Entornos de Desarrollo

Unidad 2 Tarea 2:

Uso del Debugger

En esta actividad llevaremos a cabo ciertos ejercicios relacionados con el debugger en el entorno de NetBeans, configurando puntos de ruptura condicionales y de excepción.

¿Qué es un IDE? ¿Qué debe tener un software para que sea considerado un IDE como mínimo?

Las siglas de ***IDE*** significan Integrated Development Environment o "***Entorno de Desarrollo Integrado***". Se trata de un software que proporciona herramientas integradas para facilitar el desarrollo de programas informáticos. No todo software de desarollo es automáticamente un IDE, pues deben tener como mínimo un ***editor de código*** (con funciones de autocompletado, resaltado de sintaxis y a veces sugerencias contextuales), un ***compilador*** o, dependiendo del lenguaje de programación, ***intérprete*** que traduce el código a un formato ejecutable y un ***depurador o debugger*** para detectar, diagnosticar y corregir errores en el código durante su ejecución.

Además de los componentes básicos, muchos IDE ofrecen funciones adicionales que mejoran la experiencia de desarrollo:

- **Integración con sistemas de control de versiones** (como Git).
- **Soporte para múltiples lenguajes de programación**.
- **Refactorización automática** del código.
- **Visualización del flujo o estructura del proyecto** (gestión de archivos, diagramas de clases, etc.).
- **Herramientas de pruebas** (unidades o integración).
- **Integración con plataformas en la nube** o contenedores.
- **Terminal integrada** para ejecutar comandos sin salir del entorno.
- **Plugins y extensiones** para añadir funcionalidades adicionales.

2 ¿Es el Visual Studio Code (VSC) un IDE? ¿Qué es? Instale VSC para probarlo, lo usaréis mucho durante el grado ¿Por qué no es un IDE?

Aunque puede configurarse para actuar como tal, no es por defecto un IDE, sino un editor de código fuente avanzado. Es un software altamente extensible que puede ser configurado para funcionar como un IDE gracias a su sistema de extensiones, pero en su estado base, no incluye algunas de las herramientas mínimas que se asocian con un IDE completo. Aunque tiene muchas funcionalidades avanzadas, no cumple con todos los requisitos de un IDE sin agregar extensiones adicionales. Por ejemplo:

1. **Falta de un compilador/intérprete nativo:**

VSC no incluye de manera predeterminada un compilador o intérprete. Es necesario instalar y configurar herramientas externas (como GCC, Node.js o Python) para compilar o ejecutar código.

2. **Depurador limitado:**

El depurador de VSC funciona solo después de configurar extensiones específicas para el lenguaje que estás usando.

3. **Falta de herramientas de gestión de proyectos nativas:**

Aunque puedes organizar archivos en carpetas, no incluye características avanzadas de gestión de proyectos (como diagramas de clases, vistas jerárquicas de dependencias, etc.) propias de IDEs como IntelliJ IDEA o Eclipse.

4. **Personalización requerida:**

Para funcionar como un IDE completo, necesitas instalar y configurar extensiones específicas para el lenguaje de programación o las herramientas que deseas usar.

3 ¿Es Visual Studio un IDE? ¿Qué lo diferencia de VSC?

Sí lo es, y está diseñado para soportar proyectos de desarrollo más complejos, especialmente aquellos que requieren herramientas avanzadas de gestión de proyectos, depuración, compilación y pruebas.

Sus diferencias principales son:

IDE completo diseñado para proyectos grandes y complejos.

Incluye compiladores, depuradores avanzados, herramientas de pruebas y más.

Especialmente fuerte en C#, .NET, C++, y Visual Basic.

Más pesado y requiere más recursos del sistema.

Ideal para aplicaciones empresariales y proyectos a gran escala.

Soporta extensiones, pero es menos flexible que VSC en personalización.

Puede ser gratuito (*Community Edition*), pero las ediciones profesionales tienen costo.

Soporte nativo para gestionar soluciones completas (con múltiples proyectos dentro).

4 ¿Se puede considerar ChatGPT o una IA como un entorno de desarrollo? ¿Por qué?

Chat GPT es como mucho una herramienta complementaria, pues no incluye ninguna de las herramientas mencionadas anteriormente (editor, compilador y depurador) además de no tener la capacidad de ejecutar el código o gestionar proyectos. Además, tampoco puede depurar programas en ejecución, establecer puntos de interrupción (*breakpoints*) o analizar el flujo del programa en tiempo real.

5 Explique los tres tipos de errores que hay en un código, sintáctico, de ejecución y lógicos. Ponga un ejemplo de cada uno de ellos.
¿Qué tipo de errores nos ayuda a detectar y corregir el depurador?

Un error **sintáctico** ocurre cuando el código no cumple con las reglas gramaticales del lenguaje de programación. Estos errores son detectados por el compilador o intérprete antes de ejecutar el programa. Estos errores se detectan durante la fase de compilación o interpretación. El depurador no interviene porque el programa no puede ejecutarse.

```
print("Hola, mundo'
```

Un error **de ejecución** ocurre cuando el programa intenta realizar una operación que no es válida en tiempo de ejecución, como dividir por cero o acceder a un índice fuera de rango. Estos errores solo se detectan cuando el programa se está ejecutando. El depurador puede ayudar a identificar la línea específica donde ocurrió el error y las condiciones que lo provocaron.

```
numerador = 10
denominador = 0
resultado = numerador / denominador
```

Problema: Dividir por cero. **Mensaje típico:** *ZeroDivisionError: division by zero*.

Un error **lógico** ocurre cuando el programa se ejecuta sin fallos, pero produce un resultado incorrecto debido a un fallo en la lógica del código. Estos son los más difíciles de detectar porque no generan mensajes de error. El depurador puede ayudar a analizar las variables en tiempo de ejecución, pero la corrección depende del análisis del programador.

```
def calcular_area(base, altura):
    return base + altura  # Debe ser (base * altura) / 2

print(calcular_area(5, 10))
```

(El depurador ayuda a detectar y corregir errores de ejecución y lógicos, pero no ayuda directamente con los errores sintácticos, ya que estos impiden que el programa se ejecute y, por tanto, el depurador no puede iniciarse)

6 ¿Qué es un punto de ruptura o breakpoint? ¿Puede ser de tipo condicional? ¿Qué es un breakpoint de excepción?

Un punto de ruptura (*breakpoint*) es una herramienta utilizada en depuración que permite detener la ejecución de un programa en una línea específica del código. Esto permite al programador inspeccionar el estado del programa en ese momento, como el valor de las variables, el flujo de ejecución, y el entorno general, para identificar y solucionar errores.

Un breakpoint condicional detiene la ejecución del programa solo si se cumple una condición específica. Esto es útil cuando se quiere depurar solo en casos particulares y evitar detenerse en todas las iteraciones o ejecuciones.

```
for i in range(100):
    if i == 50:  # Breakpoint condicional: solo se activa si i == 50
        print("Detenido aquí")
```

Un **breakpoint de excepción** detiene la ejecución del programa automáticamente cuando ocurre una excepción, independientemente de dónde se produzca en el código. Esto es especialmente útil para detectar errores de ejecución o manejar excepciones inesperadas.

En algunos IDEs (como Visual Studio o PyCharm), puedes establecer un breakpoint de excepción para que el depurador se detenga cuando ocurra cualquier tipo de excepción, o puedes especificar un tipo de excepción particular (por ejemplo, detenerse solo en `ValueError`).

Práctica Debugger:

```
public static void main(String[] args) {
    // TODO code application logic here

    int num = 79833600;

    int res;

    res = num;

    for (int i = 11; i>-3; i--)

    {

    res = res / i;
    System.out.print("");

    int total = 0;
```

```
for (i = 1; i < 3000; i++)
{
    total = total + i + 1;

    System.out.print(total);
    System.out.print(" ");
```

Estudie y compile los siguientes códigos, elija una línea por ejemplo en la variable res para crear un breakpoint, en la línea de la división, cree un breakpoint condicional (para i>0, i>3, i<6, i==6, i==0, i<0, i<-1, i>250) de prueba, verá que aparece una mueca en el cuadro rojo del breakpoint. Describa qué sucede ¿Qué tipo de error se produce aquí?

i > 0: El comportamiento que se desea es que se detenga en cada iteración del bucle mientras `i` sea mayor que 0.

El resultado es que no ocurren errores hasta que `i <= 0`.

`i > 3`: El comportamiento que se desea es que se detenga en las iteraciones con `i` en el rango de 4 a 11.

El resultado es que no ocurren errores.

`i < 6:` El comportamiento que se desea es que se detenga en las iteraciones con `i` en el rango de -2 a 5.

El resultado es posible error cuando `i == 0` (división por cero)

`i == 6:` El comportamiento que se desea es que se detenga cuando `i` sea igual a 0.

El resultado es error de ejecución: División por cero (*ArithmeticException*).

`i < 0:` El comportamiento que se desea es que se detenga en las iteraciones con `i` en el rango de -1 a -2.

El resultado es error lógico: Valores negativos en la división pueden producir resultados inesperados.

1 < -1: El comportamiento que se desea es que se detenga solo en la última iteración donde `i == -2.`

El resultado es sin errores, pero puede ser innecesario.

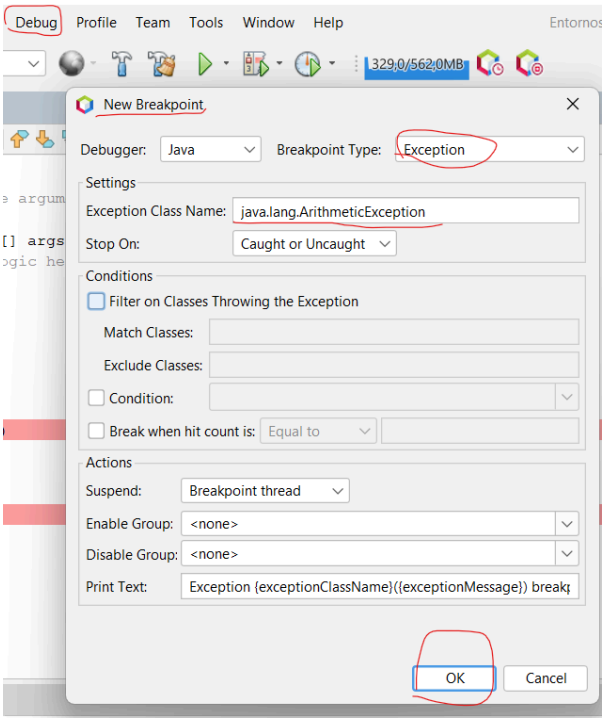
i > 250: El comportamiento que se desea es que nunca se detenga ya que la condición no se puede cumplir.

El resultado es que el breakpoint no se activa.

¿Qué utilidad tiene un breakpoint condicional una vez realizada la práctica anterior? Además cree también un breakpoint de excepción, tenga en cuenta que este tipo de breakpoints no están ligados a una línea en concreto. Si se produce la excepción el programa se detendrá antes de ejecutar dicha línea, para crearlo se va al menú debugger, new breakpoint y en type se puede poner exception.

La utilidad de realizar breakpoints condicionales es la de ***filtrar iteraciones que consideremos relevantes*** (detener la ejecución solo cuando se cumplen ciertas condiciones específicas. Esto evita que el depurador se detenga innecesariamente en cada iteración, ahorrando tiempo), ***analizar estados específicos***, ***detectar posibles escenarios problemáticos*** (enfocarte en situaciones particulares que podrían causar errores, como divisiones por cero, desbordamientos, o valores inesperados de variables) o ***evitar cambios en el código fuente***.

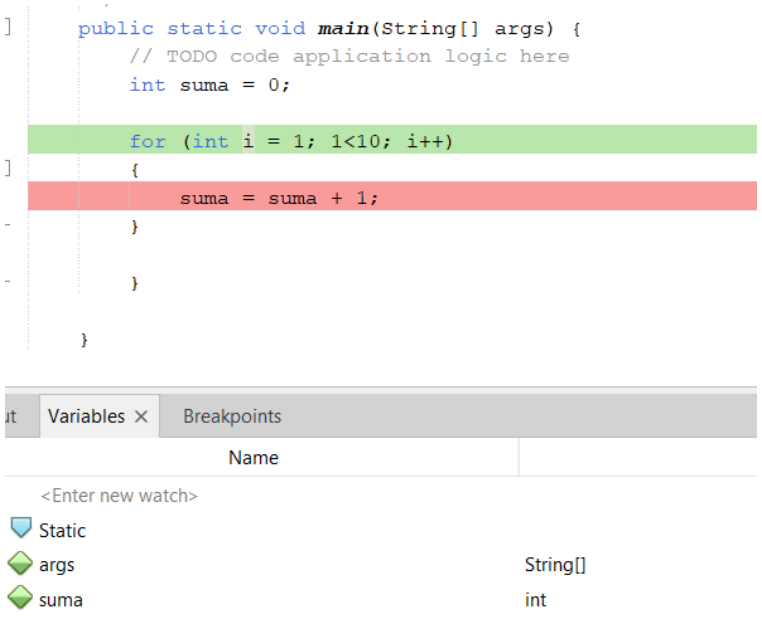
A continuación se muestra un breakpoint de excepción;



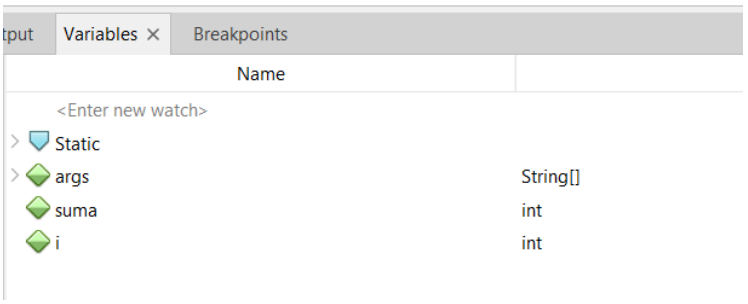
En este código, ¿qué tipo de error se produce?. Cree dos puntos de ruptura y depure el código con el F8, paso a paso y explique cuál puede ser el error. ¿Cómo lo corregirías?

```java
public class Entornos {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int suma = 0;

        for (int i = 1; 1<10; i++)
        {
            suma = suma + 1;
        }

        }

    }
```

El programa intentará ejecutar el bucle de forma infinita, incrementando la variable `i` pero nunca terminando el ciclo. Este es un error lógico porque la condición no depende de la variable de control del bucle (`i`), sino de un valor constante.



```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int suma = 0;

        for (int i = 1; 1<10; i++)
        {
            suma = suma + 1;
        }

        }

    }
```



La condición en el bucle `for` debe ser `i < 10` en lugar de `1 < 10`, para que el bucle dependa de la variable `i` y termine correctamente cuando `i` alcance el valor 10.

```
        for (int i = 1; i<10; i++)
        {
            suma = suma + 1;
        }
        System.out.println(suma);
        }

    }
```