Curso de Delphi (y VI)

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

Sumario

El proceso de depuración, uno más en la fase de desarrollo de una aplicación, precisa de herramientas eficientes que permitan identificar los posibles errores. Conoceremos algunas de esas herramientas en Delphi.

Mientras se desarrolla y ejecuta una aplicación generalmente surgen errores. Éstos impiden que la aplicación funcione correctamente o, al menos, que lo haga como esperábamos que lo hiciese. Los posibles errores pueden clasificarse en tres grupos: de compilación, de ejecución y lógicos. Los primeros, como su propio nombre indica, son identificados por el compilador en una fase previa a la ejecución. Son fácilmente corregibles al tratarse, en la mayoría de los casos, de errores de sintaxis y similares. Los errores de ejecución surgen de forma esporádica al ejecutar la aplicación, pudiendo llegar a la interrupción de ésta si no se controlan de manera adecuada. Se producen, por ejemplo, cuando intentamos abrir un archivo inexistente, se efectúa una conversión inválida o una división por cero.

La tercera categoría de errores, los que hemos denominado *lógicos*, no son detectados por la herramienta de desarrollo durante la compilación ni ejecución. Son errores no provocados por una codificación sintácticamente incorrecta pero, sin embargo, impiden que el programa funcione de forma normal. Errores de este tipo se producen cuando en una expresión se ha usado la variable que no correspondía, se invoca a un método con una parámetro que no se esperaba y casos similares.

El único método para detectar y encontrar este tipo de errores, aparte de la intuición propia de todo programador, consiste en llevar a cabo una sesión de depuración. En ésta se usan generalmente dos técnicas distintas pero combinadas entre sí: la ejecución del código sentencia a sentencia y el inspeccionamiento del contenido de objetos y variables. Hace unos años este proceso se efectuaba mediante una herramienta externa, un depurador, como podía ser Turbo Debugger. Actualmente, no obstante, la mayoría de herramientas de desarrollo integran funciones de depuración en el propio entorno y el editor de código.

Transición de los tres estados

Y no nos referimos a los estados físicos: sólido, líquido y gaseoso, sino a los estados posibles en un momento dado en el entorno de desarrollo de Delphi. Éstos son tres también: diseño, ejecución y pausa. El estado de diseño es el inicial, en el cual podemos modificar propiedades, escribir código y efectuar todas las tareas propias de diseño de la aplicación.

Al ejecutar un programa desde el propio entorno de desarrollo, pasamos al estado de ejecución. Se distingue del anterior porque en la barra de título de la ventana principal de Delphi aparece la palabra `Running`, como puede apreciarse en el detalle de la Figura 1. Al pasar a este modo también observaremos la activación de ciertos botones y opciones de menú, así como la desactivación de otros que estaban activos en el estado de diseño. El botón de ejecución, por ejemplo, se desactivará, al igual que los de ejecución paso a paso. Los botones de pausa y parada, por el contrario, se activarán.

El tercer estado posible es el de pausa. A éste se llega cuando, estando en modo de ejecución, pulsamos sobre el botón `Pause` o seleccionamos la opción equivalente del menú `Run`. Como se aprecia en el detalle de la Figura 2, la indicación `Running` que aparecía en la barra de título es sustituida por la palabra `Stopped`. Al igual que en el caso anterior, ciertos botones y opciones cambian de estado. En este momento pasarán a estar activos los de ejecución paso a paso.

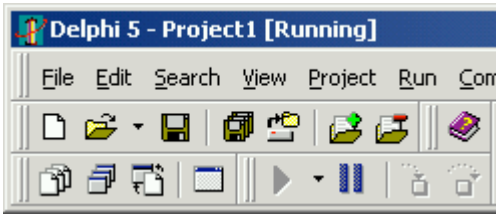

Figura 1. En el modo de ejecución aparece la indicación `Running` en la barra de título, al tiempo que se activa el botón de pausa y se desactivan los de ejecución completa y paso a paso.

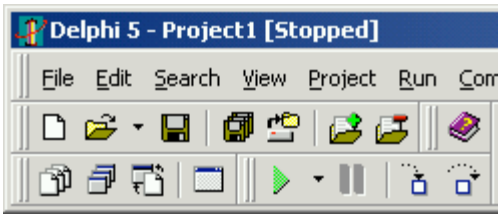

Figura 2. En el modo de pausa la indicación cambia de `Running` a `Stopped`, al tiempo que se invierte el estado de los botones de ejecución y pausa.

Al iniciar la ejecución de un programa desde el entorno de Delphi estamos, en realidad, ejecutando dos aplicaciones de forma paralela: nuestro propio programa y el depurador integrado de Delphi. En estado de ejecución el control lo tiene nuestro programa, mientras que en estado de pausa el control lo tiene el depurador. A medida que se va ejecutando paso a paso, con las opciones que veremos en el punto siguiente, el control pasa alternativamente de nuestro programa al depurador y viceversa. De esta forma es posible ir viendo el resultado que genera el programa poco a poco, al tiempo que inspeccionamos código, expresiones, objetos y variables. Es como si lo ejecutásemos a cámara lenta para poder encontrar el punto en el que falla.

Ejecutar paso a paso

Como se ha indicado en el punto anterior, una de las técnicas más conocidas de depuración consiste en la ejecución paso a paso del código. De esta forma es posible ver el resultado generado por la ejecución de cada sentencia de manera individual. Tras cada sentencia, además, es posible inspeccionar el contenido de los objetos y variables que estén implicados, haciendo más sencilla la localización del posible error.

En Delphi la ejecución paso a paso se efectúa en la propia ventana del Editor de código, es decir, no existe una ventana o herramienta separada para este trabajo. Es una ventaja, ya que al localizar los errores podemos corregirlos directamente, modificando el código según corresponda. Al iniciar una sesión de depuración el Editor de código resalta la sentencia que va a ejecutarse, distinguiéndola visualmente de las demás.

Generalmente la ejecución paso a paso parte del modo de pausa. A éste podemos llegar de diversos modos: detener la ejecución del programa manualmente cuando interese, ejecutando hasta una determinada sentencia, etc. Según el caso, bastará con pulsar el botón `Pause`, habrá que disponer un punto de parada o usar algún otro mecanismo que nos permita interrumpir la ejecución. En cualquier caso, estando ya en el modo de pausa podremos acceder a los comandos de depuración siguientes.

Al encontrarnos en modo de pausa en la ventana del Editor de código debe aparecer destacada sobre las demás la sentencia que se ejecutará en el paso siguiente. Si esto no es así, bastará con usar la opción `Show Execution Point` del menú `Run` para conseguirlo.

La sentencia destacada podemos ejecutarla usando dos opciones distintas. Con la opción `Trace Into`, el botón equivalente o la tecla F7, el control pasará temporalmente a nuestro programa y se procederá a ejecutar la sentencia. Si en ésta se efectúa una llamada a otro punto del programa, la ejecución paso a paso se transferirá a ese otro punto facilitándonos su depuración. Una alternativa consiste en usar la opción `Step Over` que, a diferencia de la anterior, ejecutará la sentencia destacada en un solo paso. Suponiendo que el punto de ejecución se encuentre en una sentencia que contiene una llamada a un procedimiento de nuestro programa, la opción `Step Over` ejecutará

directamente todo el código de ese procedimiento, de tal forma que el punto de ejecución pasará a la sentencia siguiente del mismo bloque en que nos encontrábamos.

Lanzar y detener la ejecución

Encontrándonos en el modo de pausa, ejecutando paso a paso nuestro programa, podemos volver al modo de ejecución en cualquier momento pulsando F9 o el botón correspondiente. El programa tomará el control y no lo devolverá a Delphi hasta que finalice o, de algún modo, interrumpamos su ejecución.

Otro comando muy útil en este contexto es el que nos permite transferir el control a nuestro programa, continuando con la ejecución, hasta llegar a una cierta sentencia. Esto permitiría, por ejemplo, ejecutar todos los ciclos de un bucle o, simplemente, iniciar la ejecución del programa desde el estado de diseño deteniéndola hasta llegar a una cierta sentencia. Para ejecutar hasta un determinado punto del programa tendremos que posicionar el cursor en la sentencia que nos interese, usando a continuación la opción `Run to Cursor` o la tecla F4.

En algún momento, tras iniciar la ejecución de nuestro programa, el control debe volver de nuevo al entorno de Delphi devolviendo éste al estado de diseño. La mejor forma de hacerlo es finalizar la ejecución del programa. Cuando ésta se encuentra en estado de pausa tan sólo hay que pulsar F9, como se ha indicado antes, procediendo después a cerrar la ventana o usar la opción que proceda para su finalización.

Existe, no obstante, una alternativa que permite finalizar la ejecución de un programa en cualquier momento, indistintamente de que nos encontremos en modo de pausa y ejecución. Dicha alternativa es la opción `Program Reset`, que equivale a la pulsación de Control-F2.

Puntos de parada

Conocemos al menos dos opciones que nos permiten pasar del estado de ejecución al de parada: `Pause Program` y `Run to Cursor`. Además de éstas existen otras y, entre ellas, principalmente los conocidos como puntos de parada. Éstos son unas marcas que indican al depurador que ha de cambiar de estado al llegar a una cierta sentencia, permitiendo así iniciar o continuar la sesión de depuración a partir de ese nuevo punto.

Para establecer un punto de parada lo primero que hay que hacer es situar el cursor en la sentencia que interese, pulsando a continuación F5 o usando la opción `Debug•Toggle Breakpoint` del menú emergente del Editor de código. De manera inmediata podremos ver que la sentencia se destaca del resto, generalmente apareciendo con el fondo rojo.

Una vez colocados los puntos de parada en aquellas sentencias a partir de las cuales deseemos ejecutar paso a paso, estaremos en disposición de lanzar la ejecución del programa de forma normal. En principio no notaremos nada especial, pero en cuanto el punto de ejecución llegue a una de las sentencias marcadas con un punto de parada Delphi pasará automáticamente al modo de pausa.

Cuando un punto de parada ya no nos sea útil podemos eliminarlo usando exactamente la misma opción utilizada para establecerlo, es decir, pulsando la tecla F5 o abriendo el menú emergente y seleccionando `Debug•Toggle Breakpoint`.

Los puntos de parada pueden clasificarse en grupos y también contar con condicionales. Si no existe un condicional el punto de parada detendrá la ejecución siempre que se llegue a la sentencia asociada, lo cual puede ser un poco tedioso si dicha sentencia se encuentra en el interior de un bucle o, simplemente, es ejecutada con cierta frecuencia. En estos casos podemos añadir el punto de parada usando la opción `Add Breakpoint•Source Breakpoint` del menú `Run`, que abre la ventana mostrada en la Figura 3. Como puede ver en ella, es posible establecer tanto un contador como una condición.

Para conocer cuáles son las propiedades de un punto de parada: la acción que efectúa, el contador, condicional asociado y grupo al que pertenece; bastará con situar el puntero del ratón sobre él. Al hacerlo, como se aprecia en el detalle de la Figura 4, aparece un texto flotante que nos informa. Si contamos con múltiples puntos de parada repartidos por el código, podemos usar la opción `View•Debug Windows•Breakpoints` para abrir la ventana mostrada en la Figura 5. En ella se enumeran todos los puntos de parada existentes en nuestro código, siendo posible acceder a la sentencia asociada de cada uno de ellos con un simple doble clic. También contamos con un menú emergente que nos permite operar sobre ellos.

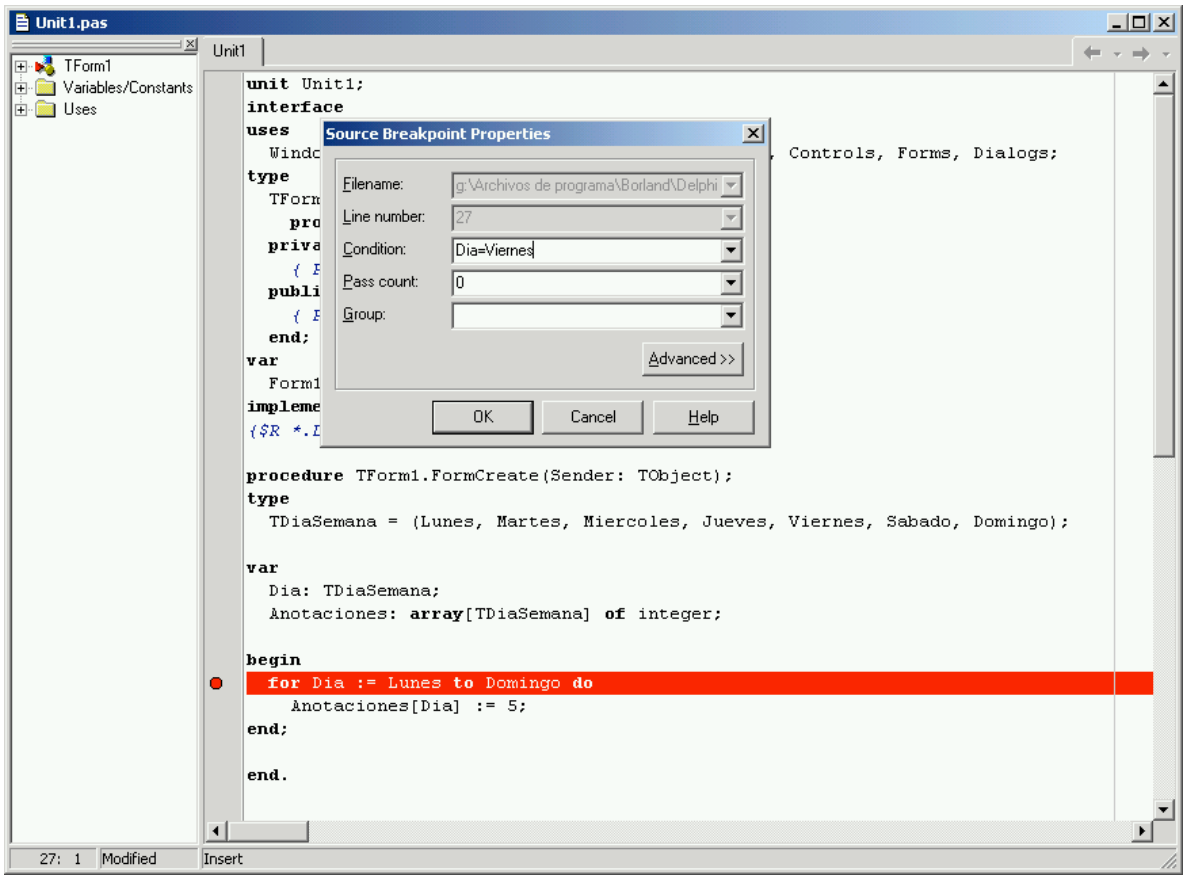

Figura 3. Los puntos de parada pueden tener asociados contadores y condicionales.

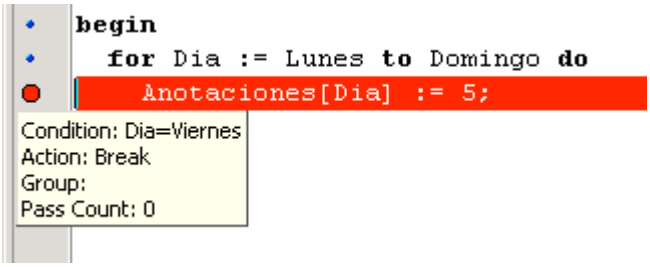

Figura 4. Basta con situar el puntero del ratón sobre un punto de parada para conocer sus propiedades.

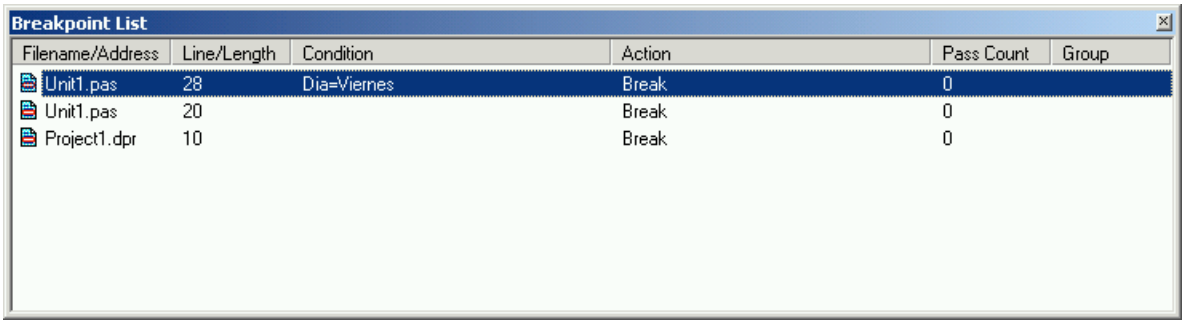

Figura 5. Ventana que enumera todos los puntos de parada existentes en nuestro proyecto.

Comprobación de expresiones y variables

El sólo hecho de poder ejecutar el código de nuestro programa sentencia a sentencia es ya de gran utilidad, siendo posible la detección de errores sólo con esta técnica. Esta tarea, no obstante, se complementa y simplifica gracias a la posibilidad de poder evaluar expresiones y comprobar el contenido de variables y objetos siempre que nos encontremos en el modo de pausa.

Podemos conocer el contenido de una variable o propiedad de un objeto de forma inmediata, con tan sólo situar el puntero del ratón sobre ella. En la Figura 6, por ejemplo, puede ver cómo, tras detenerse la ejecución en el punto de parada condicional que habíamos dispuesto previamente, comprobamos el valor de la variable `Dia`. Una alternativa consiste en situarnos sobre la variable y pulsar Control-F7 o elegir la opción `Evaluate/Modify` del menú emergente asociado. En este caso, como puede ver en la Figura 7, el contenido se examina en una ventana independiente. Lo más interesante es que en esta ventana podemos, además, modificar el valor de variables y evaluar expresiones. En la citada Figura 7, por ejemplo, estamos comprobando si la variable `Dia` contiene actualmente el valor `Viernes`.

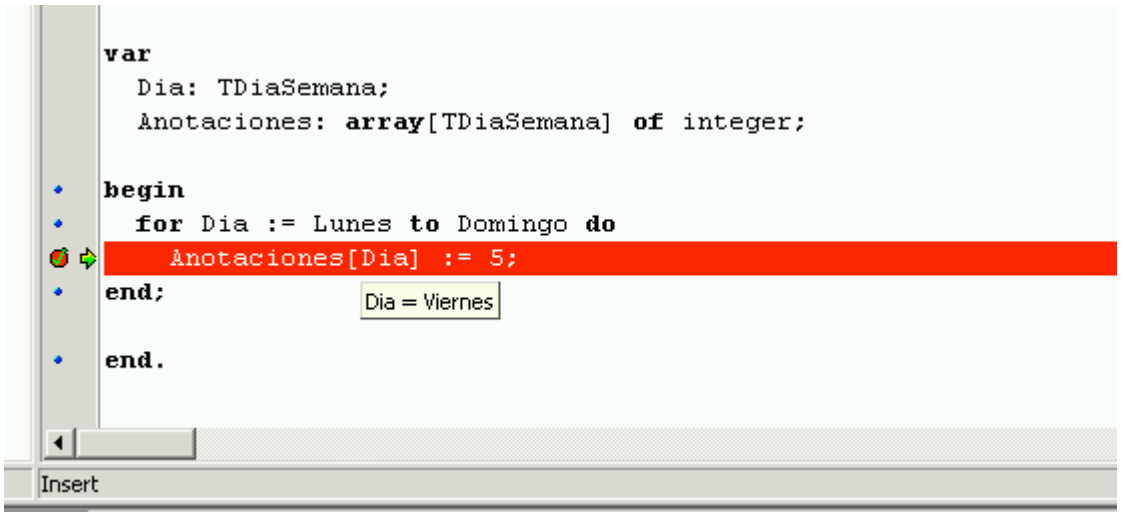

Figura 6. Sólo situando el puntero del ratón sobre una variable podemos conocer su contenido actual.

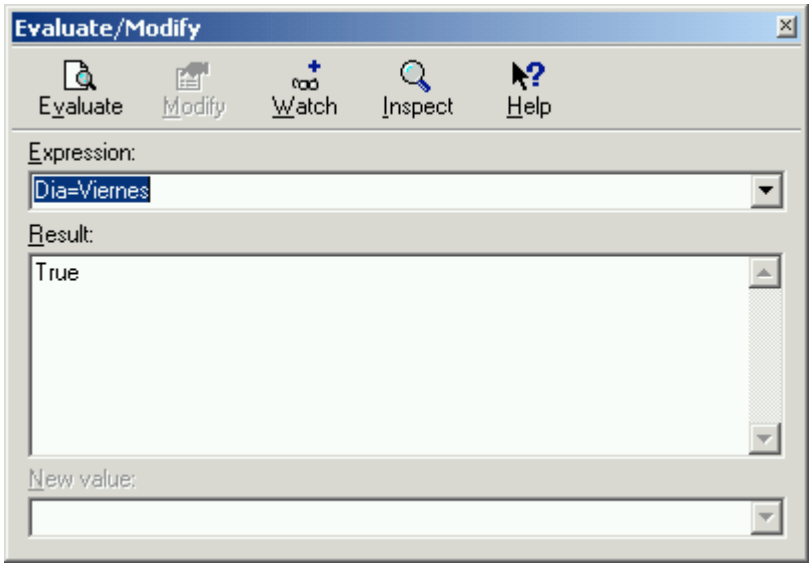

Figura 7. La ventana de evaluación permite comprobar el contenido de variables, modificarlo y evaluar expresiones.

Si nos encontramos en la necesidad de comprobar el valor de una misma variable o expresión con cierta frecuencia, no sólo de manera puntual, existen otras alternativas más cómodas a las dos opciones anteriores. Una de ellas consiste en abrir la ventana de visualizadores y añadir la expresión que deseemos, usando para ello la opción `Run•Add Watch` o la combinación Control-F5.

La lista de visualizadores, que puede ver en la Figura 8, puede acoplarse a otras y mantenerse siempre visible. De esta forma podremos ir viendo el valor de las variables o expresiones que deseemos a cada paso de la ejecución, sin necesidad de dar ningún paso adicional. Esta ventana, además, cuenta con un menú emergente en el que encontraremos las opciones necesarias para modificar las propiedades de los visualizadores, eliminarlos o añadir otros nuevos.

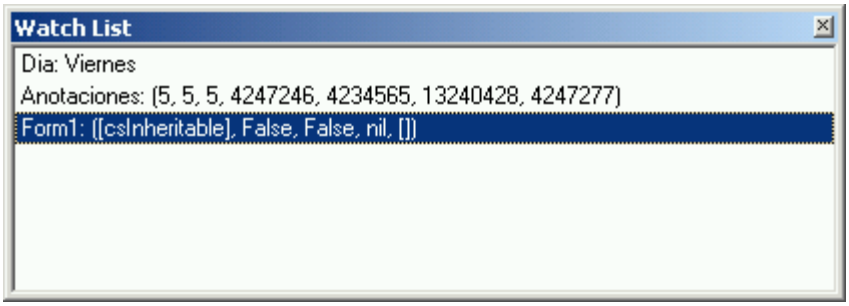

Figura 8. Manteniendo abierta esta ventana podremos conocer el valor de las variables y expresiones a medida que ejecutamos paso a paso.

¿De dónde vengo?

Al entrar en el estado de pausa, tras iniciar la ejecución del programa, nos encontraremos en un punto al que hemos llegado tras disponer un punto de parada, usar la opción `Run to Cursor` o interrumpir el programa de alguna otra forma. Independientemente de ello, es posible que estando en ese punto no sepamos muy bien cómo hemos llegado hasta él. Esto es habitual cuando estamos en un procedimiento o función que es llamado desde distintos puntos de la aplicación.

La solución a este problema es tan rápida como seleccionar la opción `View•Debug Windows•Call Stack`, que hacer aparecer una ventana similar a la de la Figura 9. Ésta es conocida como la ventana que muestra la pila de llamadas. En ella aparecen, en orden inverso, los nombres de los procedimientos por lo que se ha pasado hasta llegar al punto actual, que ocupa el primer puesto en la lista. De esta forma podemos saber de manera inmediata cuál ha sido la secuencia de ejecución que nos ha llevado hasta este punto.

Aparte de esta ventana y las que hemos visto en puntos previos, en el submenú `View•Debug Windows` contamos con opciones que nos permiten abrir muchas otras. Con ellas podemos analizar el estado de los hilos de ejecución del programa, los módulos que éste está utilizando o, incluso, conocer el estado de los registros de la CPU y FPU.

Visto y por ver

A lo largo de las seis entregas que ha durado este curso, que llega a su punto final, se ha pretendido introducir al lector en el uso de una de las herramientas de desarrollo más polivalentes y conocidas en la actualidad: Borland Delphi. Si, partiendo de unos conocimientos nulos, al llegar a esta última entrega el lector se siente cómodo en el entorno de Delphi, sabe cómo diseñar una interfaz básica, escribir código en Object Pascal, ejecutar y depurar su proyecto, habremos conseguido nuestro objetivo. Obviamente no será un experto en la materia, pero se encontrará en el punto adecuado para iniciar un aprendizaje en profundidad de Object Pascal, Delphi y sus componentes contando con una indispensable visión general.

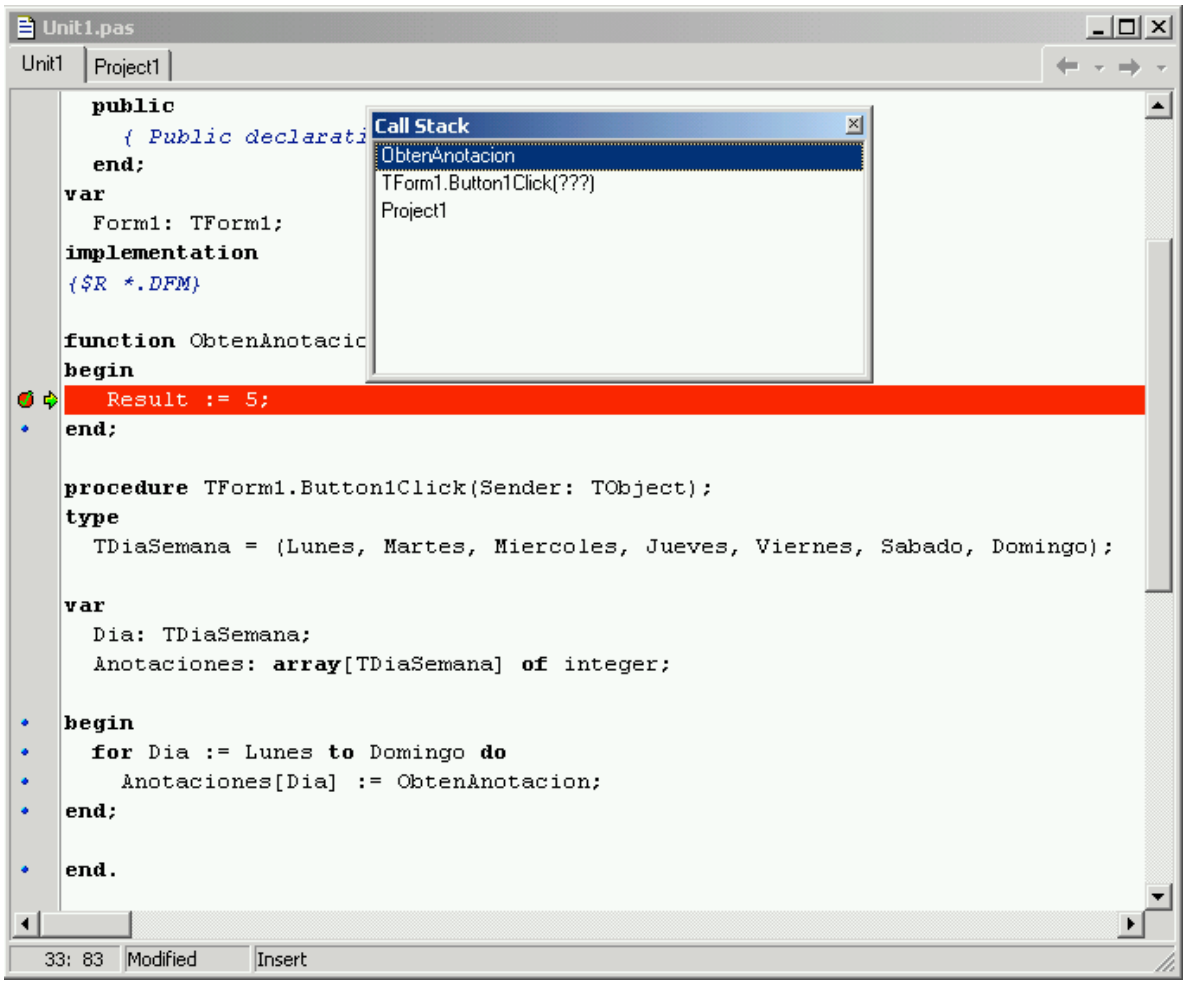

Figura 9. La ventana de la pila de llamadas aparece sobre el editor de código