

# Teoría JAVA I

## Debugger

El término "**debugger**" se utiliza para referirse a una herramienta o proceso que se emplea en la detección, análisis y corrección de errores en el código de un programa. Su funcionamiento se basa en detener el programa en puntos específicos definidos por el programador, lo cual le permite realizar las evaluaciones necesarias.

### Puntos de Interrupción (Breakpoints):

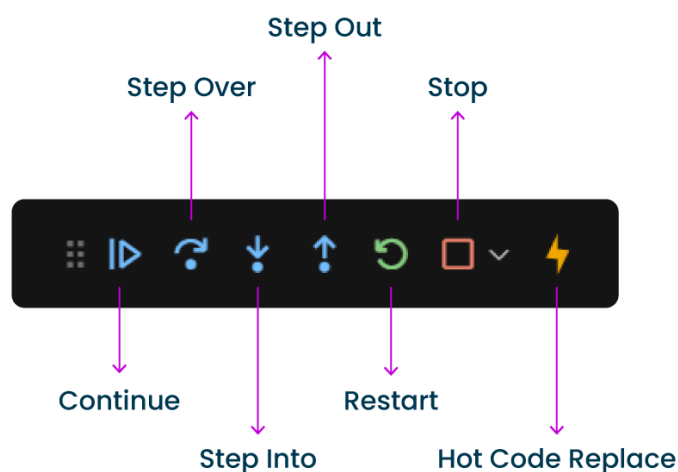
Los puntos de interrupción, también conocidos como **breakpoints**, son marcadores que se colocan en el código para detener la ejecución del programa en un **punto específico**. Cuando se alcanza un punto de interrupción, el debugger pausa la ejecución y brinda la oportunidad de examinar el estado de las variables, analizar el flujo del programa y realizar modificaciones si es necesario.

### Debugger y puntos de interrupción


### Pasos de Depuración:

Durante la depuración, tenemos la capacidad de avanzar paso a paso a través del código, lo que permite observar el comportamiento del programa línea por línea y analizar cómo se ejecuta cada instrucción.

En Visual Studio Code, esta funcionalidad se encuentra disponible de la siguiente manera:



- **Continue:** Permite reanudar la ejecución del programa después de haber detenido la depuración en un punto de interrupción. Es útil cuando se desea omitir una sección de código o cuando se ha corregido un problema y se quiere continuar sin detenerse en cada paso.
- **Step Over:** Avanza la ejecución del programa al siguiente punto de interrupción o a la siguiente línea de código. Si hay una llamada a un método en la línea actual, el depurador no ingresará en ese método y simplemente lo ejecutará en su totalidad, lo que significa que no se detendrá en cada instrucción dentro del método. Es útil cuando se quiere avanzar sin adentrarse en los detalles de los métodos.
- **Step Into:** Avanza la ejecución del programa al siguiente punto de interrupción o a la siguiente línea de código. Si hay una llamada a un método en la línea actual, el depurador ingresará en ese método y se detendrá en la primera instrucción dentro del método. Es útil cuando se quiere analizar y depurar el interior de los métodos llamados.
- **Step Out:** Avanza la ejecución del programa hasta que se sale del método actual en el que se encuentra depurando. Es útil cuando se desea volver rápidamente a la línea de código que invocó el método actual y omitir el resto del método.
- **Restart:** Reinicia la ejecución del programa desde el principio, lo que significa que se vuelven a cargar todos los archivos y se restablecen los valores de las variables. Es útil cuando se quiere comenzar la depuración desde el principio para verificar si los cambios realizados solucionaron el problema.
- **Stop:** Detiene por completo la ejecución del programa y finaliza la depuración. Es útil cuando se desea finalizar la depuración antes de que el programa alcance su finalización normal.
- **Hot Code Replace:** Permite recargar el código fuente para aplicar un cambio mientras el programa está en ejecución y probar esos cambios en tiempo real sin necesidad de reiniciar el programa. Es útil cuando se quieren realizar cambios rápidos en el código y ver los resultados inmediatamente.

Te invitamos a ver el siguiente video que te proporcionará una mejor comprensión sobre cómo utilizar estas opciones en la depuración del código:  [Debugger pasos](#)

## Watch y Expresiones:

La funcionalidad de "Watch" te brinda la capacidad de **monitorear y evaluar el valor de sentencias específicas** mientras tu programa se ejecuta. Además de monitorear variables existentes, también podemos utilizar expresiones en la sección de "Watch" para proporcionar información adicional sobre el estado del programa.

 [Debugger watches](#)

## Inspección de la Pila de Llamadas:

Utilizamos la inspección de la pila de llamadas para comprender el flujo del programa y cómo hemos llegado a un punto específico durante la depuración. Al examinar la pila de llamadas, podemos **ver qué métodos se han invocado y en qué orden**, lo que nos ayuda a identificar posibles causas de errores o comportamientos inesperados.

Te invitamos a ver el siguiente video para comprender mejor cómo utilizar esta herramienta:



[\*\*Debugger pila de llamadas\*\*](#)

## Control de Flujo de Excepciones:

En Visual Studio Code, contamos con las opciones *"Uncaught Exceptions"* y *"Caught Exceptions"* para controlar cómo el debugger responde a las excepciones que se producen en nuestro código:

- **Uncaught Exceptions:** Se refiere a aquellas *excepciones que se producen y no son capturadas ni manejadas por ningún bloque try-catch en nuestro código*. Al habilitar esta opción, el debugger detendrá la ejecución del programa en el punto donde se lanza la excepción y nos mostrará información detallada sobre la misma, como su tipo y mensaje. Esto nos permite investigar el origen del problema y depurar el código para corregir el manejo de la excepción.
- **Caught Exceptions:** Se refiere a las *excepciones que son capturadas y manejadas por bloques try-catch en nuestro código*. Al habilitar esta opción, el debugger detendrá la ejecución cuando se lance una excepción y se encuentre dentro de un bloque try-catch.

Te invitamos a ver el siguiente video para comprender mejor cómo utilizar estas opciones en el debugger de Visual Studio Code:



[\*\*Debugger control de excepciones\*\*](#)