UT07

Debuging en IntelliJ

vídeo

La depuración (debugging) en IntelliJ IDEA es una funcionalidad muy útil para identificar y corregir errores en tu código. Aquí te doy una guía paso a paso para empezar a depurar tu código en IntelliJ:

1. **Establecer puntos de interrupción (breakpoints)**

Los puntos de interrupción son lugares en tu código donde deseas que la ejecución se detenga para inspeccionar el estado del programa.

- Abre el archivo de código fuente que quieres depurar.
- Haz clic en el margen izquierdo junto al número de línea donde deseas establecer un punto de interrupción.
- Un círculo rojo aparecerá indicando que se ha establecido un punto de interrupción.

2. **Iniciar la depuración**

- Haz clic en el botón de depuración que parece un insecto verde (`Debug`) en la parte superior derecha de IntelliJ.
- Si tienes más de una configuración de ejecución, selecciona la correcta.
- Esto iniciará tu aplicación en modo de depuración y detendrá la ejecución en los puntos de interrupción que hayas establecido.

3. **Inspeccionar el estado del programa**

Cuando la ejecución se detiene en un punto de interrupción, puedes inspeccionar:

- **Variables locales**: En la ventana de depuración, verás los valores actuales de las variables locales.
- **Pila de llamadas** + (Call Stack): Esto muestra el rastro de llamadas de métodos que llevaron al punto donde se detuvo el programa.
- **Evaluar expresiones**: Puedes evaluar expresiones específicas haciendo clic derecho en una variable o expresión y seleccionando `Evaluate Expression`.

4. **Controlar la ejecución**

En el modo de depuración, tienes varios controles:

- **Step Over (F8)**: Salta a la siguiente línea, pero no entra en los métodos llamados.
- **Step Into (F7)**: Entra dentro del siguiente método llamado.
- **Step Out (Shift+F8)**: Sale del método actual y vuelve al llamador.
- **Resume (F9)**: Continúa la ejecución hasta el siguiente punto de interrupción.

5. **Finalizar la depuración**

- Cuando hayas terminado, puedes detener la depuración haciendo clic en el botón de `Stop` (cuadrado rojo) en la barra de depuración o cerrando la aplicación.

Consejos adicionales:

- Puedes agregar **puntos de interrupción condicionales** haciendo clic derecho sobre un breakpoint y seleccionando `Edit Breakpoint`, luego defines una condición específica.
- Utiliza la **vista de variables** para inspeccionar objetos complejos en detalle.

Ampliación sobre la pila de llamadas

La **pila de llamadas** (o *call stack*) en depuración es una estructura que muestra el rastro de las llamadas a métodos o funciones que han llevado a la ejecución del código en el punto donde se ha detenido, como un punto de interrupción.

¿Cómo funciona la pila de llamadas?

Cada vez que un método o función es llamado, se agrega a la pila de llamadas. Cuando el método termina su ejecución, se elimina de la pila. Esto te permite ver en qué lugar exacto se encuentra tu programa en cualquier momento y cómo llegó ahí.

¿Qué puedes hacer con la pila de llamadas en IntelliJ IDEA?

1. **Inspeccionar el flujo de ejecución:** La pila de llamadas muestra una lista de métodos que se han llamado, comenzando desde el más reciente (en la parte superior) hasta el primero que inició la cadena de llamadas.

2. **Navegar entre las llamadas:** Puedes hacer clic en cualquier nivel de la pila para ver el código en ese punto específico. Esto es útil si quieres retroceder y ver qué sucedió antes de llegar al punto actual.

3. **Detectar problemas:** Si tienes un error o comportamiento inesperado, la pila de llamadas te ayuda a encontrar el origen del problema. Por ejemplo, podrías ver si una llamada incorrecta a un método está causando un fallo.

Ejemplo típico de una pila de llamadas:

Supongamos que tienes un código simple en Java:

```
public class Main {
    public static void main(String[] args) {
        metodoA();
    }

    public static void metodoA() {
        metodoB();
    }

    public static void metodoB() {
        System.out.println("Debugging");
    }
}
```

Si pones un punto de interrupción en `System.out.println("Debugging");`, la pila de llamadas podría verse así:

```
metodoB() - Main.java:9
metodoA() - Main.java:5
main() - Main.java:3
```

Información clave en IntelliJ:

- **Nombre del método:** Por cada nivel de la pila de llamadas, verás el nombre del método que fue llamado.
- **Archivo y línea:** Junto al nombre del método, se muestra el archivo y la línea de código donde ocurrió la llamada.
- **Contexto de la clase/objeto:** Si es un método de instancia, puede mostrar el objeto en el que se llamó el método.

¿Cuándo es útil la pila de llamadas?

- **Seguimiento de errores (exceptions):** Si ocurre una excepción en tu código, la pila de llamadas te mostrará exactamente dónde y cómo ocurrió.
- **Depuración de flujos complejos:** Cuando un error proviene de una secuencia de llamadas entre múltiples clases y métodos, la pila de llamadas te ayuda a rastrear el origen del problema.
- **Depurar métodos recursivos:** Si tienes un método recursivo, la pila de llamadas te muestra cuántas veces ha sido llamado.