Programación con Delphi (III)

© Francisco Charte Ojeda - http://www.fcharte.com

Sumario

Todos los elementos que manipulamos en el entorno de Delphi son objetos, en su mayor parte componentes, que cuentan con propiedades, métodos y eventos.

Cuando se comienza a trabajar con una nueva herramienta, el usuario generalmente quiere ver resultados de manera inmediata. Por ello en la entrega anterior, segunda de esta serie, nos centramos en el desarrollo de nuestro primer programa con Delphi, a pesar de que desconocemos la mayoría de las características de esta herramienta y sus componentes. En esta tercera entrega nos ocuparemos, con algo más de detalle, de lo que estamos utilizando y generando al trabajar con Delphi. Para ello nos serviremos del mismo programa de ejemplo de la entrega anterior, analizando su diseño, el código generado y funcionamiento.

Introducción

Delphi es una herramienta de desarrollo visual, lo que habitualmente se conoce como un RAD (*Rapid Application Development*, Desarrollo rápido de aplicaciones). Esto significa que ciertas tareas, como el diseño de las interfaces de usuario o la creación del modelo de acceso a datos, se efectúan de manera visual, arrastrando y soltando elementos conocidos como componentes. Si tan sólo conoce herramientas visuales, esta metodología le parecerá la más lógica. No obstante, hasta la aparición del concepto RAD, hace unos años, y aún hoy, existen muchas herramientas de desarrollo en las que crear una interfaz de usuario, por poner el ejemplo más simple, requiere un gran trabajo de codificación.

Las herramientas RAD disponen de componentes prefabricados, que efectúan una determinada labor, y asistentes que generan código automáticamente. Estos elementos hacen que crear una aplicación simple sea un juego de niños pero, a cambio, ocultan gran parte del funcionamiento del sistema y las propias aplicaciones, por lo que el desarrollador desconoce lo que está ocurriendo y, en ocasiones, incluso lo que él mismo está haciendo.

El lenguaje Object Pascal

Todos los componentes que incorpora Delphi están escritos en Object Pascal. Los asistentes de Delphi generan código en Object Pascal. Nuestras reglas de negocio, el código que tendremos que escribir para implementar la funcionalidad de nuestra aplicación, deberemos escribirlas usando Object Pascal. No hay duda, por tanto, de que se trata de un lenguaje que deberíamos conocer bien para poder aprovechar Delphi.

Describir el lenguaje Object Pascal, tan sólo básicamente, abarcaría algunos cientos de páginas de pura teoría. Una buena forma de aprender un lenguaje, sin comenzar con esa descripción, consiste en ver código, analizarlo y modificarlo. Es el sistema que todos hemos empleado, de niños, para aprender el idioma que hablamos: oyendo, repitiendo y analizando. Posteriormente, cuando sepamos defendernos con él, tendremos ocasión de conocer la sintaxis general y teórica.

Pascal es un lenguaje originalmente pensado para la educación, lo cual, indudablemente, influyó en su estilo, muy claro y fácil de entender, especialmente para los anglo-parlantes. En Pascal existen variables, constantes, bucles y funciones, como en cualquier otro lenguaje actual.

Los elementos esenciales del lenguaje no son muchos, más allá de las estructuras de control y manipulación de datos más o menos simples. El resto de las operaciones se efectúan con procedimientos y funciones, escritos en módulos de código independientes que es posible incluir en nuestros proyectos.

Clases y objetos

Como su propio nombre denota, Object Pascal es una versión del lenguaje Pascal original pero orientada a objetos. Esto significa que en Object Pascal es posible definir clases de objetos, que serían los moldes a partir de los cuales, posteriormente, se crearían copias de ese objeto. Existen multitud de clases predefinidas, disponibles para ser usadas por nosotros. Utilizando la clase `TButton`, por ejemplo, podemos crear tantos objetos de tipo botón como deseemos o necesitemos.

Hágase a la idea de que la clase es un molde de madera o metal, mientras que el objeto es lo que se obtiene al verter la escayola en ese molde. Lógicamente, puede usar el mismo molde para crear tantos objetos como quiera. Posteriormente, utilizando pinturas y otros tratamientos, puede diferenciar cada objeto de los demás, aunque, básicamente, todos ellos son similares puesto que han salido del mismo molde.

En Object Pascal las clases pueden heredar unas de otras, pueden derivarse clases nuevas de otras que ya existen. Suponga que tiene un molde ya hecho y que, partiendo de él, construye otro más complejo, que tiene los elementos del anterior más otros nuevos que ha añadido posteriormente. Básicamente, lo que ha hecho es derivar un molde de otro, heredando en el nuevo las características del antiguo. Esta técnica, que en Object Pascal se utiliza continuamente, permite crear clases muy complejas, derivando repetidamente unas de otras y dando lugar a lo que, habitualmente, se conoce como *jerarquías de clases*. La VCL (*Visual Component Library*, Librería de componentes visuales) de Delphi es una jerarquía de clases.

El Listado 1 corresponde a la definición de una clase simple en Object Pascal. La palabra `class` es la que identifica a esta porción de código, hasta encontrar la palabra `end`, como una definición de clase. La clase que está definiéndose se llamará `TBeepButton`, indicándose que ésta será una derivada de `TButton`, una clase ya existente. No se preocupe en este momento por comprender todos los elementos de ese código, lo importante es que le resulte familiar la construcción. En este caso concreto, lo único que se hace es añadir una propiedad, llamada `Beep`, de tipo `Boolean`, por lo que tan sólo podrá tomar los valores `True` y `False`.

Como iremos viendo en ésta y futuras entregas, hay clases predefinidas, clases que se generan mediante asistentes y otras que podemos escribir nosotros manualmente. Prácticamente todos los elementos que usamos en una aplicación Delphi: el formulario, los botones, cajas de texto, etc., son objetos, es decir, elementos obtenidos a partir de una definición de una clase.

Propiedades

En la analogía usada anteriormente, en que se comparaba una clase con un molde para escayola, decíamos que los objetos obtenidos a partir del molde, todos exactamente iguales, podían personalizarse de diferentes formas. Es posible, por ejemplo, pintar toda la figura de un cierto color, añadirle ornamentos, vestirla, etc.

Los objetos que se crean a partir de una clase, como `TBeepButton`, también son todos idénticos. Si esto quedase así, lógicamente, no serían objetos muy útiles. Imagine para qué puede servir la clase `TButton` si todos los botones tienen siempre las mismas dimensiones, el mismo título, tipo de letra,

etc. Los objetos creados a partir de clases, obviamente, también pueden ser personalizados, aunque usando elementos algo menos tangibles que la pintura o una prenda de vestir.

La personalización de un objeto se efectúa, en Delphi, a través de sus propiedades. Las propiedades son elementos expuestos por el componente a fin de que el programador pueda personalizarlo, estableciendo colores, dimensiones, títulos, enlaces a bases de datos, conexiones con servidores, etc. Al igual que otros elementos, las propiedades van heredándose de unas clases a otras. Nuestra clase `TBeepButton`, por tanto, contará con todas las propiedades que ya existían en `TButton`, aparte de la propiedad `Beep` propia.

Clases, objetos, componentes y controles

Son éstos cuatro términos que tienden a confundirse y utilizarse de manera indistinta, aunque realmente, en la terminología propia de los lenguajes orientados a objetos y, en particular, la de Delphi, no significan lo mismo.

La diferencia entre una clase y un objeto, tras las explicaciones dadas en esta entrega, han quedado bastante claras. Una clase es una definición, con la que se describen los miembros y características que tendrán los objetos. Un objeto es una copia, también llamada *instancia*, de una clase.

Todo elemento creado a partir de una clase es, por definición, un objeto. Éste puede estar compuesto sólo de métodos, disponer también de propiedades y quizá de eventos. Ninguna de las tres categorías de miembros, no obstante, es obligatoria.

Cuando un objeto dispone de los elementos necesarios para ser manipulado visualmente en un entorno de desarrollo, en una fase previa a la ejecución, entonces estamos hablando de un componente. Los componentes VCL, por ejemplo, pueden ser insertados en un contenedor con una operación de arrastrar y soltar, estableciendo sus propiedades y eventos también visualmente, mediante el Inspector de objetos. Si el objeto no es un componente, sus métodos, propiedades y eventos estarán accesibles tan sólo en ejecución, mediante código.

Los controles, por último, son aquellos componentes que cuentan con una interfaz de usuario. Son lo que podríamos llamar componentes visuales, ya que el usuario del programa podrá verlos y, en ocasiones, interactuar con ellos. En el ejemplo utilizado en la entrega previa, todos los componentes usados son controles. Existen, no obstante, componentes que efectúan otras tareas que no requieren contar con una parte de interfaz.

Desde nuestro punto de vista, el usuario de los objetos para crear aplicaciones, una propiedad se utiliza de manera similar a como se usa una variable cualquiera. Es decir, podemos leer el contenido de una propiedad, por ejemplo para saber las dimensiones de un botón, así como asignarle un valor, por ejemplo para modificar el título. Existen, no obstante, propiedades que no pueden ser escritas, tan sólo leídas.

Métodos

Además de propiedades, en las clases también se definen métodos, que actuarán sobre los objetos creados a partir de dicha clase. Sintácticamente hablando, un método es similar a una función o un procedimiento cualquiera, con la salvedad de que actúa no de forma general, sino sobre un cierto objeto de la clase a la que pertenece.

Suponga que su molde, anteriormente para figuras de escayola, se convierte en algo más avanzado, permitiéndole crear figuras que cuentan con diversos pulsadores y dispositivos electro-mecánicos o electrónicos. Al pulsar sobre el botón titulado `Rie`, el muñeco se ríe, y al hacerlo sobre el botón `Habla`, la figura dice algunas palabras. Esto ocurre, lógicamente, con el muñeco cuyos pulsadores está utilizando, es decir, no se ríe otro muñeco distinto a aquél sobre el que ha pulsado. De la misma forma, tampoco tiene sentido la existencia del botón `Rie` por sí solo, tiene que existir una figura que pueda reírse.

Los métodos de una clase son comparables a los pulsadores de las figuras. Como se ha dicho antes, se codifican y usan de manera similar a las funciones, pero a diferencia de éstas, los métodos no existen por sí solos, sino siempre asociados a un determinado objeto, sobre el que actuarán según proceda.

Al igual que las propiedades, y cualquier otro elemento perteneciente a una clase, los métodos se heredan de unas clases a otras. La clase `TBeepButton` del Listado 1, por tanto, dispondrá de todos los métodos que existieran en la clase `TButton`.

Eventos

La tercera categoría de miembros de una clase, mencionadas las propiedades y los métodos, la componen los eventos. Un evento es una señal generada por el objeto, ante un suceso concreto, y que nuestra aplicación puede aprovechar para ejecutar alguna acción. Dicho suceso puede ser externo, como la pulsación de una tecla en el teclado o el desplazamiento del ratón, o bien interno, como la recepción de una señal por parte del sistema operativo.

Para terminar con nuestros moldes y nuestras figuras, últimamente muy avanzadas tecnológicamente hablando, añadámosles un elemento más. Además de los pulsadores que actúan como métodos, haciendo que la figura hable o se ría a demanda del usuario, ahora vamos a añadir unos sensores de temperatura. De esta forma, cuando la temperatura sea inferior o supere una determinada marca, el muñeco emitirá directamente un mensaje, temblará o sudará. Estas acciones, que el usuario de la figura puede atender o ignorar, serían los eventos. Si somos considerados, abrigaremos o refrescaremos nuestro muñeco respondiendo al evento.

Los eventos que puede generar un objeto tienen nombre, al igual que las propiedades y métodos. No obstante, no puede llamárseles, ya que no están diseñados para ello. Nosotros podemos indicar al objeto que ejecute un determinado procedimiento cuando se produzca un cierto evento, atendiéndole como proceda. Al pulsar un botón del ratón sobre una ventana, por ejemplo, podríamos abrir un menú emergente.

En el programa utilizado como ejemplo en la entrega previa, aprovechábamos el evento `OnClick` del botón, que se produce al pulsarlo, para mostrar un mensaje en la pantalla. De forma análoga, podríamos controlar cada una de las pulsaciones de teclas sobre un campo de edición, la recepción de información por la red o la selección de un elemento en una lista.

Definición visual de clases

No todas las clases se definen escribiendo código, en ocasiones dicha definición es generada por Delphi a partir de las operaciones de diseño efectuadas por nosotros. Esto es lo que ocurre, por ejemplo, cuando usamos una ficha para definir el aspecto de nuestras ventanas. A medida que se incluyen componentes y gestionan eventos, Delphi genera automáticamente una definición de clase que describe a esa ficha.

Repitamos los pasos dados en la entrega previa a la hora de crear nuestra primera aplicación con Delphi. Comience por iniciar un nuevo proyecto. Pulse F12 para acceder al Editor de código, encontrará una definición de clase como la del Listado 2. Todos los formularios, el elemento visual que nosotros vemos como una ventana, son objetos de la clase `TForm` o alguna derivada de ésta. Cuando diseñamos un nuevo formulario, indirectamente estamos definiendo una nueva clase que, en principio, no añade nada a lo que hereda de `TForm`. En este momento, por tanto, tendríamos una clase: `TForm1`, idéntica funcionalmente a `TForm`.

Añada ahora la etiqueta de texto, el campo de edición y el botón, los tres elementos que había en la ventana, abriendo de nuevo al Editor de código. La definición de la clase ha cambiado, y ahora encuentra el código mostrado en el Listado 3. La clase `TForm1` cuenta con tres miembros que no existen en `TForm`. Estos miembros son tres variables, llamadas `Label1`, `Edit1` y `Button1`, cuyos tipos se indican detrás de los dos puntos, `TLabel`, `TEdit` y `TButton`, respectivamente. Cada uno de estos miembros es un objeto que, a su vez, cuentan con sus características propias.

Por último, haga doble clic sobre el botón para gestionar el evento `OnClick` y mostrar el mensaje. A la definición de la clase se añade un elemento más, la definición de un método en el que escribiremos el código que deberá ejecutarse cuando se genere el mencionado evento. Finalmente, la definición de la clase queda como se muestra en el Listado 4.

La clase `TForm1`, en este momento, ya no es igual a `TForm`. Hereda toda la funcionalidad de ésta pero, además, añade algunos elementos adicionales. Se ha creado una clase derivando de otra, `TForm` es la clase base y `TForm1` la derivada. Como ha podido ver, la definición de esta nueva clase se ha efectuado de manera visual, sin necesidad de escribir código.

El objeto `Application`

Llegados a este punto, en nuestro proyecto contamos con una nueva clase, llamada `Form1`, que contiene los elementos y el código asociado para que al pulsar el botón se muestre un mensaje de saludo. Sin embargo, en ningún punto creamos un objeto a partir de esa clase, ni indicamos a Windows que muestre la ventana para poder actuar sobre ella. Estas acciones, y otras muchas, quedan en manos del objeto `Application`.

Cada vez que se inicia un nuevo proyecto en Delphi, éste crea un módulo de código, no visible en principio, que contiene el código por donde comienza la ejecución de la aplicación. Puede acceder a dicho módulo mediante la opción `View Source` del menú `Project` de Delphi. En el caso de nuestro programa de ejemplo, el contenido de este módulo será similar al del Listado 5.

Este módulo, a diferencia del asociado al formulario, no comienza con la palabra `unit` en su cabecera, sino con la palabra `program`. Esto es lo que le diferencia de cualquier otro módulo, indicando que es el punto de inicio del programa. El código escrito entre las palabras `begin` y `end` de este módulo, por tanto, será lo que ejecute Windows al poner en marcha al programa.

Como puede ver en el Listado 5, el código a ejecutar se compone tan sólo de tres sentencias, en las que repetidamente se usa un objeto llamado `Application`. Éste es un objeto creado a partir de la clase `TApplication`, predefinida en la VCL. Los objetos de esta clase son capaces de controlar la ejecución de una aplicación. Por ello, como es lógico, tan sólo puede existir un objeto en cada proyecto. Lo que se hace es llamar a tres métodos de ese objeto, inicializando la aplicación, creando un objeto de la clase `TForm1` (nuestro formulario) y poniendo en marcha la visualización, recepción de mensajes y su procesamiento.

Visto y por ver

Esta tercera entrega, básicamente teórica, nos ha servido para comprender mejor cómo funcionan nuestros proyectos creados con Delphi. Sabemos que dichos proyectos cuentan con un objeto, creado a partir de la clase `TApplication`, que se encarga de las tareas de inicialización, crear los formularios y poner el programa en marcha. También sabemos que los formularios son objetos creados a partir de clases que, derivadas de `TForm`, se definen visualmente, a medida que insertamos componentes. Los componentes, que son objetos manipulables durante la fase de diseño, cuentan con propiedades, métodos y eventos, las tres categorías de miembros fundamentales.

Con la base adquirida hasta ahora, en futuras entregas nos centraremos en el trabajo con algunos componentes, conceptos de Object Pascal, manipulación de propiedades y gestión de eventos. Nuestro objetivo: diseñar aplicaciones con un mínimo de utilidad práctica.

```
TBeepBoton = class(TButton)
private
  FBeep: Boolean;
published
  property Beep: Boolean read FBeep write FBeep;
end;
```

Listado 1. Definición de la clase TBeepButton en Object Pascal.

```
TForm1 = class(TForm)
private
  { Private declarations }
public
  { Public declarations }
end;
```

Listado 2. Definición de clase correspondiente a un formulario vacío.

```
TForm1 = class(TForm)
  Label1: TLabel;
  Edit1: TEdit;
  Button1: TButton;
private
  { Private declarations }
public
  { Public declarations }
end;
```

Listado 3. Definición de la clase asociada al formulario tras añadir los componentes

```
TForm1 = class(TForm)
  Label1: TLabel;
  Edit1: TEdit;
  Button1: TButton;
  procedure Button1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

Listado 4. Versión final de la definición de la clase TForm1

```
program EmisorSaludos;

uses
  Forms,
  Formulario in 'Formulario.pas' {Form1};

{$R *.RES}

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

Listado 5. Código correspondiente al módulo de proyecto