスト４のソースコードでは表記していない若干の操作をデザイナ画面のオブジェクトインスペクタで以下のように行っている。プログラム実行後に指定を変更しないオブジェクトのプロパティは、オブジェクトインスペクタで指定した方が便利である。

まずフォーム Form1 のプロパティ BorderStyle を bsNone に指定した。フォームの最上部の Caption が表示される部分を消すためである。また、Windows のタスクバーの上にフォームを置くために、FormStyle を fsStayOnTop に指定している。さらにプロパティの Font で表示する文字のフォントをＭＳゴシック、文字の色を白に指定している。

また、イベントハンドラの手続きを定義するときにはオブジェクトインスペクタのイベントのページを使うと便利である。イベントのページの OnCreate というイベントの項の右側のセルに FormCreate と入力すると、下に begin と end;の行を持つ手続きの表示（行８６）がエディタの中に現れ、また TForm1 を定義する箇所にその手続きのヘッダが作成される（行１５）。同様に OnKeyDown の手続きはイベントのページの OnKeyDown の右のセルに OnKeyDown と入力すればエディタの中に作成される。Timer コンポーネントをフォームに付けた後にタイマーをポイントし、オブジェクトインスペクタのイベントのページの OnTimer の右のセルに Timer1Timer と入力すれば、この手続き行がエディタに作成される。また、chap8_3 では手続き FormCreate の中で描画を行っているので、同様にフォームの OnPaint というイベントの右のセルで FormCreate を指定しておく必要がある。

## ４　結び

本章では実験用プログラムとして chap8_3 を解説した。このプログラムは単純なプログラムであるけれど、拡張すればより機能の高いプログラムを Delphi で作ることはできる。

まず chap8_3 では、呈示する文字列や呈示順序、呈示数などをソースファイルの中に書き込んでいた。むろん Delphi を起動すれば呈示する文字列や呈示順序などを変更することもでき

る。しかし文字列や呈示順序を書いた別のテキストファイルを読み込んで表示するようにできれば設定の変更には便利である。付録に付けた chap8_4 はそのようなプログラムである。

また、社会心理学では短い文字列ではなく、センテンスを被験者に呈示して実験を行うことも多い。錯誤相関(illusory correlation)の実験での刺激呈示はその例である。錯誤相関の典型的な実験では、２つの集団の成員についての行動事例（センテンス）を被験者に呈示する。付録に付けた chap8_5 はこの錯誤相関の実験で使うことを想定したプログラムである。chap8_4 と同様に、呈示する刺激文は別のテキストファイルから読み込むので、刺激文の変更をすることが容易である。

本章では専ら、何らかの文字列を呈示するプログラムを例にしてきた。しかし Delphi は静止画像や動画、音声などを扱うこともできる。適切にプログラムすればネットワークを介した集団実験を制御するプログラムを開発することもできる。

Delphi によるプログラミングに興味を持った読者は次の３つの事項を学習することが望ましい。第１は特定の言語を学ぶ以前の、プログラミング言語に共通する基本的なコンセプトの学習である。プログラミング言語の参考書は知らず識らずに一定の基礎知識があることを前提にしていることが多いからである。第２は Pascal そのものの学習である。特定の言語ソフトの使い勝手は、Delphi を含め、ＯＳやプラットフォームやソフトのヴァージョンによって変わることがある。しかし Pascal そのものはある程度不変であり、アルゴリズムの学習にも Pascal は適している。第３は表題に Delphi をうたった Delphi の参考書を参照することである。巻末の資料で適当と思える参考書を紹介しておく。