



UD 2

Uso de estructuras de control

RESULTADOS DE APRENDIZAJE Y CRITERIOS DE EVALUACIÓN



RA3. Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje.

Criterios de evaluación

- a) Se ha escrito y probado código que haga uso de estructuras de selección.
- b) Se han utilizado estructuras de repetición.
- c) Se han reconocido las posibilidades de las sentencias de salto.
- d) Se ha escrito código utilizando control de excepciones.
- e) Se han creado programas ejecutables utilizando diferentes estructuras de control.
- f) Se han probado y depurado los programas.
- g) Se ha comentado y documentado el código.

INDICE



1. INTRODUCCIÓN

2. ESTRUCTURAS DE SELECCIÓN

- 2.1 IF – SENTENCIA CONDICIONAL SIMPLE
- 2.2 IF – SENTENCIA CONDICIONAL COMPUESTA
- 2.3 IF – ANIDACIÓN
- 2.4 EL OPERADOR TERNARIO ?:
- 2.5 SWITCH

3. ESTRUCTURAS DE REPETICIÓN

- 3.1 WHILE
 - 3.1.1 BUCLES CON CONTADOR
 - 3.1.2 BUCLES CON CENTINELA
- 3.2 DO WHILE
- 3.3 FOR
- 3.4 BUCLES ANIDADOS

4. ESTRUCTURAS DE SALTO

5. CONTROL DE EXCEPCIONES

6. PRUEBAS Y DEPURACIÓN DE PROGRAMAS

- 6.1 HERRAMIENTAS DE DEPURACIÓN
- 6.2 DEPURACIÓN DE CÓDIGO EN NETBEANS (DEBUGGER)
- 6.3 EJEMPLO DE DEPURACIÓN DE CÓDIGO EN NETBEANS

7. DOCUMENTACIÓN Y COMENTARIO DE CÓDIGO

8. ANEXOS

- 8.1 GENERACIÓN DE NÚMEROS ALEATORIOS



6. PRUEBAS Y DEPURACIÓN DE PROGRAMAS

6. Pruebas y depuración de programas ...



6.1 Herramientas de depuración

Las **pruebas** constituyen una de las tareas del ciclo de vida del software y tienen como **objetivo fundamental la detección de defectos** que se hayan podido cometer inadvertidamente durante el proceso de construcción del software.

Ya sea una vez que se realicen dichas pruebas de forma exitosa (es decir, cuando se localiza un defecto o fallo del programa tras la realización de las pruebas), ya sea durante el proceso mismo de creación de los programas, es útil disponer de herramientas que permitan **reconocer la localización exacta del defecto que provoca un mal funcionamiento del software** para poder corregirlo. A dicho proceso de localización y corrección del defecto se le conoce con el nombre de **depuración** – *debug* en inglés - y los IDE disponen de herramientas para ayudar a hacerlo.

6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Que es una herramienta de depuración

La depuración es una herramienta de análisis de programas potente que tiene un extenso uso por los desarrolladores de software. Un depurador permite ir paso a paso a través de cada línea de código en un programa, con lo cual puedes rastrear la ejecución y descubrir los errores. También puede mostrar el contenido de la memoria, los valores de las variables y las direcciones, así como registrar el contenido. El entorno de desarrollo integrado NetBeans tiene un depurador que puedes utilizar para depurar aplicaciones Java.

Que permite esta herramienta

- Ejecutar el código fuente paso a paso
- Ejecutar métodos del JDK paso a paso.
- Utilizar breakpoint para detener la ejecución del programa y poder observar el estado de las variables.
- Conocer el valor que toma cada variable o expresión según se van ejecutando las líneas de código.
- Modificar el valor de una variable sobre la marcha y continuar la ejecución.

Que nos facilita

- Encontrar de forma rápida y fácil errores en el programa.
- Entender mejor el flujo de ejecución del programa.

6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Opciones principales de esta herramienta

Antes de iniciar la ejecución del **debugger**, es recomendable crear un **punto de ruptura (breakpoint)**

- ¿Qué es un breakpoint? los breakpoints (puntos de interrupción o puntos de ruptura) son marcas o detenciones en el código, determinadas a través de las herramientas que ofrece el entorno, que sirven para ordenar al depurador que se detenga al llegar a un punto de la ejecución.
- ¿Para qué se utiliza el breakpoint? al detenerse en el punto seleccionado, el programa permite examinar los valores actuales de las variables, detectar cuando se crea un objeto y continuar la depuración línea a línea del programa. De esta manera, línea a línea, el código se va depurando.

¿Cómo se crea un breakpoint?

Tienes varias opciones:

- **Clic en el margen izquierdo.** Pinchando sobre el número (izq.) de la línea donde se quiere fijar. La línea del código quedará resaltada en rojo, y el número será sustituido por un cuadrado del mismo color.
- Menú contextual > "Ocultar/Mostrar línea de punto de interrupción"
- Pulsando la combinación de teclas: Ctrl +Mayúsc+ F8
- Menú "Depurar > Ocultar/Mostrar línea de punto de interrupción"

6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

```
13 public class MiClase {
14
15     public static void main(String[] args) {
16         try {
17             Scanner teclado = new Scanner(System.in);
18             int numero1;
19             int numero2;
20             int resultado=0;
21             System.out.println("Introduzca el primer número");
22             numero1 = teclado.nextInt();
23             System.out.println("Introduzca el segundo número");
24             numero2 = teclado.nextInt();
25             if (numero1 != 0 && numero2 != 0) {
26                 resultado = suma_enteros(numero1, numero2);
27             }
28             System.out.println("El resultado es " + resultado);
29         } catch (Exception e) {
30             System.out.println("se ha producido un error");
31         }
32     }
33
34     private static int suma_enteros(int a, int b) {
35         int resultado = a + b;
36         return resultado;
37     }
38 }
```

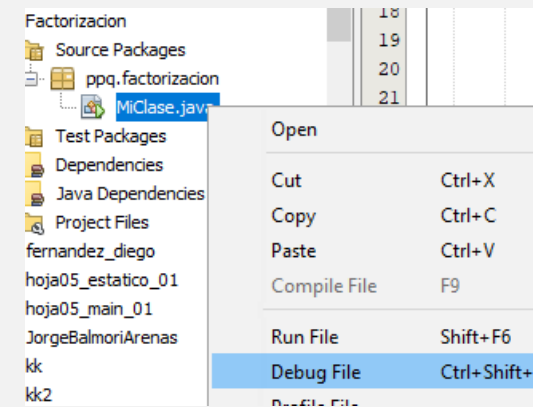
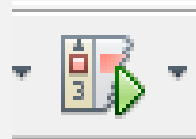

6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Una vez establecido al menos un punto de interrupción, se debe ejecutar la aplicación en modo depuración. Esto se puede llevar a cabo **sobre el proyecto** o sólo sobre el archivo actual:

- Depurar archivo actual:
 - Menú contextual > "Debug nombreArchivo"
 - Menú "Depurar > Debug nombreArchivo"
 - Pulsando la combinación de teclas: Ctrl + Mayúsculas + F5
- Depurar proyecto:
 - Menú "Depurar > Debug Main Project"
 - Icono "Debug Main Project"

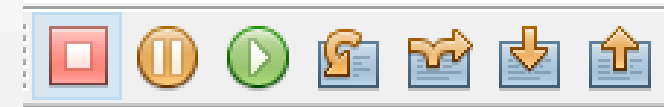


Elegida una opción de ejecución, se destaca en **color verde** la línea de código en la que se encuentre la ejecución. Comienza la depuración desde la primera línea del método main. El depurador se detiene esperando que decidamos el modo de depuración.

6. Pruebas y depuración de programas ...

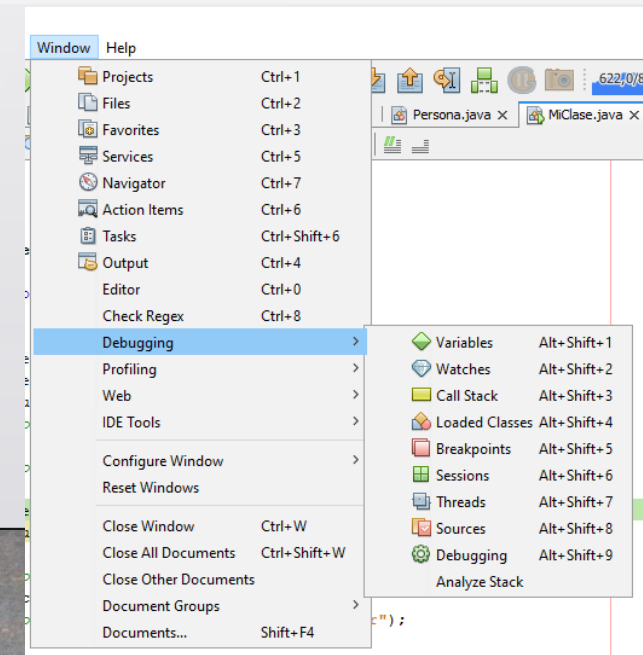


6.2 Depuración de código en Netbeans (debugger) ...



Aparece una nueva barra de opciones en la parte superior. También podemos mostrar una serie de ventanas, todas las que figuran en la siguiente opción que explicaremos a continuación.

```
1 public class MiClase {
2
3     public static void main(String[] args) {
4         try {
5             Scanner teclado = new Scanner(System.in);
6             int numero1;
7             int numero2;
8             int resultado=0;
9             System.out.println("Introduzca el primer número");
10            numero1 = teclado.nextInt();
11            System.out.println("Introduzca el segundo número");
12            numero2 = teclado.nextInt();
13            if (numero1 != 0 && numero2 != 0) {
14                resultado = suma_enteros(numero1, numero2);
15            }
16            System.out.println("El resultado es " + resultado);
17        } catch (Exception e) {
18            System.out.println("se ha producido un error");
19        }
20    }
21
22    private static int suma_enteros(int a, int b) {
23        int resultado = a + b;
24        return resultado;
25    }
26 }
```

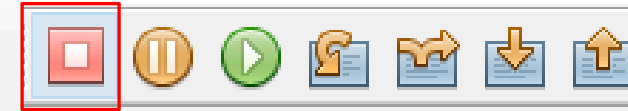


6. Pruebas y depuración de programas ...

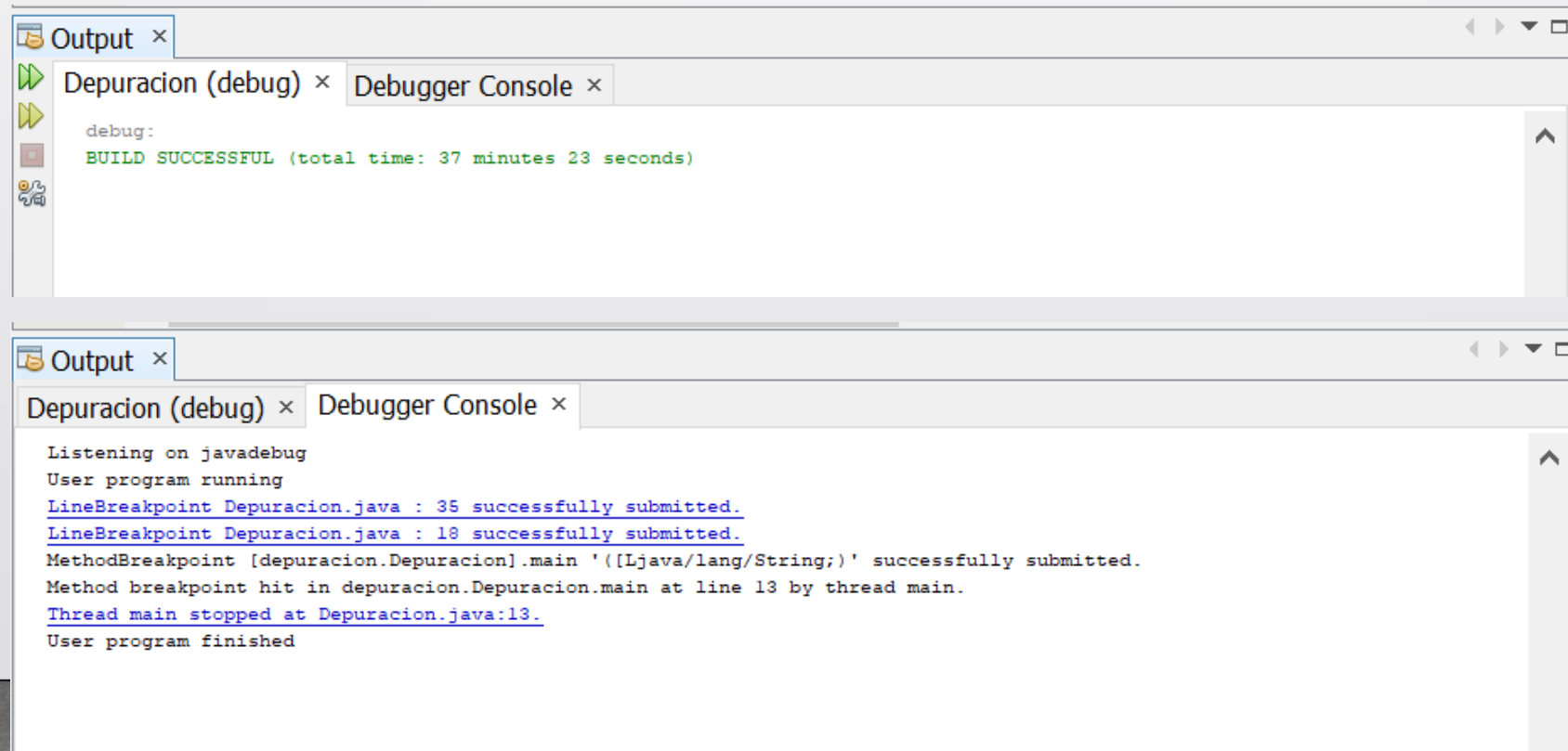


6.2 Depuración de código en Netbeans (debugger) ...

Botonera de la barra de herramientas:



- **1. Finish Debugger Session (Mayúsculas + F5).** Termina la depuración del programa:

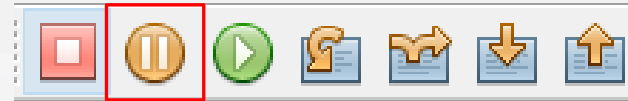


6.2. Pruebas y depuración de programas ...

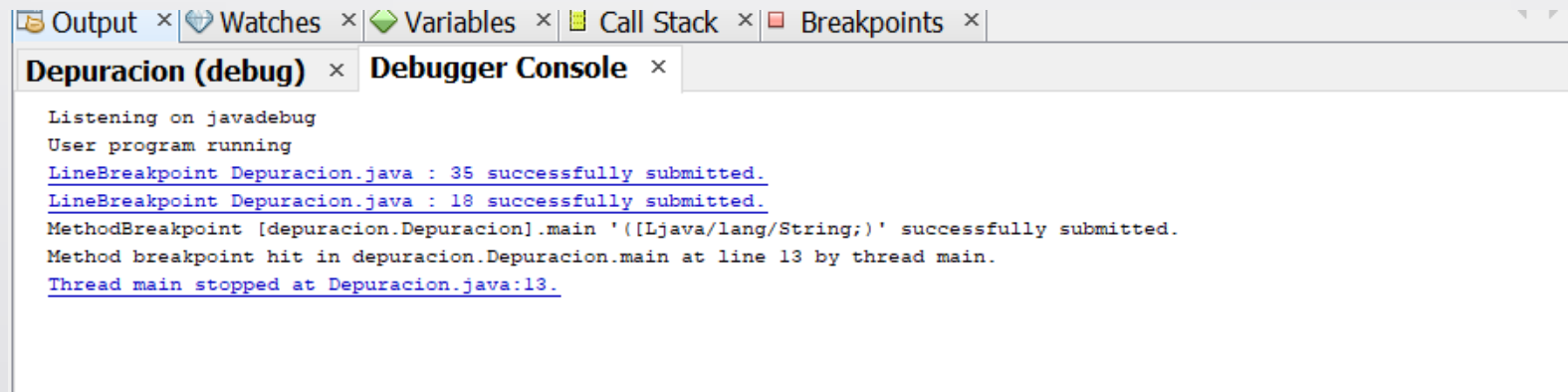


6.2 Depuración de código en Netbeans (debugger) ...

Botonera de la barra de herramientas:



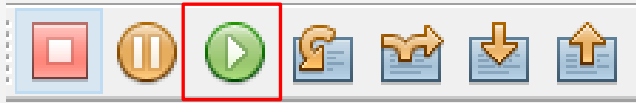
2. Pause. Pausa la depuración del programa.



Una pausa finaliza con cualquiera de las opciones siguientes.

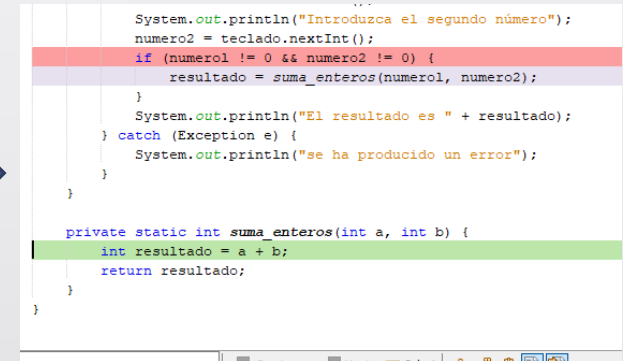
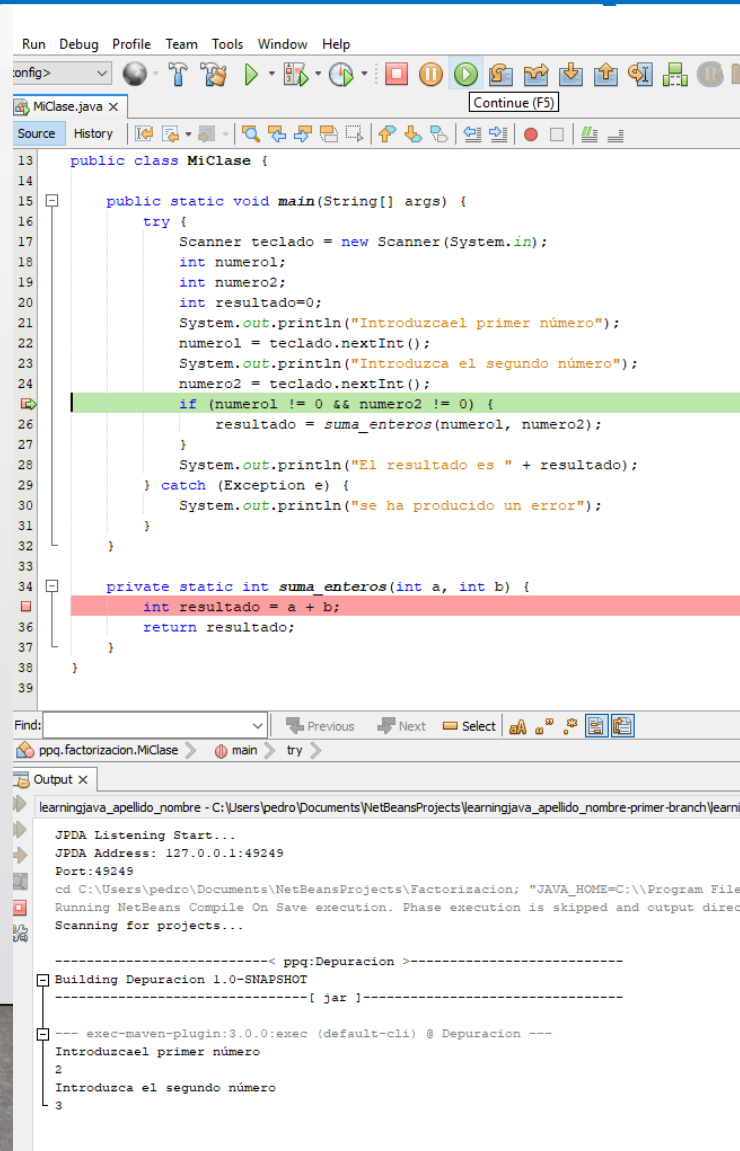
6. Pruebas y depuración de programas ...

6.2 Depuración de código en Netbeans (debugger) ...



Botonera de la barra de herramientas:

3. Continue (F5). La ejecución del programa continúa hasta el siguiente breakpoint. Si no existe un breakpoint se ejecuta hasta el final. En el ejemplo, dependiendo donde realicemos el breakpoint, si el programa requiere antes la intervención del usuario se para antes y una vez introducidos los valores continúa hasta el breakpoint.



6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...



Botonera de la barra de herramientas:

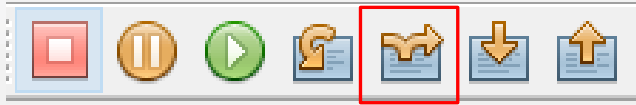
4. Step over (F8). Ejecuta una línea de código. Si la instrucción es una llamada a un método, ejecuta el método sin entrar dentro del código del método. No obstante, si hemos introducido algún punto de ruptura dentro de ese método, en ese caso si entra (es el caso de nuestro ejemplo) y se pausa, pudiendo salir o realizar otro Step Over, que en este caso imprimiría el resultado y finalizaría el programa, ya que no quedan más líneas que ejecutar.



```
System.out.println("Introduzca el segundo número");
numero2 = teclado.nextInt();
if (numero1 != 0 && numero2 != 0) {
    resultado = suma_enteros(numero1, numero2);
}
System.out.println("El resultado es " + resultado);
} catch (Exception e) {
    System.out.println("se ha producido un error");
}

private static int suma_enteros(int a, int b) {
    int resultado = a + b;
    return resultado;
}
```


6.2 Depuración de código en Netbeans (debugger) ...



Botonera de la barra de herramientas:

5. Step over Expression (Mayús +F8). Permite continuar con cada llamada del método en una expresión y ver los parámetros de entrada, así como los valores de salida resultantes de cada llamada de método. Si no hay más llamadas de método, Step Over Expression se comporta como el Step Over.

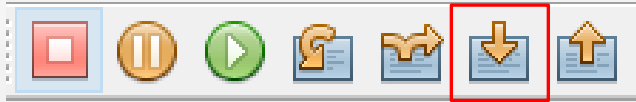
En el ejemplo, si lo volvemos a pulsar se detendrá en el método print, de la misma expresión.

[illegible]

6. Pruebas y depuración de programas ...



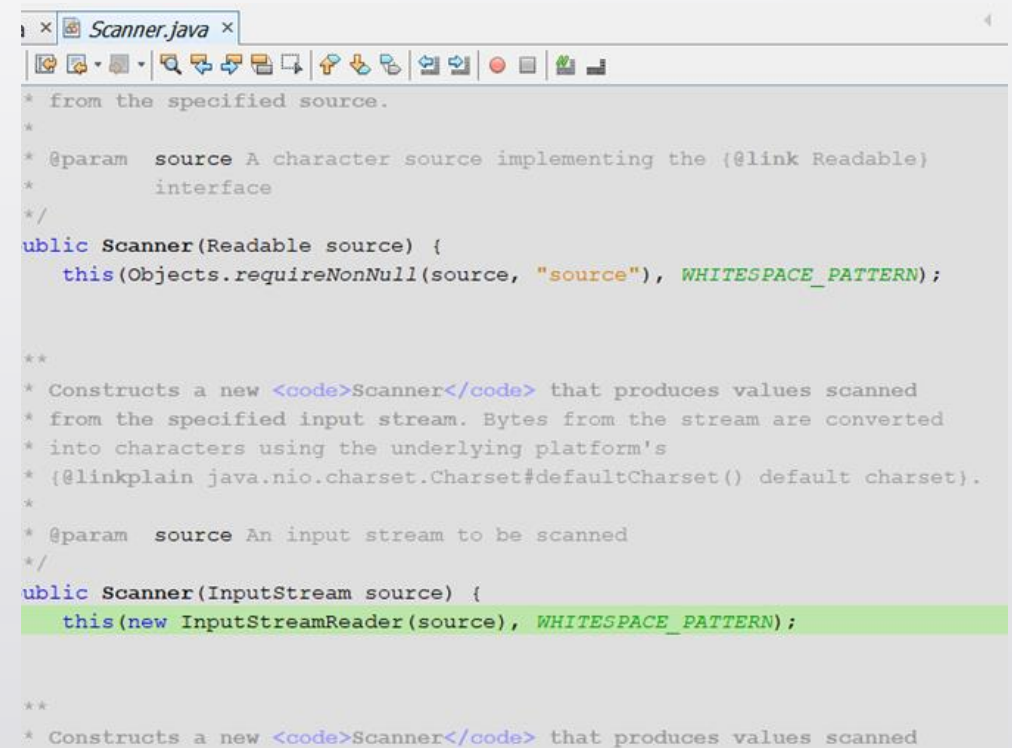
6.2 Depuración de código en Netbeans (debugger) ...



Botonera de la barra de herramientas:

6. Step Into (F7) Ejecuta una línea de código. Si la instrucción es una llamada a un método, salta al método y continúa la ejecución por la primera línea del método.

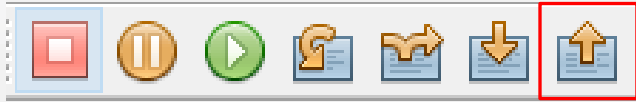
Ejemplo: comienza la ejecución del debugger, situado en la clase Scanner y Realizamos un Step into, entra dentro del método de la clase Scanner que se ejecuta.



6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...



Botonera de la barra de herramientas:

7. Step Out (Ctrl+F8). Ejecuta una línea de código.

Si la línea de código actual se encuentra dentro de un método, se ejecutarán todas las instrucciones que queden del método y se vuelve a la instrucción desde la que se llamó al método.

Siguiendo con el ejemplo anterior, que estábamos dentro de la clase Scanner, si pulsamos step out regresamos de nuevo a la instrucción desde la que se llamó.

```
15 public static void main(String[] args) {  
16     try {  
17         Scanner teclado = new Scanner(System.in);  
18         int numero1;  
19         int numero2;  
20         int resultado=0;
```

6. Pruebas y depuración de programas ...

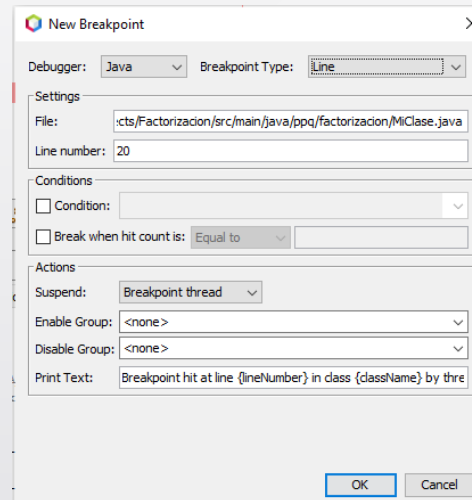


6.2 Depuración de código en Netbeans (debugger) ...

Otras opciones

- **New Breakpoint (Ctrl+mayús+F8).** Nos permite crear Breakpoint de manera más compleja. Nos permite crear breakpoints de los siguientes tipos:
 - Class
 - Exception
 - Field
 - Line
 - Method
 - Thread
 - AWT

En cada caso nos permite establecer condiciones y acciones según se necesite. En este caso crearemos un breakpoint de tipo Line: especificamos el archivo y la línea en la que queremos que se detenga el programa.



Los diferentes tipos de breakpoints en Java NetBeans se utilizan para detener la ejecución del programa en diferentes situaciones:

- 1.Class:** Este tipo de breakpoint se detiene en la primera línea de código de una clase en particular cuando se carga en memoria. Es útil para encontrar problemas que pueden estar relacionados con la inicialización de la clase.
- 2.Exception:** Este tipo de breakpoint se detiene cuando se lanza una excepción. Es útil para encontrar errores de excepción que pueden estar ocurriendo en el código.
- 3.Field:** Este tipo de breakpoint se detiene cuando se accede a un campo (variable) específico en una clase. Es útil para encontrar problemas que pueden estar relacionados con el valor de una variable.
- 4.Line:** Este tipo de breakpoint se detiene en una línea de código específica. Es útil para encontrar problemas que pueden estar ocurriendo en una parte específica del código.
- 5.Method:** Este tipo de breakpoint se detiene en la primera línea de un método específico. Es útil para encontrar problemas que pueden estar relacionados con la ejecución de un método en particular.
- 6.Thread:** Este tipo de breakpoint se detiene cuando se accede a un hilo específico. Es útil para encontrar problemas que pueden estar relacionados con la concurrencia y el paralelismo en el código.
- 7.AWT:** Este tipo de breakpoint se detiene cuando ocurre un evento AWT (Abstract Window Toolkit). Es útil para encontrar problemas que pueden estar relacionados con la interfaz de usuario y la interacción del usuario con el programa.

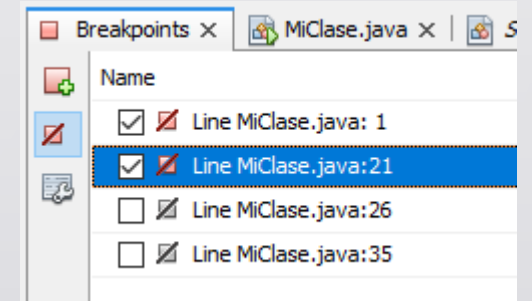
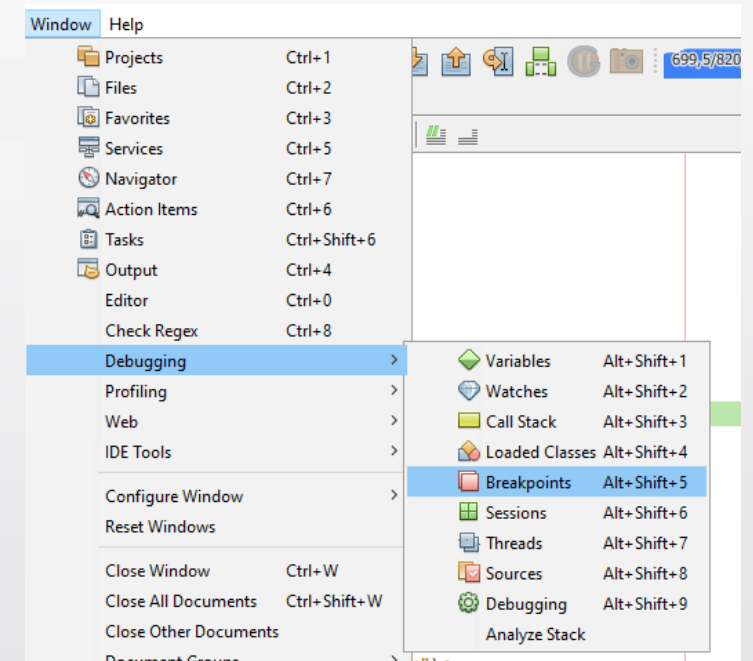
6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Otras opciones

- Para eliminar un breakpoint se pulsa sobre el cuadrado rojo.
- Para eliminarlos todos: botón derecho en la ventana de breakpoints -> Delete All
- Para desactivar un breakpoint, botón derecho sobre la marca roja -> breakpoint -> desmarcar Enable.
- Para desactivarlos todos: botón derecho en la ventana de breakpoints -> Disable All
- Los breakpoints desactivados aparecen en gris.



6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

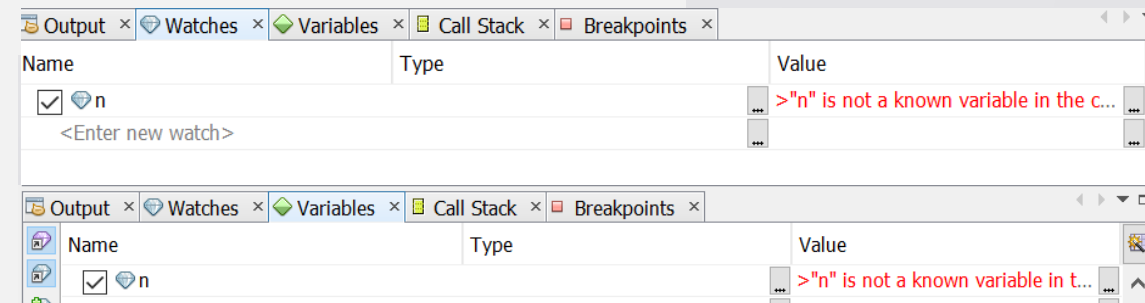
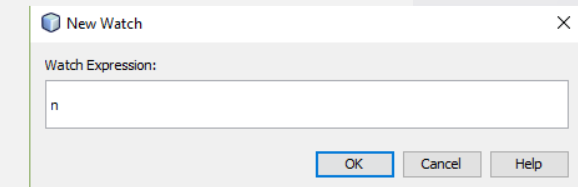
- **New Watch (Ctrl+mayús+F7).** Para obtener el valor de una variable durante la depuración de un programa se sitúa el ratón sobre la variable y se mostrará, el depurador mostrará el tipo y el valor de la variable. Es otra forma de monitorizar el valor de una variable. Las variables a las que se les ha asignado un watch aparecen en la ventana Watches y también aparecen en primer lugar en la ventana de variables.

Para agregar un watch a una variable o una expresión, selecciona la variable o expresión a supervisar y a continuación pulsar. Debug-> New Watch.

O también: botón derecho sobre la variable-> New Watch

En el ejemplo creamos un watch para la variable n.

- Una variable Watch queda definida en Netbeans hasta que no la borremos. Situados en la variable Watch y botón derecho del ratón nos muestra un menú contextual de opciones donde encontramos la opción de eliminar esa watch.



6. Pruebas y depuración de programas ...












6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

En el proceso de depuración se usan distintas ventanas situadas bajo la ventana de código. Algunas aparecen automáticamente.

Para mostrarlas pulsar Windows-> Debugging y seleccionar la que queramos.

Las más importantes son: Breakpoints, Variables, Watches y Call Stack.

	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
	Analyze Stack	

6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

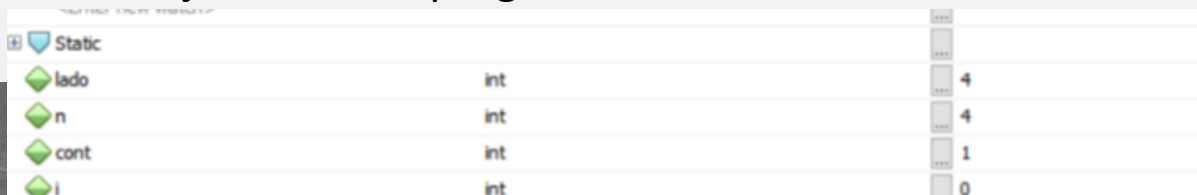
Las ventanas de debugging

Variables. Muestra las variables en uso durante la ejecución del programa. En esta ventana se muestran las variables, su tipo y su valor actual.

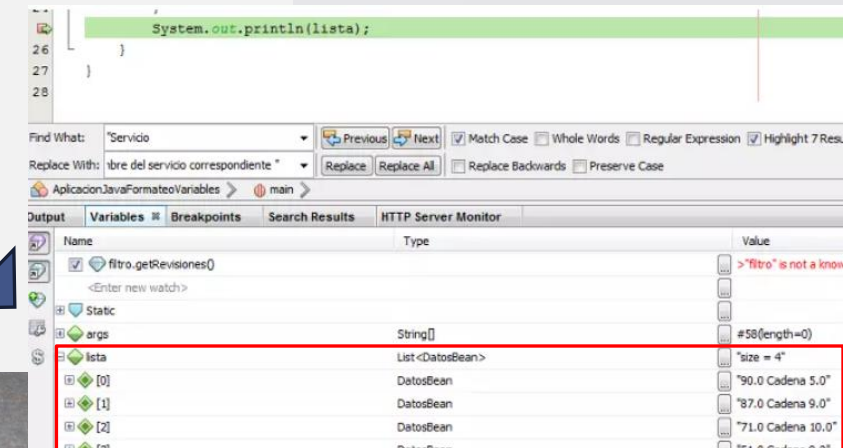
En Netbeans tenemos la posibilidad de formatear variables para el debugger. En el siguiente link explica una posibilidad <https://www.genbeta.com/desarrollo/netbeans-formatear-variables-para-el-debugger>: desde Tools > Options > Java (pestaña) > Java Debugger (pestaña) y dentro de ella en "Variable formatters" > botón "Add..." permite para un determinado método obtener los valores que resultan de su aplicación ("Value formatted as a result of code snippet"). Sirve, por ejemplo, para mostrar directamente el despliegue de valores de variables de objetos y arrays en el debugger.

El debugger permite cambiar el valor de una variable local en esta ventana y continuar la ejecución del programa usando el nuevo valor de la variable.

	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
	Analyze Stack	



Static			
	lado	int	4
	n	int	4
	cont	int	1
	i	int	0



Name	Type	Value
filtro.getRevisiones()		> "filtro" is not a known variable
<Enter new watch>		
	Static	
args	String[]	#58(length=0)
lista	List<DatosBean>	"size = 4"
lista [0]	DatosBean	"90.0 Cadena 5.0"
lista [1]	DatosBean	"87.0 Cadena 9.0"
lista [2]	DatosBean	"71.0 Cadena 10.0"
lista [3]	DatosBean	"51.0 Cadena 0.0"







6. Pruebas y depuración de programas ...












6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Watches.** Muestra las “watches” configuradas.

: Watches		⌵	⌵	: Variables	: Breakpoints
Name	Type			Value	
 i	int			1	
 letter	String			""	
 letter_counter	int			0	

	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
Analyze Stack		

6. Pruebas y depuración de programas ...



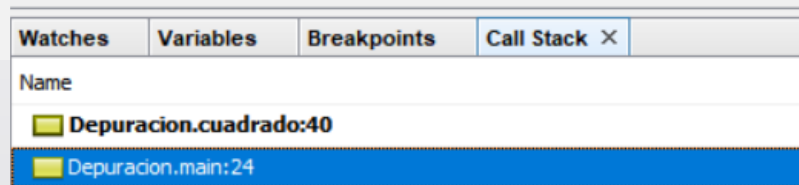
Las ventanas de debugging

) ...

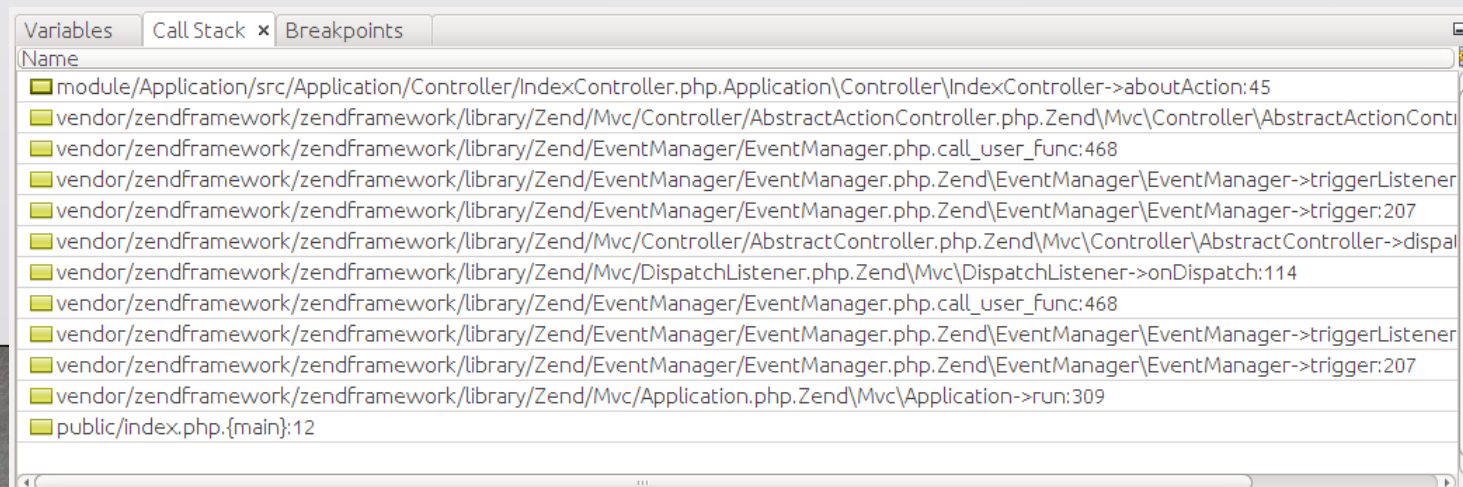
- **Call Stack.** Muestra información para identificar dónde se encuentra el punto de ejecución actual en el código y para comprender cómo llegó allí. La pila de llamadas, «call stack», muestra la lista de funciones anidadas que están siendo ejecutadas. Cada línea de la pila de llamadas (también llamada «stack frame») contiene el nombre completo de la clase, el nombre del método dentro de la clase y el número de línea. Moviéndonos hacia abajo de la pila, podemos entender mejor el actual estado de ejecución del programa.

	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
Analyze Stack		

Ejemplo 1



Ejemplo 2



6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Loaded Classes.** Muestra todas las clases utilizadas durante la ejecución del programa

Class Name	Instances [%] ▼	Instances
byte[]		4.876 (23,7 %)
java.lang.String		4.789 (23,3 %)
java.util.HashMap\$Node		2.612 (12,7 %)
java.util.concurrent.ConcurrentHashMap\$Node		1.125 (5,5 %)
java.lang.Object[]		1.063 (5,2 %)
java.lang.Class		892 (4,3 %)
java.lang.module.ModuleDescriptor\$Exports		362 (1,8 %)
java.util.HashMap		274 (1,3 %)
java.util.HashMap\$Node[]		271 (1,3 %)
java.lang.Integer		262 (1,3 %)
java.util.ImmutableCollections\$Set12		254 (1,2 %)
char[]		221 (1,1 %)
java.util.HashSet		201 (1 %)
java.util.ImmutableCollections\$SetN		142 (0,7 %)
java.lang.String[]		142 (0,7 %)
java.lang.invoke.MethodType		137 (0,7 %)
java.lang.invoke.MethodType\$ConcurrentWeakInternSet\$WeakEntry		137 (0,7 %)
java.lang.module.ModuleDescriptor\$Requires		133 (0,6 %)
java.lang.invoke.LambdaForm\$Name		108 (0,5 %)
java.lang.Class[]		106 (0,5 %)
char[][]		103 (0,5 %)
java.lang.invoke.MemberName		103 (0,5 %)
java.lang.Object		100 (0,5 %)
java.lang.invoke.ResolvedMethodName		93 (0,5 %)
java.lang.ref.SoftReference[]		80 (0,4 %)
java.lang.Module		69 (0,3 %)
java.lang.invoke.LambdaForm\$Kind		68 (0,3 %)
jdk.internal.module.ServicesCatalog\$ServiceProvider		64 (0,3 %)
java.lang.module.ResolvedModule		63 (0,3 %)

	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
Analyze Stack		

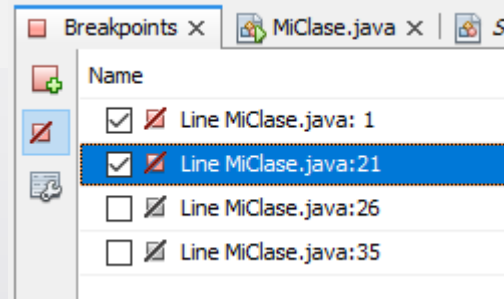
6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Breakpoints.** Muestra los breakpoints establecidos por el usuario.



	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
	Analyze Stack	


6. Pruebas y depuración de programas ...












6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Sessions.** Muestra las sesiones de depuración activas y su estado.

Watches	Variables	Breakpoints	Call Stack	Loaded Classes	Sessions X
Name		State		Language	
 depuracion.Depuracion		Running		Java	

	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
Analyze Stack		

6. Pruebas y depuración de programas ...












6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Threads.** Muestra los hilos utilizados en la ejecución del programa. Los que aparecen en **negrita** son los que se están utilizando actualmente.

Threads	
Name	State
system	
main	
main	Running
InnocuousThreadGroup	
Common-Cleaner	Waiting
Reference Handler	Running
Finalizer	Waiting
Signal Dispatcher	Running
Attach Listener	Running
Notification Thread	Running

	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
Analyze Stack		

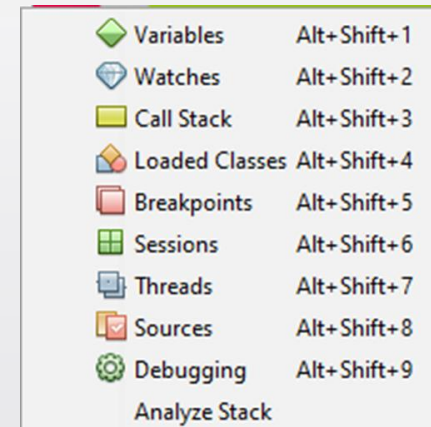
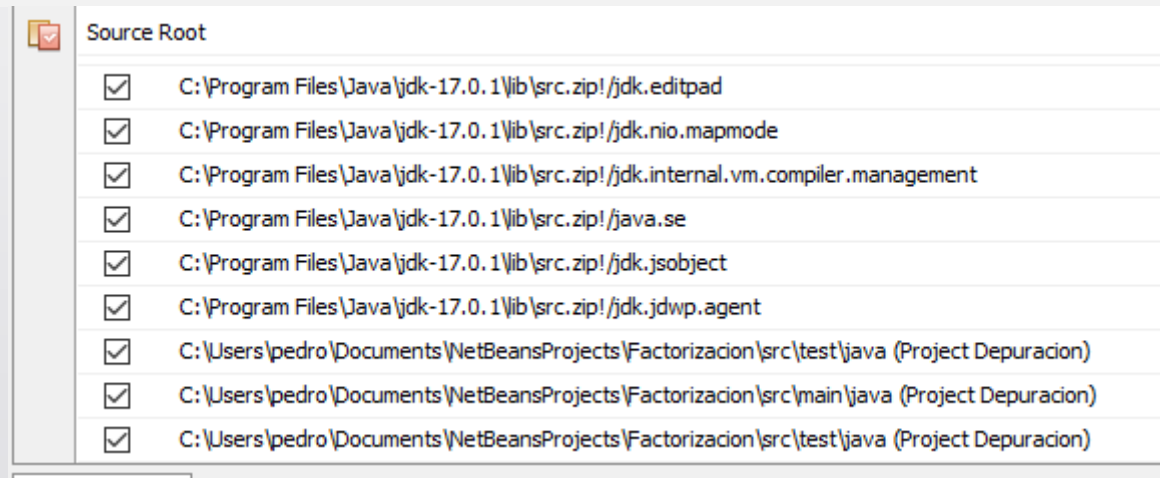
6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Sources.** Muestra los directorios de los archivos que se utilizan en el programa. En este caso el kit de desarrollo de java y los archivos del programa.



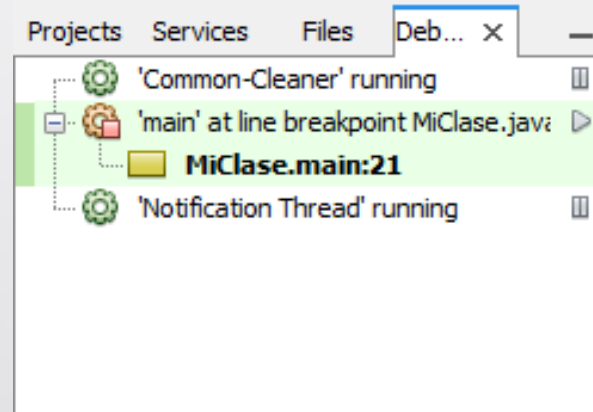
6. Pruebas y depuración de programas ...












6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Debugging.** Muestra el estado de la depuración y los pasos que se van realizando.



	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
Analyze Stack		

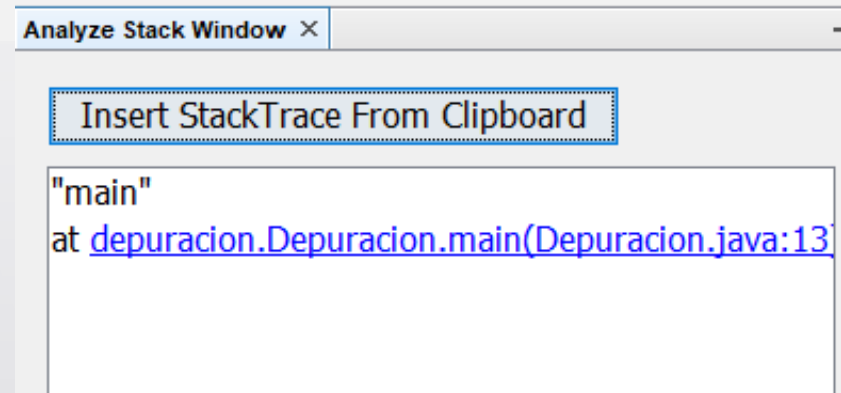
6. Pruebas y depuración de programas ...



6.2 Depuración de código en Netbeans (debugger) ...

Las ventanas de debugging

- **Analyze Stack.** Permite realizar un seguimiento de cada Stack insertándolo, para así acceder rápidamente a la línea de código en la que se encuentra.



	Variables	Alt+Shift+1
	Watches	Alt+Shift+2
	Call Stack	Alt+Shift+3
	Loaded Classes	Alt+Shift+4
	Breakpoints	Alt+Shift+5
	Sessions	Alt+Shift+6
	Threads	Alt+Shift+7
	Sources	Alt+Shift+8
	Debugging	Alt+Shift+9
	Analyze Stack	

6. Pruebas y depuración de programas ...



6.3 Ejemplo depuración de código en Netbeans ...

Como vemos, la depuración de código en Netbeans, como en otros IDE, permite colocar puntos de interrupción (breakpoints) en tu código fuente, agregar observadores de campo, recorrer tu código, ejecutar métodos, tomar instantáneas y monitorear la ejecución a medida que ocurre. También se puede adjuntar el depurador a un proceso que ya se está ejecutando.

Realizaremos distintos pasos:

6. Pruebas y depuración de programas ...



6.3 Ejemplo depuración de código en Netbeans

1. En **primer lugar** vamos a añadir **breakpoints**. Un breakpoint es un punto de parada de ejecución en el código. Es decir, el programa se ejecutará hasta la línea que marcas como breakpoint. Para añadir o quitar un breakpoint debes hacer click en el número de la línea y ésta te saldrá en rojo si está activado, en nuestro caso vamos a poner dos puntos de parada, en la línea de creación del objeto teclado y en la petición del primer número. También dentro del menú debugger tendrá la opción “toggle line breakpoint”

```
public class MiClase {  
  
    public static void main(String[] args) {  
        // TODO code application logic here  
        //Creamos el objeto  
        try {  
            Scanner teclado = new Scanner(System.in);  
            //declaramos las variables  
            int numero1;  
            int numero2;  
            int resultado;  
            System.out.println("Introduzca el primer número");  
            numero1 = teclado.nextInt();  
            System.out.println("Introduzca el segundo número");  
            numero2 = teclado.nextInt();  
            if (numero1 != 0 && numero2 != 0) {  
                resultado = suma_enteros(numero1, numero2);  
                System.out.println("El resultado es " + resultado);  
            }  
        } catch (Exception e) {  
            System.out.println("se ha producido un error");  
        }  
    }  
  
    private static int suma_enteros(int a, int b) {  
        int resultado = a + b;  
        return resultado;  
    }  
}
```

6. Pruebas y depuración de programas ...

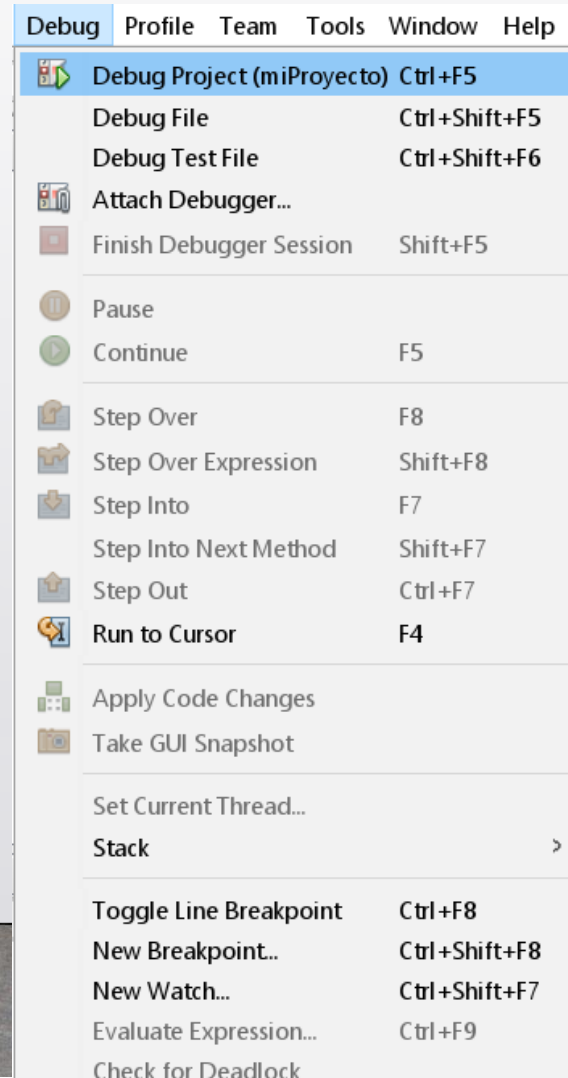
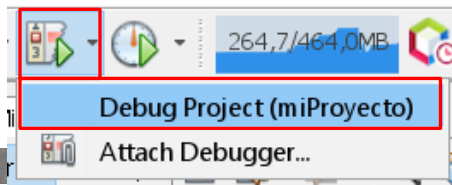


6.3 Ejemplo depuración de código en Netbeans ...

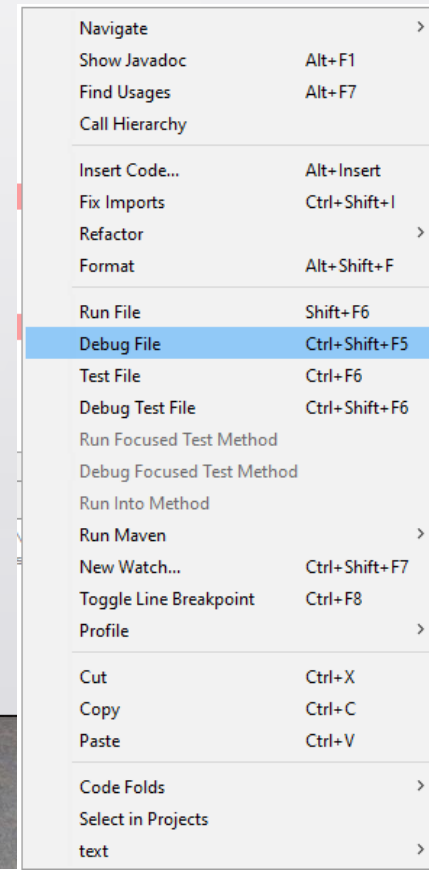
2. Ahora, para habilitar la ejecución en modo “debug” vamos a hacer click en el menú “Debug/Debug Project”, donde encontramos todas las opciones disponibles.

Las opciones que se encuentran deshabilitadas lo están porque no pueden utilizarse hasta que realmente no ejecutemos el modo “debug”.

También podemos iniciar el proceso de depuración en el icono



También se puede hacer click con botón derecho sobre el fichero que queremos depurar.

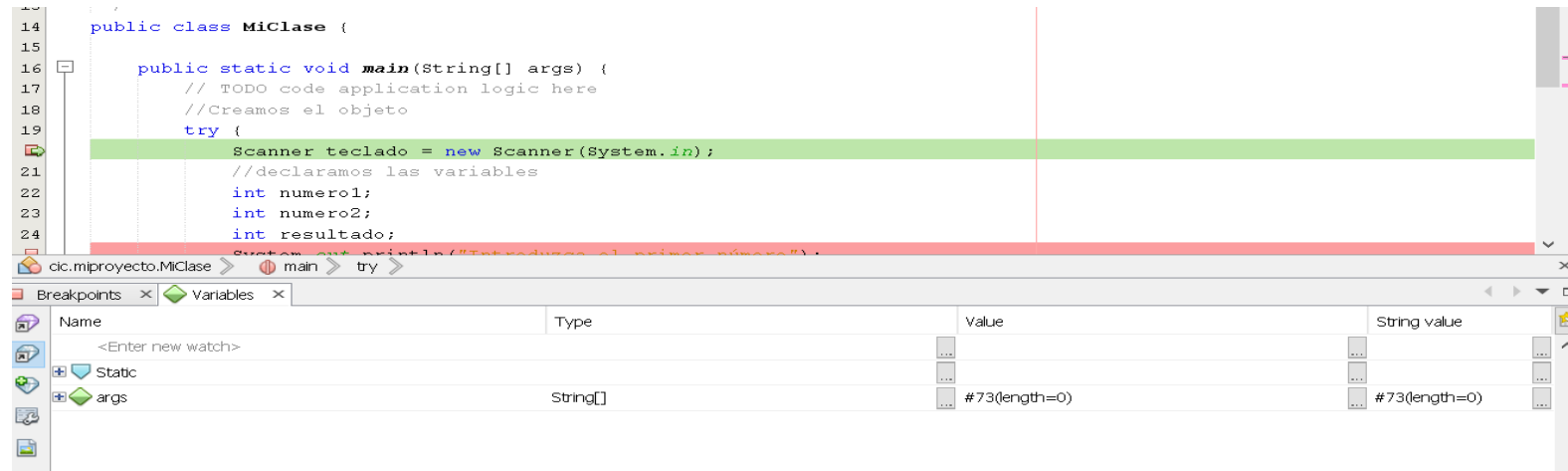


6. Pruebas y depuración de programas ...



6.3 Ejemplo depuración de código en Netbeans ...

Al hacer clic en la opción “Debug Project” el código se ejecutará hasta la primera línea de breakpoint. Verás que la línea se ha puesto en verde (pero está aún sin ejecutar) y aparecen abajo varias pestañas (variables, Breakpoints).



En la pestaña Variables puedes ver los valores de las variables declaradas hasta el punto de ruptura. En la pestaña Breakpoints verás un resumen de los breakpoints que están en el código.



6. Pruebas y depuración de programas ...

6.3 Ejemplo depuración de código en Netbeans ...

3. Nos aparecen una serie de iconos



que también se encuentran en el menú debug. Con la opción del menú “Debug/**Step Over**” o **F8**, podemos ejecutar una sola línea de código. En ese momento aparece en verde la siguiente línea y se ha ejecutado la línea en la que teníamos el breakpoint. Se actualiza también la pestaña de variables con los nuevos valores. Podemos ir ejecutando todo el código en busca de la instrucción que establece un valor inesperado en una variable o que provoca un fallo en el programa.

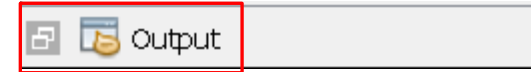
```
public class MiClase {  
    public static void main(String[] args) {  
        // TODO code application logic here  
        //Creamos el objeto  
        try {  
            Scanner teclado = new Scanner(System.in);  
            //declaramos las variables  
            int numero1;  
            int numero2;  
            int resultado;  
            System.out.println("Introduzca el primer número");  
            numero1 = teclado.nextInt();  
            System.out.println("Introduzca el segundo número");  
            numero2 = teclado.nextInt();  
            if (numero1 != 0 && numero2 != 0) {
```

Name	Type	Value	String value
<Enter new watch>			
Static			
args	String[]	#73(length=0)	#73(length=0)
teclado	Scanner	#385	java.util.Scanner[d...]

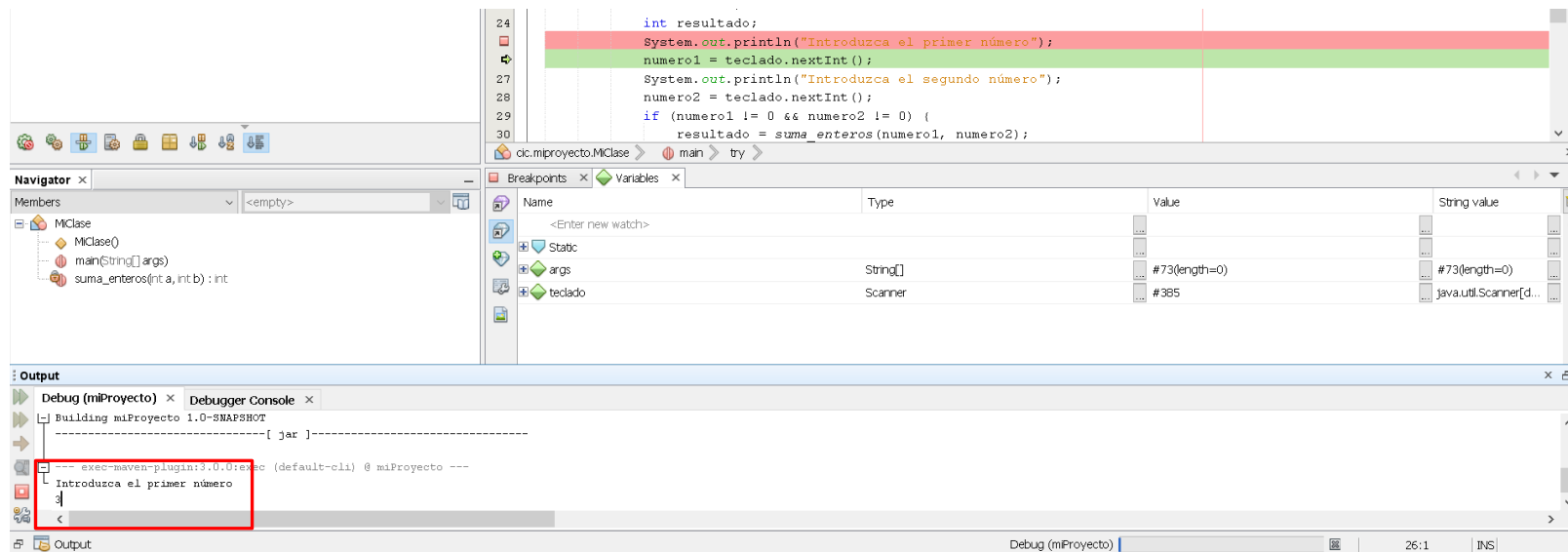
6. Pruebas y depuración de programas ...

6.3 Ejemplo depuración de código en Netbeans ...

Ahora la opción de output se encuentra a la izquierda abajo



Tenemos que abrir la ventana para ver el mensaje y volvemos a pulsar F8 para introducir el primer valor.



Introducimos el valor y damos ENTER y entonces nos aparece en la ventana de variables que `numero1` tiene el valor introducido por pantalla. Podemos comprobar que recoge bien el valor de la variable, también nos permite cambiarlo.

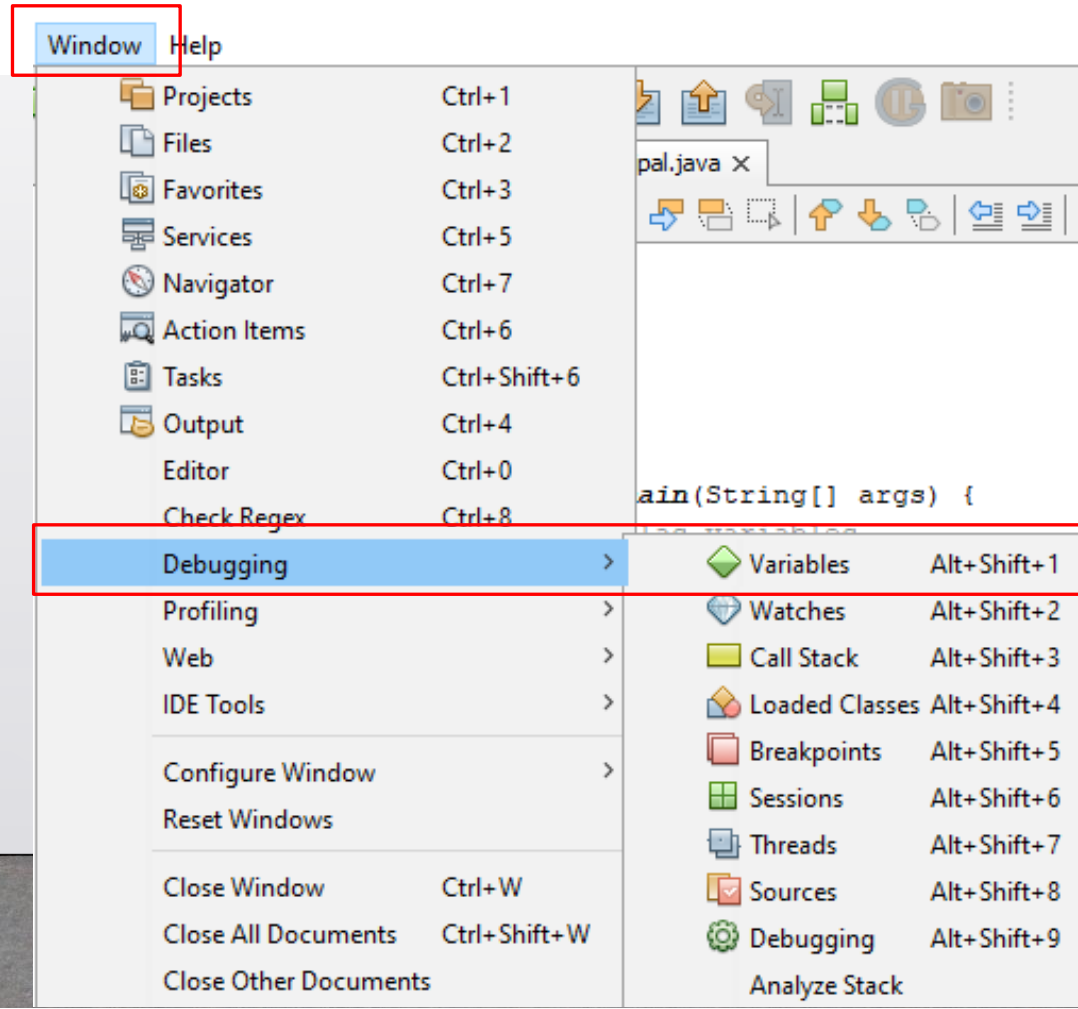


6. Pruebas y depuración de programas ...



6.3 Ejemplo depuración de código en Netbeans ...

Si no aparecen las variables se pueden mostrar desde la pestaña Window>Debugging>Variables



6. Pruebas y depuración de programas ...



6.3 Ejemplo depuración de código en Netbeans ...

4. Vamos a ir ejecutando paso a paso hasta situarnos en el método `suma_enteros`, que en vez de dar F8, en este caso daremos **F7** o “**Step Into**” y vemos como entra en el método `suma_enteros`.

The screenshot shows the NetBeans IDE interface. The top pane displays a Java code editor with the following code:

```
37  
38 private static int suma_enteros(int a, int b) {  
39     int resultado = a + b;  
40     return resultado;  
41 }  
42  
43  
44
```

The line `int resultado = a + b;` is highlighted in green. Below the code editor, the breadcrumb navigation shows `cic.miproyecto.MiClase > suma_enteros`. The bottom pane is the **Variables** window, which contains a table of current variables:

Name	Type	Value	String value
<Enter new watch>			
Static			
a	int	3	3
b	int	5	5

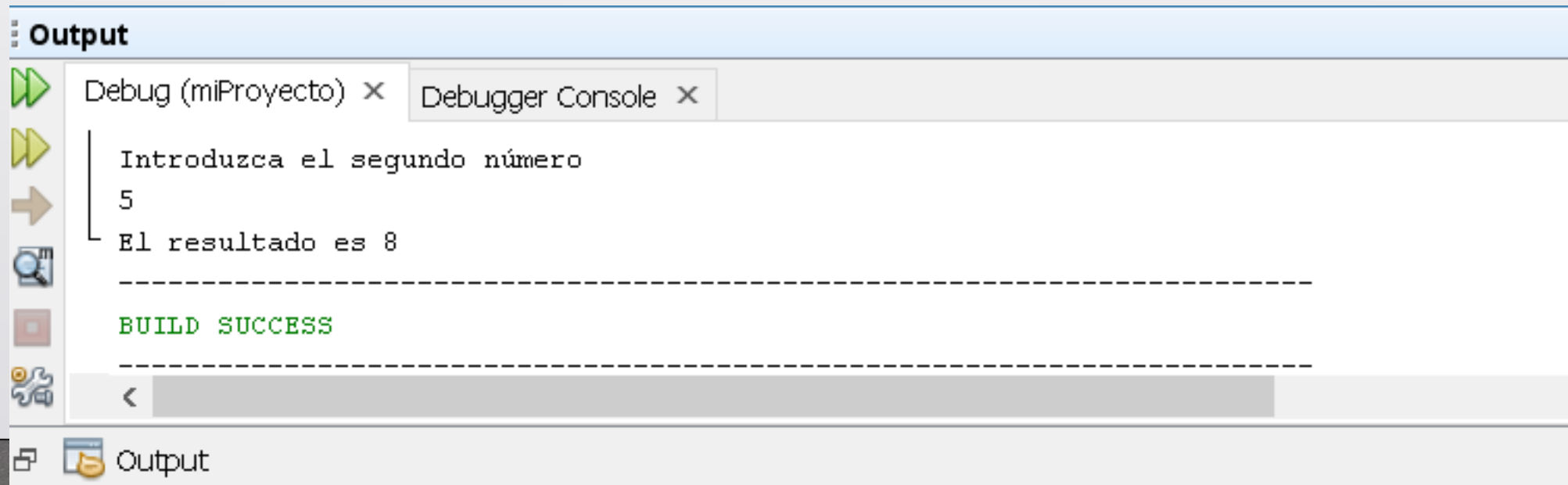
6. Pruebas y depuración de programas ...



6.3 Ejemplo depuración de código en Netbeans ...

5. Otras opciones que tenemos en el menú Debug que son interesantes son “**Run to cursor**”, que nos permite ejecutar el código hasta la línea en la que tenemos el cursor en ese momento, o “**Continue**” o **F5**, que ejecuta el código hasta el siguiente breakpoint o hasta el final del programa, si no hemos puesto más.

Si pulsamos F5 en nuestro caso terminará la ejecución





TAREA 9. Depuración de código en Netbeans

Realiza la tarea correspondiente que encontrarás en la plataforma educativa.

Realiza la entrega de esta tarea de acuerdo con las instrucciones que se indiquen.