



Debugger en Eclipse y Netbeans

Descripción

En esta práctica se analiza cómo trabajar con el **debugger** de Eclipse o Netbeans.

Un debugger o depurador es la herramienta software que permite analizar el código mientras se ejecuta de forma controlada. Con él se puede encontrar la causa de un error, o incluso conocer mejor su funcionamiento. Algunas de las acciones principales son: **establecer puntos de interrupción** o de ruptura, suspender la ejecución del programa, **ejecutar el código paso a paso** o examinar el contenido de las variables.

EL DEPURADOR DE ECLIPSE

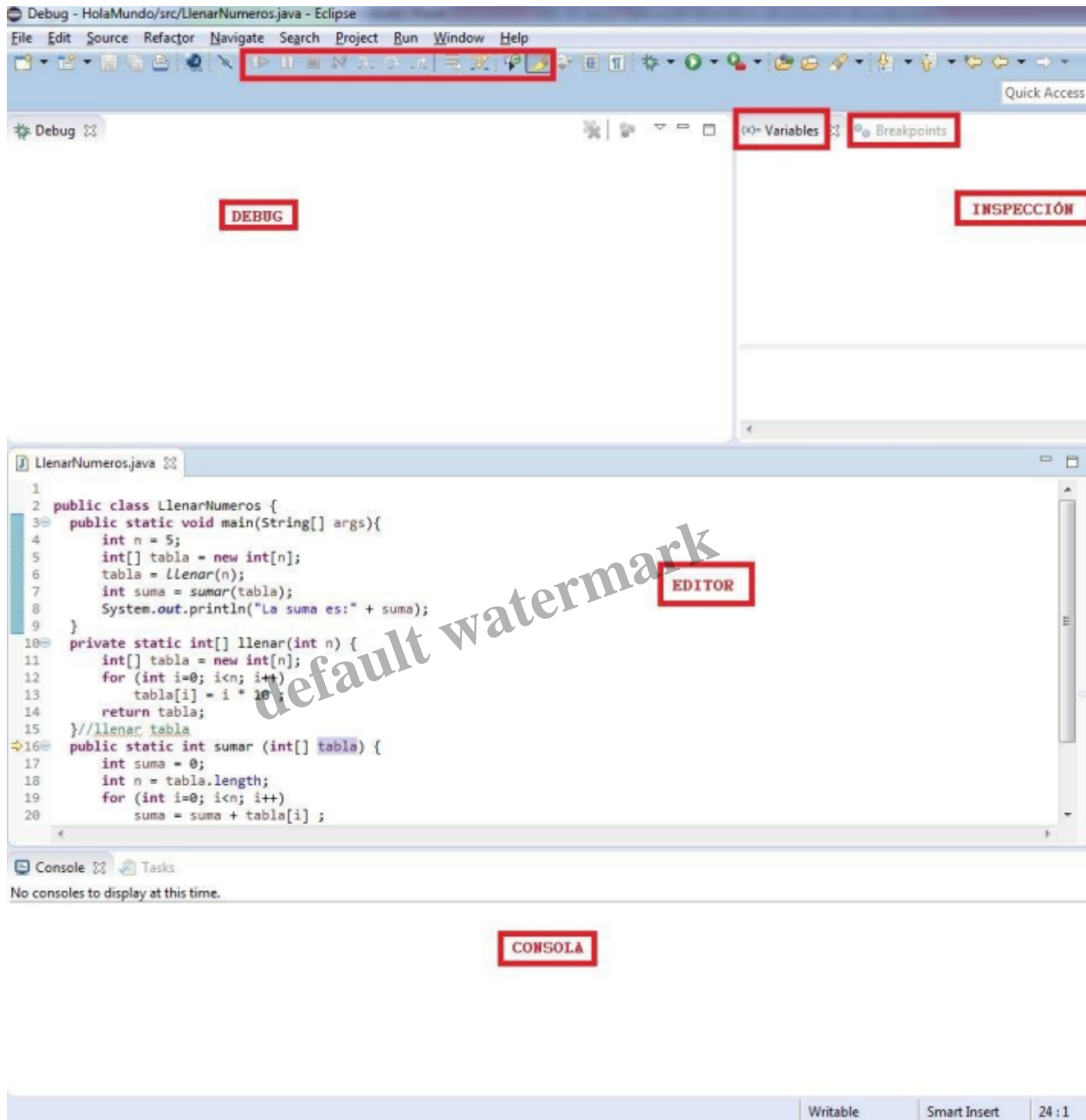
Hay **varias maneras** de lanzar el **debugger** en Eclipse:

- Pulsando el botón **Debug**
- Seleccionando el menú **Run > Debug As**
- Botón derecho del ratón en la clase a ejecutar y seleccionando en el menú contextual **Debug as > Java Application**

En todos estos casos, la clase se ejecuta. Pero es conveniente en todos ellos abrir la **vista de depuración** desde el menú **Windows > perspective > Open perspective > Other > Debug**. Veremos información relativa al programa que se está ejecutando. Si no tuviéramos abierta esta vista, se mostraría un mensaje al ejecutar en modo depuración preguntándonos por su apertura.

Al cambiar a la vista depuración, se muestra en la barra de herramientas una serie de botones: para continuar la ejecución, suspenderla, pararla, para meterse dentro de la primera línea de un método, avanzar un paso la ejecución, avanzar el programa hasta salir del método actual, etc.

En esta vista se pueden ver varias zonas:



EDITOR: se va marcando la traza de ejecución del programa y se ve una flecha azul en el margen izquierdo de la línea que se está ejecutando

DEBUG: muestra los hilos de ejecución. En la imagen solo se muestra un hilo y debajo la clase en la que está parada la ejecución mostrando el número de línea.

INSPECCIÓN: muestra los valores de las variables y de los puntos de ruptura (breakpoints) que intervienen en el programa en un momento determinado. Desde aquí se puede modificar el valor de

las variables haciendo clic en **On** y cambiándolo (el nuevo valor se usará en los siguientes pasos de ejecución). También desde la pestaña Breakpoints se puede activar o desactivar un punto de ruptura, eliminarlo, configurarlo para que la ejecución se detenga cuando se pase por **On** un número determinado de veces, etc.

CONSOLA: muestra la consola de ejecución del programa que se está depurando.

Veamos ahora el menú Run de la perspectiva de depuración:

- **Resume:** reanuda un hilo suspendido. Se usa cuando no queremos analizar instrucciones por instrucción y queremos que el depurador se pare en la siguiente línea donde hay un breakpoint
- **Suspend:** suspende el hilo seleccionado
- **Terminate:** finaliza el proceso de depuración
- **Step Into:** se ejecuta paso a paso cada instrucción. Si el depurador encuentra una llamada a un método o función, al pulsar esta opción se irá a la primera instrucción de dicho método.
- **Step Over:** se ejecuta paso a paso cada instrucción, pero si el depurador encuentra un método, al pulsar esta opción se irá a la siguiente instrucción, sin entrar en el código del método
- **Step Return:** si nos encontramos dentro de un método, el depurador sale del método actual
- **Run to Line:** se reanuda la ejecución de código a partir de la línea seleccionada
- **Use Step Filters:** cambia los filtros de paso de activado a desactivado. Estos filtros están definidos en el menú Windows > Preferences > Java > Debug > Step Filtering. Se usan sobre todo para filtrar tipos que no se desean recorrer durante la ejecución.
- **Run:** ejecutar el programa
- **Debug:** ejecutar en modo depuración
- **Skip All Breakpoints:** se omiten todos los breakpoints
- **Toggle:** establece u omite puntos de ruptura a nivel de línea o de método
- **Remove All Breakpoints:** elimina todos los breakpoints
- **Add Java exception Breakpoints:** permite añadir una excepción Java como breakpoint
- **All instances:** abre un cuadro de diálogo emergente que muestra una lista de todas las instancias del tipo Java seleccionado
- **Watch:** permite crear nuevas expresiones basadas en las variables del programa
- **Inspect:** crea una nueva expresión para la variable seleccionada y la agrega a la vista de inspección
- **Display:** muestra el resultado de evaluar la expresión seleccionada

SEGUIMIENTO DE PUNTOS DE RUPTURA

Creando un nuevo proyecto en Eclipse y llámalo **P1T4**. Copia el siguiente fichero Java y guárdalo como **LlenarNumero.java**:

```
public class LlenarNumero {
    public static void main(String[] args) {
        int n = 5;
        int[] tabla = new int[n];
        tabla = llenar(n);
        int suma = sumar(tabla);
    }
}
```

```
System.out.println(«La suma es:» + suma);
}
private static int[] llenar(int n) {
int[] tabla = new int[n];
for (int i = 0; i < n; i++)
tabla[i] = i * 10;
return tabla;
} // llenar tabla
public static int sumar(int[] tabla) {
int suma = 0;
int n = tabla.length;
for (int i = 0; i < n; i++)
suma = suma + tabla[i];
return suma;
} // sumar tabla
} // fin c
```

Fíjate que se definen tres bloques, el método **main()**. El método **Llenar()** recibe el parámetro entero **n** y devuelve un array (vector) con **n** enteros. El método **Sumar()** recibe un array (vector) de datos, suma sus elementos

Vamos a empezar el proceso de depuración abriendo la vista de depuración (menú Window > perspective > Open perspective > other > Debug).

Ahora, para empezar a depurar una clase establecemos un breakpoint. Para ello, se ha de seleccionar una línea en nuestro código donde queremos que la ejecución se detenga y así podremos ver los valores que tienen las variables en ese momento. Para ponerlo, hacemos doble click en el margen izquierdo del editor, justo en la línea donde queremos que se detenga la ejecución. Aparecerá un circulito a la izquierda. Pon uno en la misma línea que se ve en la figura (tabla[i] = i*10;) línea 14. En esta línea se va llenando el array): Estableciendo un breakpoint y devuelve la suma.

```
*LlenarNumeros.java
1 public class LlenarNumeros {
2     public static void main(String[] args){
3         int n = 5;
4         int[] tabla = new int[n];
5
6         tabla = llenar(n);
7         int suma = sumar(tabla);
8         System.out.println("La suma es:" + suma);
9     }
10
11     private static int[] llenar(int n) {
12         int[] tabla = new int[n];
13         for (int i=0; i<n; i++)
14             Line breakpoint:LlenarNumeros [line:14] - llenar(int)
15             return tabla;
16     }//llenar tabla
17
18     public static int sumar(int[] tabla) {
19         int suma = 0;
20         int n = tabla.length;
21         for (int i=0; i<n; i++)
```

Ahora ejecuta el programa en modo depuraci3n. Para ello, pulsa el bot3n Debug. Lo que ocurrir3 ser3 que el programa se ejecutar3 de manera natural hasta llegar al punto de ruptura establecido.

En la ventana Debug aparece la pila de llamadas, donde se ven cada uno de los hilos de ejecuci3n. En este caso solo hay uno (Thread main[]).

Debajo de esta l3nea se muestra el nombre de la clase con el m3todo donde est3; ahora la ejecuci3n parada clase LlenarNumero, m3todo llenar() y se muestra tambi3n el n3mero de l3nea donde est3; detenida la ejecuci3n (l3nea 22):

La siguiente l3nea muestra qui3n ha llamado a este m3todo, en este caso la llamada se hace desde la clase LlenarN3mero y dentro del m3todo main() en la l3nea 14, en esta l3nea est3; la sentencia tabla = llenar(n). Al hacer click en estas l3neas se muestra la l3nea de c3digo que se est3; ejecutando.

Ahora podemos usar los siguientes botones:

- **Step Into:** para ejecutar el programa paso a paso (instrucci3n por instrucci3n)
- **Step Over:** para ejecutar instrucci3n a instrucci3n pero si encuentra un m3todo saltarlo
- **Step Return:** si estamos dentro de un m3todo, al pulsarlo el depurador hace avanzar la ejecuci3n del programa saliendo del m3todo

En cualquier momento podemos finalizar el depurador pulsando **Terminate**

Una flechita nos indica la l3nea que se est3; ejecutando. Para quitar el punto de ruptura de alguna l3nea simplemente hacemos doble click en el circulito.

Se pueden establecer puntos de ruptura condicionales. Vamos a modificar el que tenemos para que la ejecuci3n se detenga cuando i = 3. Para ello, desde la zona "Inspecci3n" hacemos clic en la

pestaña Breakpoints y marcamos la casilla Conditional. Seleccionamos Suspend when `!true` y escribimos la condición a evaluar. Esta condición debe devolver un valor booleano.

Ahora cuando la `i` alcance el valor 3 se parará la ejecución y podremos empezar a usar los botones para ejecutar paso a paso.

Si queremos eliminar este breakpoint seleccionamos el botón y con los borraremos todos.

EL DEPURADOR DE NETBEANS

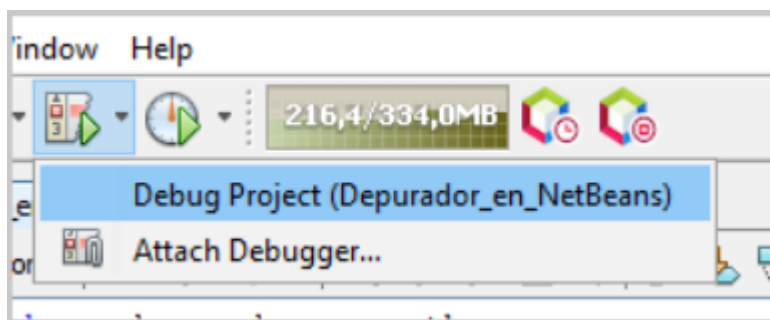
El depurador de Netbeans es mas sencillo y muy parecido. Primero crear el siguiente proyecto con nombre: **depurador_en_netbeans con el siguiente código**

```
package depurador_en_netbeans;

public class Depurador_en_NetBeans {

    public static void main(String[] args) {
        int n = 0;
        for (int i = 0; i < 10; i++) {
            n = n + i;
        }
        System.out.println(n);
    }
}
```

Si lo ejecutáis da de solución: 45 . Ahora con el debugger vamos a entender que sucede. Añadir un breakpoint en la línea donde esta el `n = n + i`; (mi línea 8). Se marcará la línea en rojo. Lanzamos el código en modo depurador es



En la ventana inferior aparecerá una ventana llamada Variables en la cual si vamos avanzado la ejecución del código con los botones veremos como cambia el valor de las variables , en este caso el botón verde.



Para quitar un breakpoint doble click en .

Categoría

1. Programación

Fecha de creación

9 de noviembre de 2019

Autor

joseordazwp

default watermark