

- 1 Introducción
- 2. Crea el proyecto
- 3. IP-Scanner
- 4. Breakpoints
- 5. Iniciar la depuración
- 6. Ejecución paso a paso
  - 6. 1. Breakpoints condicionales
  - 6.2 Evaluar expresiones

Fecha	Versión	Descripción
16/09/2021	1.0.0	Versión inicial



# Unión Europea

Fondo Social Europeo

*El FSE invierte en tu futuro*

## 1 Introducción



El objetivo de este CodeLab es la iniciación al uso de depurador (debugger).

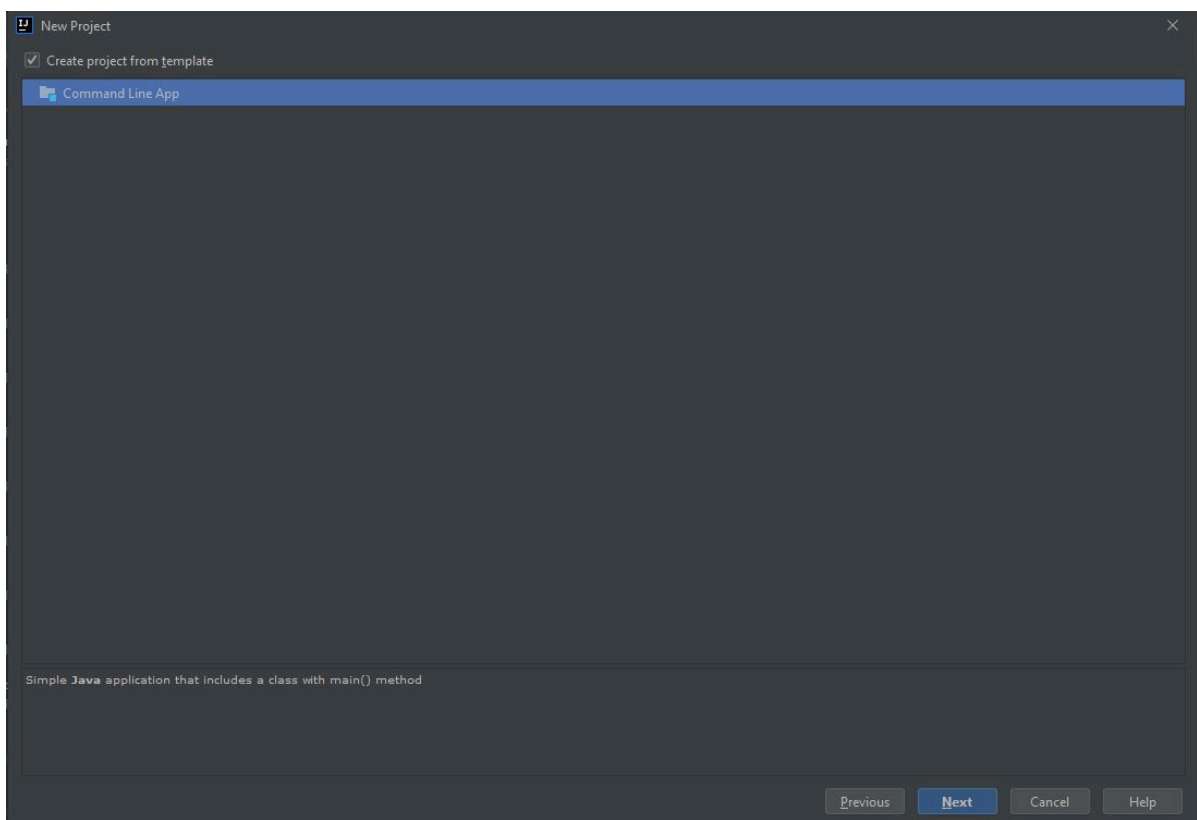
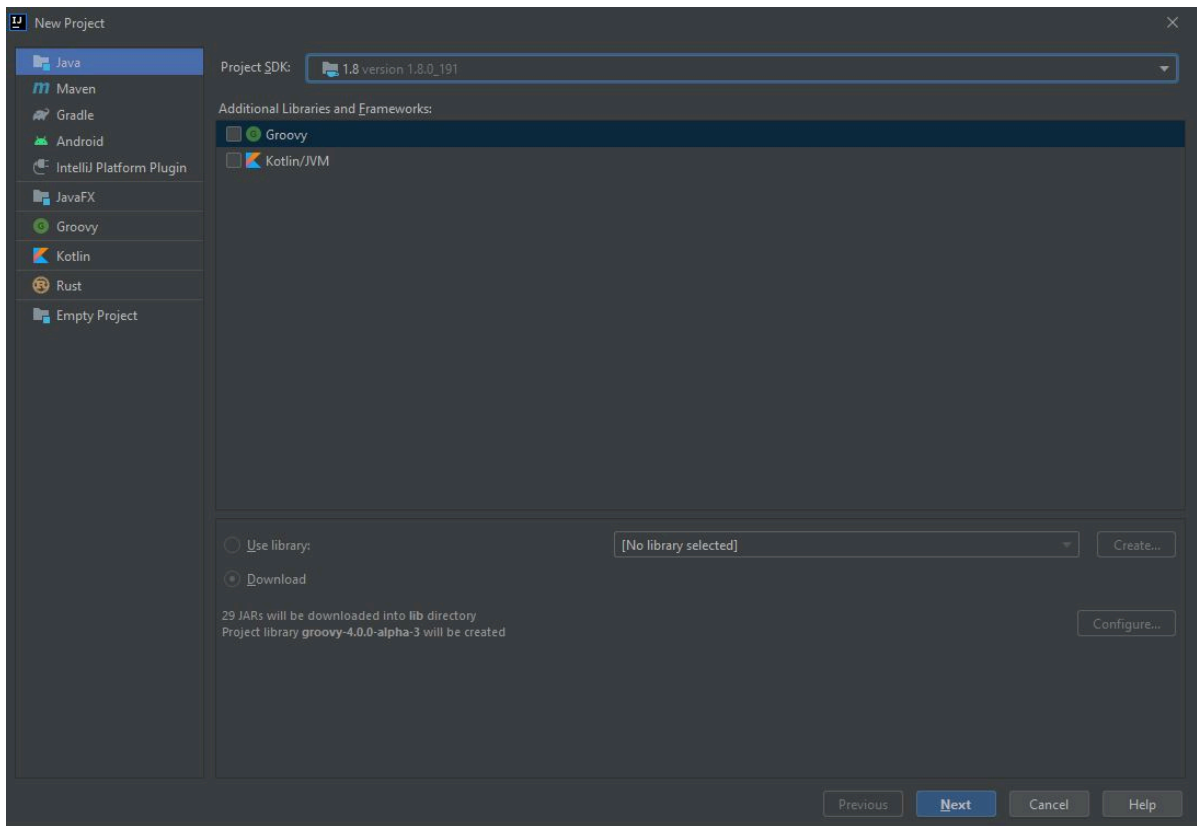
La depuración de programas es el proceso de identificar y corregir errores de programación (bugs).

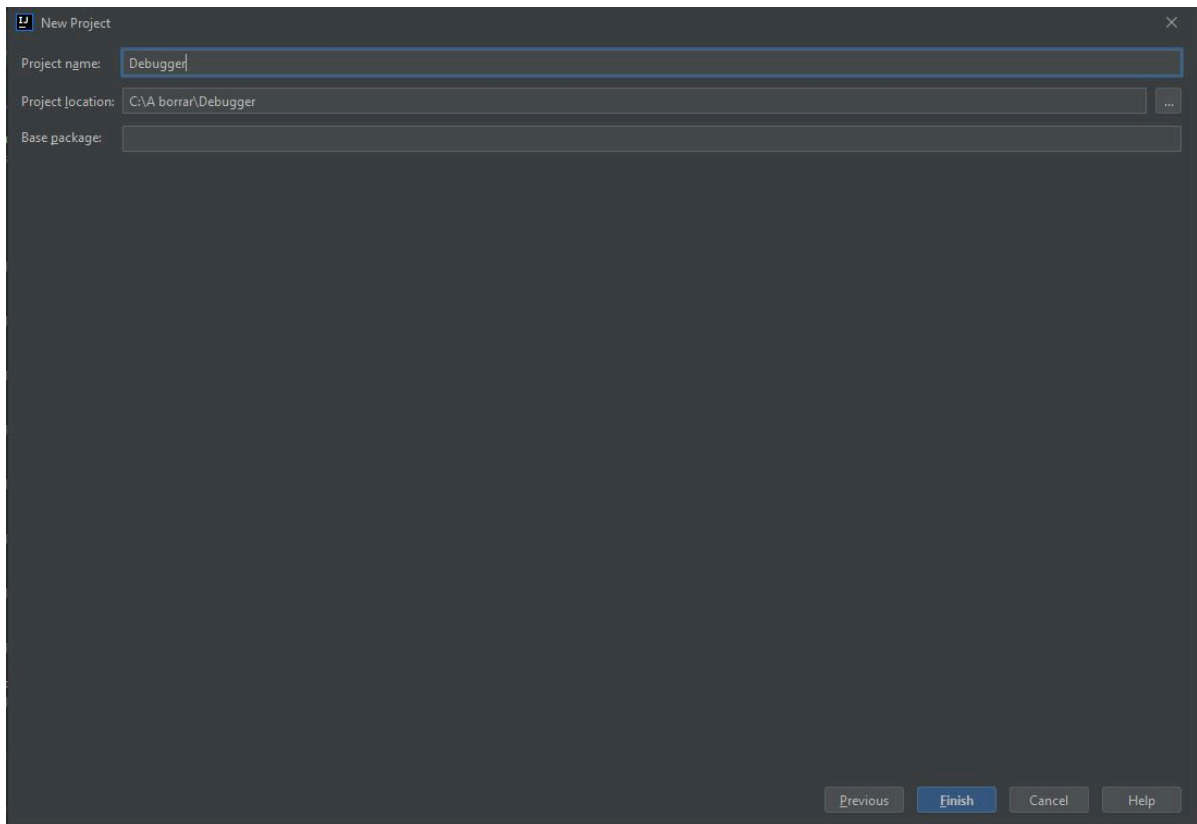
**IntelliJ** proporciona una amplia gama de herramientas para la depuración del código. Se puede encontrar una guía exhaustiva en el siguiente enlace: <https://www.jetbrains.com/help/idea/debugging-code.html>

En este CodeLab desarrollaremos un IP-Scanner para averiguar qué ordenadores que son accesibles.

## 2. Crea el proyecto

Inicia IntelliJ y crea un nuevo proyecto llamado "Debugger".





### 3. IP-Scanner

Para ver si un ordenador es accesible disponemos del método `isReachable()` de la clase `java.net.InetAddress`.

Su uso es muy sencillo, únicamente debemos invocar el método con la dirección IP del ordenador que queremos comprobar:

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Main {

    public static void main(String[] args) throws IOException {
        if(InetAddress.getByName("192.168.1.1").isReachable(3000)){
            System.out.println("El ordenador 192.168.1.1 está activo");
        }
    }
}
```

El valor 3000 es un parámetro que indica el tiempo que hay que esperar para ver si el ordenador al que estamos conectando responde. Si no responde en el tiempo indicado se considera que no es accesible.

😊 En lugar de utilizar la red 192.168.1.x averigua la dirección ip de tu ordenador y pone esa red.

Lo que queremos ahora es comprobar si son accesibles todos los ordenadores de la red: 192.168.1.1, 192.168.1.2, 192.168.1.3 ... hasta el 192.168.1.254

Si vamos copiando y pegando las instrucciones anteriores y cambiando la IP de forma manual, el programa funcionaría pero ... son demasiado ordenadores!

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Main {

    public static void main(String[] args) throws IOException {
        if(InetAddress.getByName("192.168.1.1").isReachable(3000)){
            System.out.println("El ordenador 192.168.1.1 está activo");
        }
        if(InetAddress.getByName("192.168.1.2").isReachable(3000)){
            System.out.println("El ordenador 192.168.1.2 está activo");
        }
        // ¡ Son muchos ordenadores !
        if(InetAddress.getByName("192.168.1.254").isReachable(3000)){
            System.out.println("El ordenador 192.168.1.254 está activo");
        }
    }
}
```

Si nos fijamos bien, estamos realizando la misma instrucción todo el tiempo, y lo único que cambia es el número de HOST de la IP.

Podemos tratar de extraer este patrón con un bucle:

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Main {

    public static void main(String[] args) throws IOException {
        int host = 1;
        while (host < 255) {
            if (InetAddress.getByName("192.168.1." + host).isReachable(3000)) {
                System.out.println("El ordenador 192.168.1." + host + " está activo");
            }
            host++;
        }
    }
}
```

Con este bucle pretendemos automatizar la ejecución de la instrucción aumentando en cada iteración el número de HOST.

Si ejecutamos este código, parecerá que no está haciendo nada, ya que no saldrá nada en la consola ... puede ser no hay ningún ordenador accesible en la red ... comprobémoslo.

## 4. Breakpoints

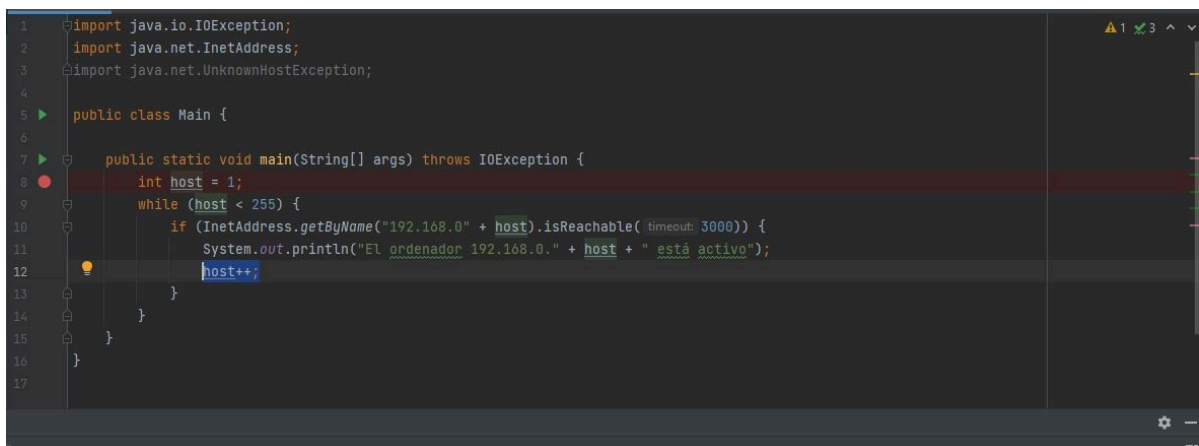
Normalmente podemos ir siguiendo la ejecución de un programa viendo que es lo que va mostrando la consola. En este caso no nos sirve de nada, ya que no muestra nada en la consola.

Una herramienta mejor para seguir la ejecución de un programa y ver en qué orden se están ejecutando las instrucciones es el depurador.

Los breakpoints son puntos en los que podemos detener la ejecución de un programa, y observar paso a paso su comportamiento.

Para añadir un breakpoint, basta con colocar el cursor sobre la línea donde queremos poner y apretar Ctrl + F8.


También lo podemos hacer con el ratón, haciendo clic sobre el panel gris que hay justo al lado del código:



Cuando añadimos un breakpoint, se indica con un **círculo rojo**.

## 5. Iniciar la depuración

Una vez añadido los breakpoints deseados, se puede iniciar la depuración de la aplicación.

Para iniciarla hacemos click en el icono Debug (  ) o pulsamos Ctrl + F9.

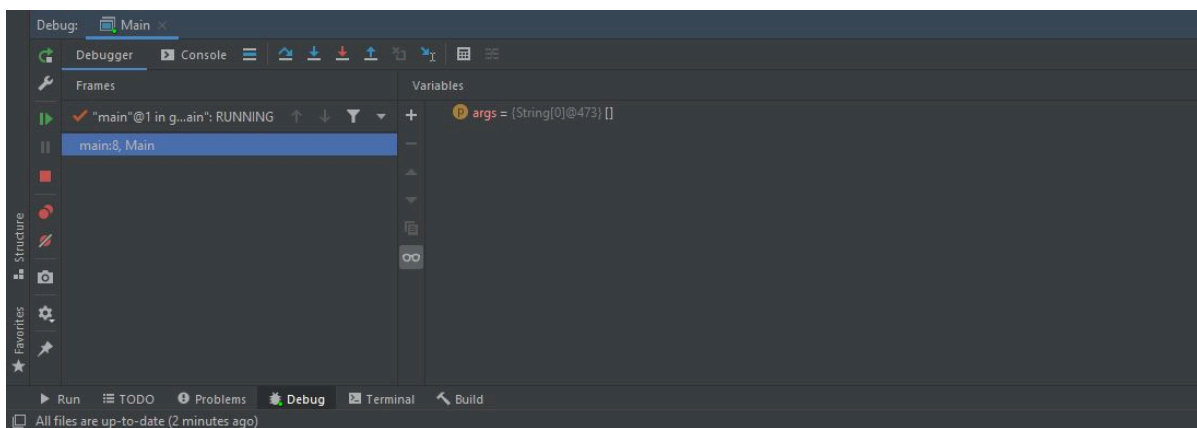
IntelliJ lanzará la sesión de depuración. Compilará y ejecutará la aplicación, suspendiéndose la al primer breakpoint. Podemos ver dos cosas que han cambiado:

La primera cosa que ha cambiado es el color de la primera línea con un breakpoint. Ahora se verá de color azul


```
1 import java.io.IOException;
2 import java.net.InetAddress;
3 import java.net.UnknownHostException;
4
5 public class Main {
6
7     public static void main(String[] args) throws IOException { args: []
8         int host = 1;
9         while (host < 255) {
10             if (InetAddress.getByName("192.168.0." + host).isReachable( timeout: 3000)) {
11                 System.out.println("El ordenador 192.168.0." + host + " está activo");
12                 host++;
13             }
14         }
15     }
16 }
17 }
```

Esto significa que la ejecución ha alcanzado el breakpoint y se ha parado antes de ejecutar la sentencia.

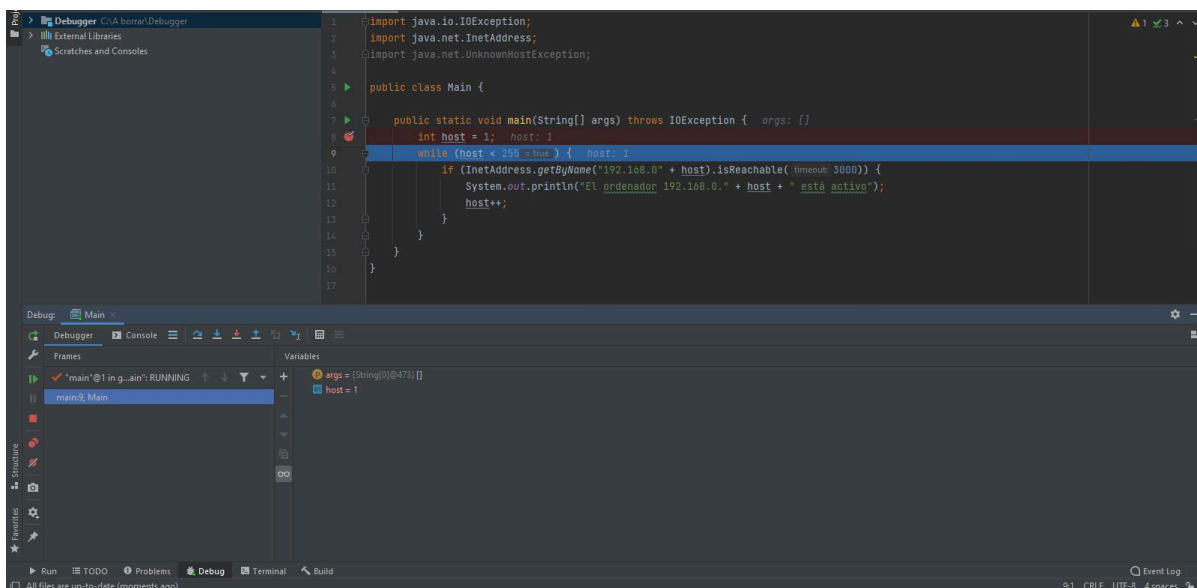
En la parte inferior de la ventana, ha surgido el Debug Tool Window, con mucha información sobre la ejecución del programa.




## 6. Ejecución paso a paso

Vamos a dar un paso en la ejecución del programa. Haz click sobre el icono  , o simplemente pulsad F8.

La siguiente instrucción se pone en azul:



Si vamos avanzando en la ejecución del programa con  , veremos que el programa se queda permanentemente en el bucle while y no entra nunca en el if.

Si miramos al panel Variables, veremos que la variable host siempre tiene el valor 2 (encuentra un equipo en mi red con ip 192.168.1.1, la incrementa dentro del if) , y que no va aumentando.

Podemos deducir que tenemos un error en el lugar donde hemos puesto la sentencia host ++, ya que como no entra nunca en el if, nunca aumenta el valor de esta variable. Corregimos esto:

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Main {

    public static void main(String[] args) throws IOException {
        int host = 1;
        while (host < 255) {
            if (InetAddress.getByName("192.168.0." + host).isReachable(3000)) {
                System.out.println("El ordenador 192.168.0." + host + " está
activo");
            }
            host++;
        }
    }
}
```

Si volvemos a hacer la ejecución paso a paso, veremos como la variable host sí va aumentando, de forma que cada vez comprueba si es accesible un ordenador diferente.

Sin embargo, sigue sin aparecer nada en la consola, como si no hubiera ningún ordenador accesible en la red ... Qué raro, al menos nuestro ordenador debería responder!

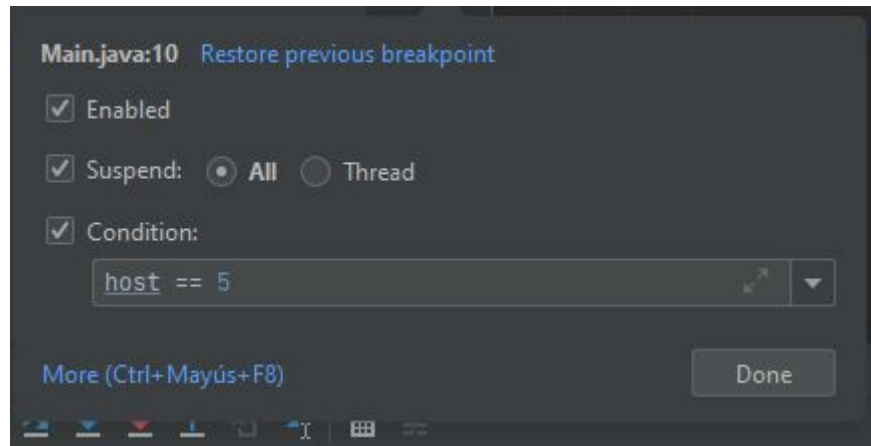
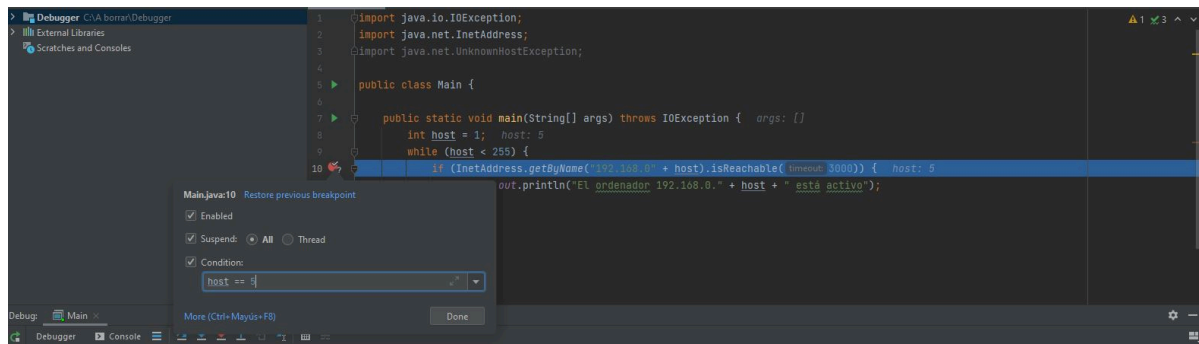
## 6. 1. Breakpoints condicionales

Podemos ir avanzando la ejecución del programa hasta que llegue al número de HOST de nuestro ordenador (o podríamos directamente hacer que la variable host empiece por este número). Pero hay otra forma de hacerlo.

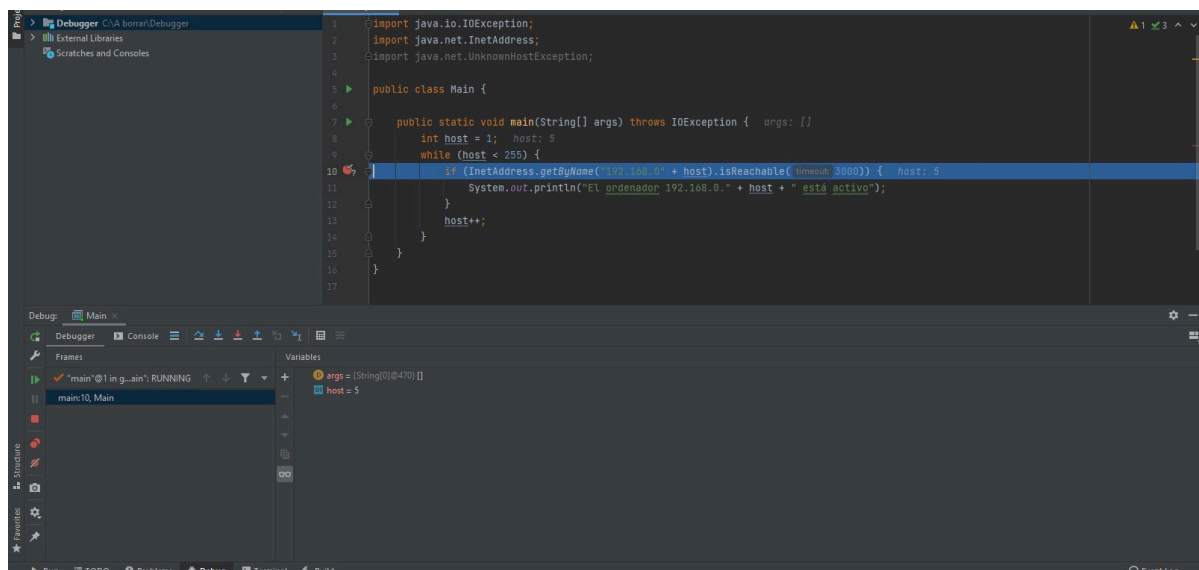
Podemos hacer que el programa se ejecute normalmente, sin paradas, hasta que la variable host llegue al número que deseamos, y entonces detener la ejecución y pasar al modo depuración.

Supongamos que nuestra dirección es 192.168.1.5. Entonces queremos pausar la ejecución cuando la variable host sea 5. O sea, podemos poner un breakpoint condicionado a que la variable host tenga el valor 1.

Para añadir un breakpoint condicional, hacemos click-derecha sobre el breakpoint y añadimos la condición deseada. En nuestro caso, nos interesa que se pare cuando el valor de host sea 5:



Si iniciamos la depuración, la ejecución del programa se detendrá en el breakpoint cuando se cumpla la condición `host == 5`.



¿Por qué llegados a este punto no está entrando en el if? ¿Por qué la instrucción `InetAddress.getByName ("192.168.1" + host) .isReachable (3000)` no está respondiendo si se trata de nuestro ordenador?

En este punto podemos tratar de ver si puede ser el error lo tenemos en la forma en que construimos la ip: `"192.168.1" + host`

## 6.2 Evaluar expresiones

Seleccionamos la expresión `"192.168.1" + host` y pulsamos `Alt + F8`. En la ventana que nos saldrá pulsar el botón Evaluate.





Podemos ver que la expresión "192.168.1" + host evalúa a "192.168.15". Claro, le falta un punto entre medio!

Con la información obtenida de la depuración del programa ya podemos terminar nuestro IP-Scanner:

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Main {

    public static void main(String[] args) throws IOException {
        int host = 1;
        while (host < 255) {
            if (InetAddress.getByName("192.168.0." + host).isReachable(3000)) {
                System.out.println("El ordenador 192.168.0." + host + " está
activo");
            }
            host++;
        }
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
```

```
Connected to the target VM, address: '127.0.0.1:56953', transport: 'socket'
```

```
El ordenador 192.168.0.1 está activo
```