



Introducción al depurador

Curso 2023-2024

Entornos de desarrollo
Desarrollo de Aplicaciones Web

Problems Javadoc Declaration Console Debug

<terminated> OWLAPITutorial [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_05.jdk/Contents/Home/bin/java (19/8/2015 11:43:08)

Exception in thread "main" java.lang.NoClassDefFoundError: com/google/inject/Provider

```
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:760)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:455)
at java.net.URLClassLoader.access$100(URLClassLoader.java:73)
at java.net.URLClassLoader$1.run(URLClassLoader.java:367)
at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:360)
at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:260)
at CompileSourceInMemory.main(CompileSourceInMemory.java:50)
```

Output - compiler (run) X

run:

Success: true

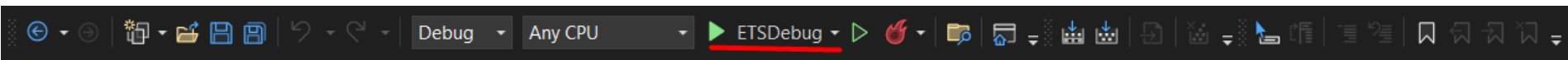
java.lang.ClassNotFoundException: HelloWorld

```
at java.net.URLClassLoader$1.run(URLClassLoader.java:372)
at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:360)
at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
at java.lang.Class.forName0(Native Method)
at java.lang.Class.forName(Class.java:260)
at CompileSourceInMemory.main(CompileSourceInMemory.java:50)
```

BUILD SUCCESSFUL (total time: 2 seconds)

Encontrar errores

- Una de las tareas más comunes del programador pero más infravaloradas es la capacidad que tiene de encontrar errores en un código y de solucionarlos. De hecho, un [50% del tiempo](#) de los trabajadores se emplea para encontrar errores en el código (según un cuestionario rellenado por programadores en USA)
- Encontrar errores en el código de forma manual, sin ayuda y sin estrategia puede ser un proceso estresante que dificulta que acabemos las tareas que tengamos que hacer.
- Es por ello que la mayoría de los IDE ofrecen una herramienta llamada **debugger (depurador)** que nos ayuda a detectar y corregir errores.
- Para depurar nuestro código con el VS, hay que darle al play que tenéis con fondo relleno de color verde.



Ejemplo

- Vamos a depurar un ejemplo:
- Tenemos un programa que le pide a un usuario un número en bucle mientras no introduzca 0. Cuando leo el numero, calculo si es divisor de 5 y le informo de si es divisible por 5 o no.

```
int numero = 0;
int divisor = 5;
int resultado = 0;

String nombre = "El rey de españa";
Console.WriteLine("Hola " + nombre + " dime n mero y te digo si son m ltiplos de " + divisor);
numero = int.Parse(Console.ReadLine());
while (numero != 0)
{
    resultado = numero / divisor;
    if (resultado == 0) Console.WriteLine("Si, el numero " + numero + " es divisble por " + divisor);
    else Console.WriteLine("NO!!!, el numero " + numero + " NO es divisble por " + divisor);
    Console.WriteLine("Introduce otro numero");
    numero = int.Parse(Console.ReadLine());
}
```

% (resto)

```
Hola El rey de españa dime n mero y te digo si son m ltiplos de 5
15
NO!!!, el numero 15 NO es divisble por 5
Introduce otro numero
```

Encontrar errores

- Por lo general lo que hace un depurador es analizar el código del software paso a paso, y para ello se establecen unos **puntos de control o breakpoints**, lugar donde la ejecución del código se parará, y cuando el usuario lo desee, se retomará la ejecución desde el mismo punto.
- Paramos el programa en un punto por dos factores:
 - Poder ver en qué estado se encuentra nuestro programa, es decir, qué valores tienen nuestras variables, si ha ocurrido ya un error
 - Para poder descartar errores, puesto que si se llega bien a un breakpoint quiere decir que no ha ocurrido ningún error
- En VS, para poder crear breakpoints, tenemos que hacer click en la banda grisácea que hay al lado izquierdo del código, y aparecerá una bola. También se puede colocar pulsando F9
- Este breakpoint significa que la ejecución se parará allí.

```
Program.cs  + X
C# Ejercicio 1
{
1  // TODO: Auto-generated method stub
2
3  int numero = 0;
4  int divisor = 5;
5  int resultado = 0;
6
7  String nombre = "El rey de España";
8  Console.WriteLine("Hola " + nombre + " dime número y te digo si son múltiplos de " + divisor);
9  numero = int.Parse(Console.ReadLine());
10 while (numero != 0)
11 {
12     resultado = numero / divisor;
13     if (resultado == 0) Console.WriteLine("Si, el número " + numero + " es divisible por " + divisor);
14     else Console.WriteLine("NO!!!, el número " + numero + " NO es divisible por " + divisor);
15     Console.WriteLine("Introduce otro número");
16     numero = int.Parse(Console.ReadLine());
17 }
```

Encontrar errores

- La interfaz se nos va a actualizar con varias opciones que no teníamos antes:

The screenshot displays the Visual Studio Code interface with a C# program in the main editor and various diagnostic tools on the right and bottom.

Program.cs

```
1 // TODO: Auto-generated method stub
2
3 int numero = 0;
4 int divisor = 5;
5 int resultado = 0;
6
7 String nombre = "El rey de españa";
8 Console.WriteLine("Hola " + nombre + " dime número y te digo si son mltiplos de " + divisor);
9 numero = int.Parse(Console.ReadLine());
10 while(numero != 0)
11 {
12     resultado = numero / divisor;
13     if (resultado == 0) Console.WriteLine("Si, el numero " + numero + " es divisble por " + divisor);
14     else Console.WriteLine("NO!!!, el numero " + numero + " NO es divisble por " + divisor);
15     Console.WriteLine("Introduce otro numero");
16     numero = int.Parse(Console.ReadLine());
17 }
18
```

Herramientas de diagnóstico

Sesión de diagnóstico: 3 segundos (3,006 s selec...)

Eventos

Memoria de proceso (MB) 13

CPU (% de todos los procesadores) 100

Resumen Eventos Uso de memoria Uso de CPU

Eventos

Mostrar eventos (1 de 1)

Uso de memoria

Tomar instantánea

Uso de CPU

Registrar perfil CPU

Automático

Buscar (Ctrl+E)

Profundidad de búsqueda: 3

| Nombre | Valor | Tipo |
|--------|-------|------|
| numero | 6 | int |

Pila de llamadas

| Nombre | Leng |
|--|------|
| Ejercicio 1.dll Program.<Main>\$(string[] args) Línea 10 | C# |

Automático Variables locales Inspección 1

Pila de llamadas Puntos de interrupción Configuración de excepciones Ventana Comandos Ventana Inmediato Salida

Encontrar errores



Estos botones nos permiten ir moviéndonos entre breakpoints, es decir, poder reiniciar el código y pararlo. Explico los botones desde la izquierda:

- El botón de continuar permite reanudar el programa y ejecutar las instrucciones hasta el siguiente breakpoint
- El botón de detener, el cuadrado rojo, sirve para parar la ejecución del depurador
- El botón de reiniciar permite relanzar el programa
- El siguiente botón que es una flecha en blanco permite ver la siguiente instrucción a ejecutar
- El siguiente botón es **Step Into** y lo que hace es avanzar un paso la ejecución del programa, **metiéndose en la función**.
- El siguiente botón lo que hace es ejecutar la instrucción que está marcada, la acción conocida como **Step Over**. **Si la instrucción es la llamada a una función, pasa por encima, no se mete.**
- El último botón lo que hace es salir del método en el que estemos, o **Step Out**

Encontrar errores



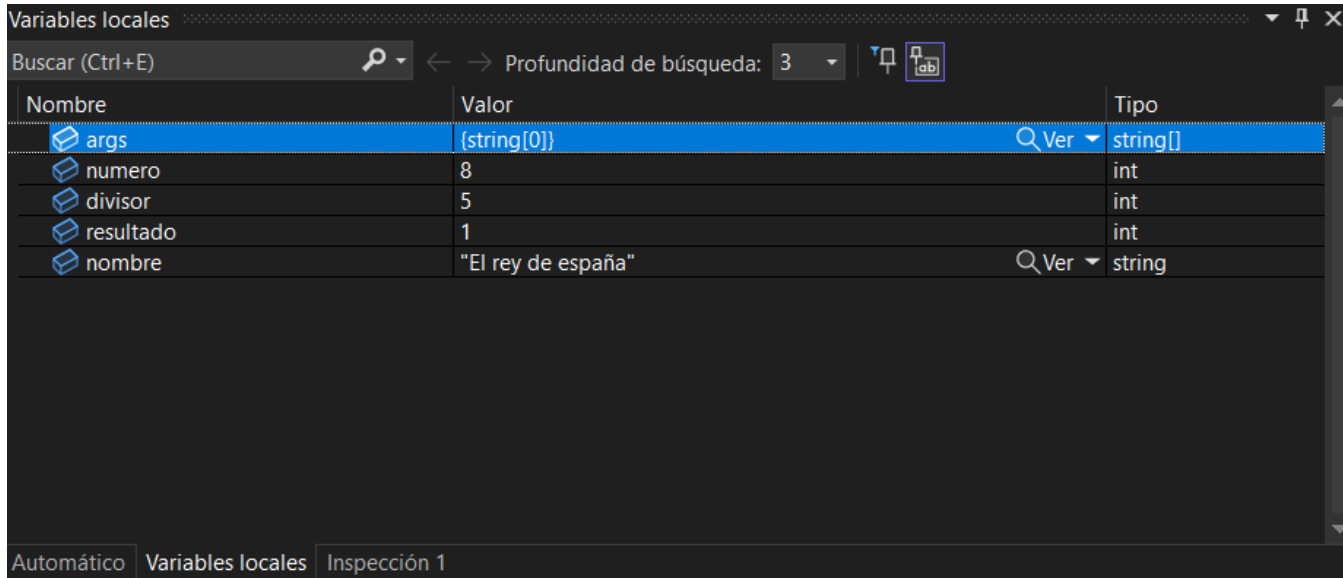
La flecha hacia abajo con el punto (**F11**) es **paso a paso por instrucción**. Hace que el depurador entre dentro de la llamada al método en la línea actual y detenga la ejecución allí. Si la línea actual no contiene una llamada al método, mueve el depurador a la línea siguiente.

La flecha hacia la derecha con el punto (**F10**) es **paso a paso por procedimiento**. Avanza el depurador a la siguiente línea sin depurar paso a paso ninguna llamada al método. Es útil cuando queremos obtener el valor devuelto de un método sin entrar en los detalles de implementación.

La flecha hacia arriba con el punto (**Shift+F11**) es **paso a paso para salir**. Continúa ejecutando la función actual y pausa la ejecución cuando se devuelva. Nos lleva de vuelta al punto en el que decidimos entrar.

Encontrar errores

En la parte inferior (Variables Locales) podemos ver una lista de las variables y su valor en el momento en el que se paró la ejecución por el breakpoint.



Variables locales

Buscar (Ctrl+E) 🔍 < > Profundidad de búsqueda: 3 🔍

| Nombre | Valor | Tipo |
|-----------|--------------------|----------|
| args | {string[0]} | string[] |
| numero | 8 | int |
| divisor | 5 | int |
| resultado | 1 | int |
| nombre | "El rey de españa" | string |

Automático Variables locales Inspección 1

Encontrar errores

Podéis ver el valor que tiene cada variable asignada al poner el ratón encima de la variable.

```
while (numero != 0)
{
    resultado = numero / divisor;
    if (resultado > 1) Console.WriteLine("S");
    else Console.WriteLine("NO!!!, el numero");
    Console.WriteLine("Introduce otro numero");
    numero = int.Parse(Console.ReadLine());
}
```

En ocasiones nos puede interesar monitorear el valor de una variable a medidas que navegamos por el código. Pincha en la variable con botón derecho y dale en Agregar Inspección.

Inspección 1

Buscar (Ctrl+E) Profundidad de búsqueda: 3

| Nombre | Valor |
|--------|---|
| result | 76 X 1 = 7676 X 2 = 15276 X 3 = 22876 X 4 = 30476 X 5 = 38076 X 6 = 45676 X 7 = 53... |

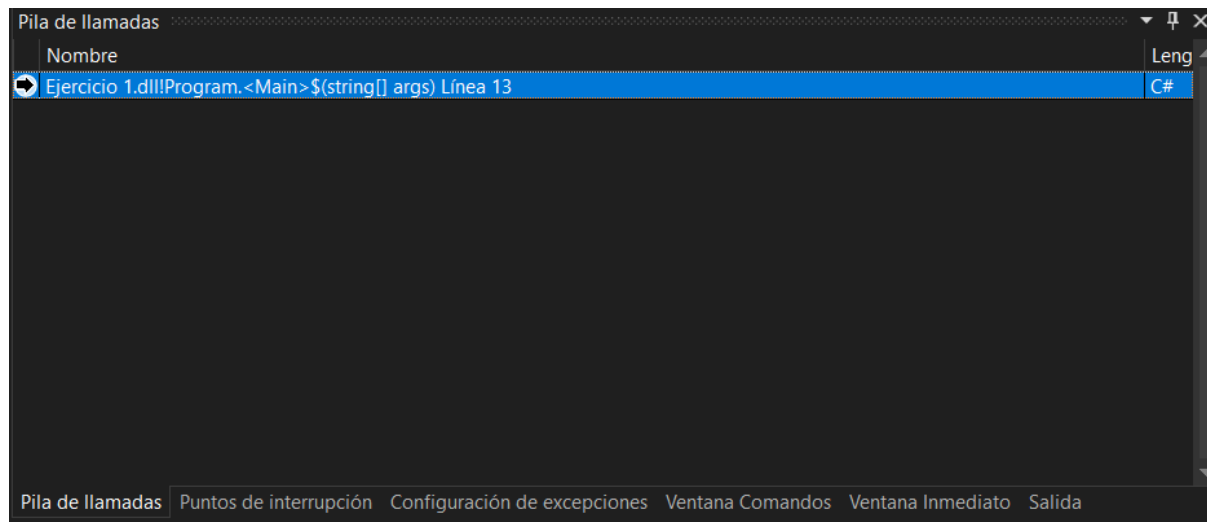
Agregar elemento a inspección

Salida Variables locales Automático Inspección 1

Encontrar errores

A la izquierda tendremos nuestra pila de llamadas, para ver el orden en el que se llaman las funciones.

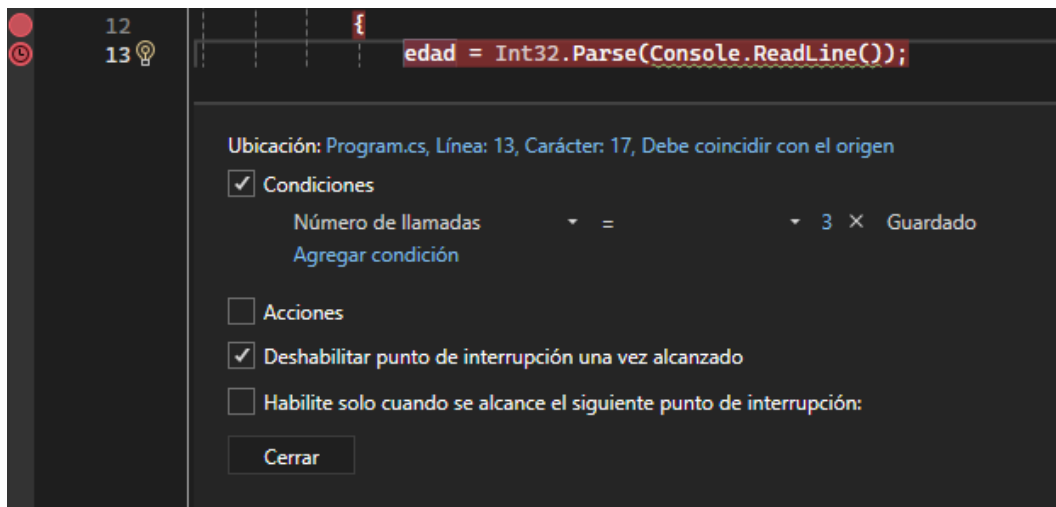
De momento sólo veréis la función **main**, pero a medida que vayáis modularizando vuestros programas, podéis ir viendo las llamadas a las funciones.



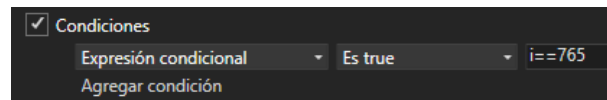
Encontrar errores

Si hacemos clic sobre un breakpoint podemos deshabilitarlo (se queda el círculo en rojo sin relleno).

Si pinchamos en configuración del breakpoint se puede parar la ejecución añadiendo alguna condición, por ejemplo cuando se haya ejecutado ese breakpoint X número de veces



La depuración condicional es especialmente útil cuando se trabaja con bucles. Imagina una lista con miles de objetos y sólo queremos comprobar el 765.





Vídeos y páginas de refuerzo

https://www.youtube.com/watch?v=7avOkI9D9BQ&ab_channel=GFCSoft

<https://code-maze.com/debugging-csharp-visual-studio/>

Ejercicio

Crea la siguiente aplicación que llamarás Tabla de Multiplicar:

```
static void Main(string[] args)
{
    var result = new StringBuilder();
    byte number = 3;
    Console.WriteLine("Introduzca un número para calcular su tabla de multiplicar:");
    number = byte.Parse(Console.ReadLine());

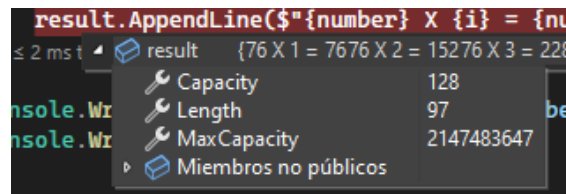
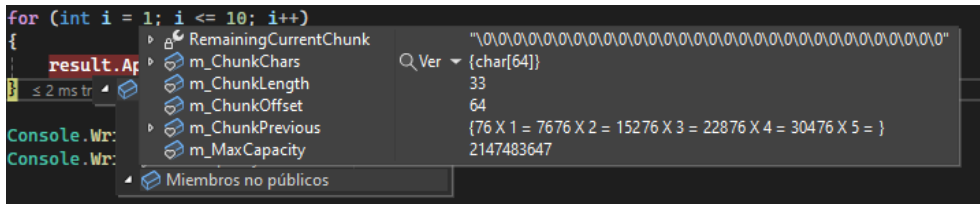
    for (int i = 1; i <= 10; i++)
    {
        result.AppendLine($"{number} X {i} = {number * i}");
    }

    Console.WriteLine("La tabla del: {0}", number);
    Console.WriteLine(result.ToString());
}
```

Ejercicio

Realiza lo siguiente:

- 1) Teclea F5 o Iniciar depuración. No hay puntos de ruptura así que se ejecutará sin más.
- 2) Colocar un breakpoint en la línea `result.AppendLine...`
- 3) Utiliza la tecla F10 (o la flecha Paso a Paso por procedimiento) para comprobar como avanza en el bucle. Observa como la variable `result` va tomando nuevos valores
- 4) Pincha sobre la variable `result` para que veas que puedes examinar sus valores
- 5) Agrega la variable a inspección
- 6) Modifica la parada condicional en la línea `result.AppendLine` para que el breakpoint tenga efecto cuando el valor de `i` sea 7



Realiza lo siguiente:

- 1) Abre el código que teníamos de la primera unidad **Coche.java**
- 2) Investiga como colocar un breakpoint y ejecuta la depuración del código¿De qué manera puedo avanzar paso a paso¿Dónde ves las variables?¿Cómo desactivo un breakpoint sin eliminarlo?
- 3) Ahora abre el código **DiaSiguiente.java** en un nuevo proyecto con NetBeans. Coloca un breakpoint en la línea “aa = sc.nextInt();. Investiga de que manera puedes avanzar saltándote las llamadas a los procedimientos (que no entre a comprueba(...)).
- 4) Investiga como colocar un breakpoint condicional en la línea “**switch(mm)**” sólo en el caso de que la variable mm tenga el valor 12.
- 5) Haz una depuración y prueba la opción “Run to cursor (F4)”. ¿Qué ventaja te da?
- 6) Prueba la opción New Watch (inspección) para una variable y fíjate en la pestaña que te las muestra.

Realiza lo siguiente:

- 1) Crea un nuevo proyecto en VS que se llame “DiaSiguiente”. Copia el código DiaSiguiente.cs
- 2) Coloca un breakpoint en la línea “aa = Conver.To...”. Comprueba que sabes utilizar la opción adecuada del depurador para que no entre a la función **comprueba(...)**
- 3) Haz una depuración ahora línea a línea y observa en “Pila de Llamadas” como aparecen las funciones que son llamadas.
- 4) Coloca un breakpoint condicional en la línea “**switch(mm)**” sólo en el caso de que la variable **mm** tenga el valor 12.
- 5) Agrega las variables dds, mms y aas a inspección. Haz una depuración para comprobar que aparecen sus valores en la pestaña correspondiente.
- 6) Modifica el código de forma que lance una excepción si el año es menor que 0 ó mayor que 2100. Saldrá un mensaje que dirá “El año introducido no es válido (Válido: 0-2100).”.
- 7) **Reto**. Modifica el código para que el programa no acabe si introduces una fecha errónea. Deberá preguntarte una fecha hasta que introduzcas una válida.