Material descargable

egg

Teoría **JAVA VI**

¡Hola nuevamente! 👋

En esta ocasión, exploraremos el depurador (debugger) en programación.

Aprenderemos sobre *puntos de interrupción*, *pasos de depuración*, *watch* y *expresiones*, *inspección de la pila de llamadas*, y *control de flujo de excepciones*.

Estas herramientas nos permitirán identificar y corregir errores, analizar el flujo del programa y controlar el comportamiento de las excepciones.

Prepárate para mejorar tus habilidades de depuración y llevar tu programación al siguiente nivel.

¡Vamos a empezar! 🚀

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.

A continuación, brindaremos más información acerca de esta herramienta:

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.

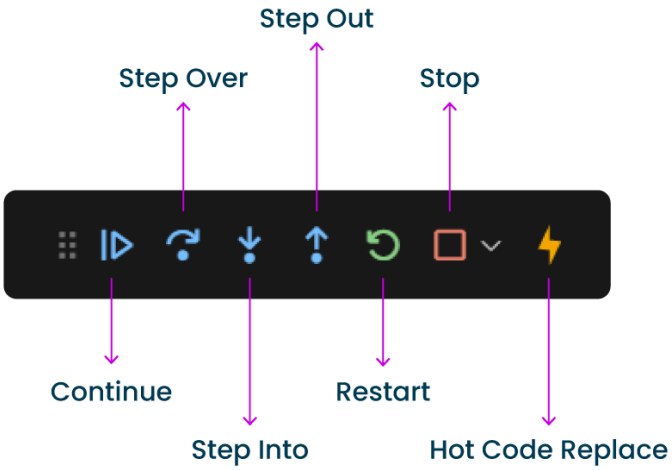
A continuación, te proporcionamos un video que ofrece una breve descripción del funcionamiento del debugger de Java en Visual Studio Code:

🎥 **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**. Podemos agregar variables a la sección de "Watch" para observar en tiempo real cómo sus valores cambian paso a paso durante la ejecución del programa. *Esto resulta*

especialmente útil cuando deseamos mantener un seguimiento cercano del valor de una variable en particular para identificar posibles problemas o comportamientos inesperados.

Además de monitorear variables existentes, también podemos utilizar expresiones en la sección de "Watch". *Las expresiones son combinaciones de código que se evalúan en tiempo real para proporcionar información adicional sobre el estado del programa.* Podemos emplear operadores, métodos, referencias a otras variables y métodos que hayamos creado para construir expresiones más complejas. Por ejemplo, podemos utilizar expresiones para realizar cálculos, imprimir mensajes o realizar comparaciones.

El uso de "Watch" y expresiones nos permite obtener una visión más detallada del comportamiento de tu programa mientras lo depuramos. Podemos observar cómo cambian los valores de las variables en diferentes etapas de la ejecución y verificar si se cumplen ciertas condiciones o realizar acciones específicas en función de esos valores.

Para comprender mejor cómo utilizar estas funcionalidades, te invitamos a ver el siguiente video:

🎥 **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. La pila de llamadas es una estructura de datos que registra el orden en el que se han realizado las llamadas a los métodos en nuestro programa.

Al examinar la pila de llamadas, *podemos ver qué métodos se han invocado y en qué orden*. Esto nos proporciona una visión secuencial de cómo se ha ejecutado nuestro programa hasta el punto actual. Cada llamada a un método se representa como un marco de pila, que contiene información sobre el método en sí, los parámetros que se han pasado y la ubicación en el código fuente.

La inspección de la pila de llamadas nos permite comprender mejor cómo se ha producido un error o un comportamiento inesperado. Podemos hacer clic en cada método de la pila para desplazarnos y ver el contexto en el que ha sido

invocado antes de llegar al punto problemático. Esto nos ayuda a identificar posibles causas y a seguir el flujo del programa para analizar el estado de las variables en cada nivel de la pila.

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 no detendrá la ejecución cuando se lance una excepción y sea capturada por un bloque try-catch. Esto puede resultar útil cuando estamos depurando una parte del código que sabemos que lanza excepciones, pero que están siendo manejadas adecuadamente y no necesitamos detener la ejecución cada vez que ocurran.

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**