

UT05. DISEÑO Y REALIZACIÓN DE PRUEBAS

Entornos de Desarrollo

1 DAW – C.I.F.P. Carlos III - Cartagena

3. Procedimientos y casos de prueba

Procedimientos y casos de prueba

- Según el IEEE

*Un **caso de prueba** es un*

- *conjunto de entradas,*
- *condiciones de ejecución y*
- *resultados esperados,*

desarrollados para un objetivo particular, como, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito, incluyendo toda la documentación asociada.

- Complejidad de las aplicaciones informáticas ➔ imposibilidad de probar todas la combinaciones ➔ asegurar que los casos de prueba obtienen un nivel aceptable de probabilidad de detección de errores

4. Herramientas de depuración

Herramientas de depuración

- Herramientas de depuración de un IDE
 - Inclusión de **puntos de ruptura**
 - **Ejecución paso a paso** de cada instrucción
 - **Ejecución por procedimiento**
 - **Inspección de variables**
 - etc.

Herramientas de depuración

- Tipos de errores en el proceso de desarrollo
 - **Errores de compilación:** errores de sintaxis.
 - P.e. no poner punto y coma al final de una sentencia
 - El entorno avisará del error
 - No se puede continuar hasta corregir el error
 - **Bugs o errores de ejecución o lógicos**
 - No evitan que el programa se compile con éxito
 - Genera resultados inesperados, termina de forma inesperada o no termina nunca
- El proceso de depuración ➔ comienza con la ejecución de un caso de prueba.
- Se **evalúan los resultados** de la ejecución
- Se comprueba si hay **falta de correspondencia** entre los resultados esperados y los obtenidos.

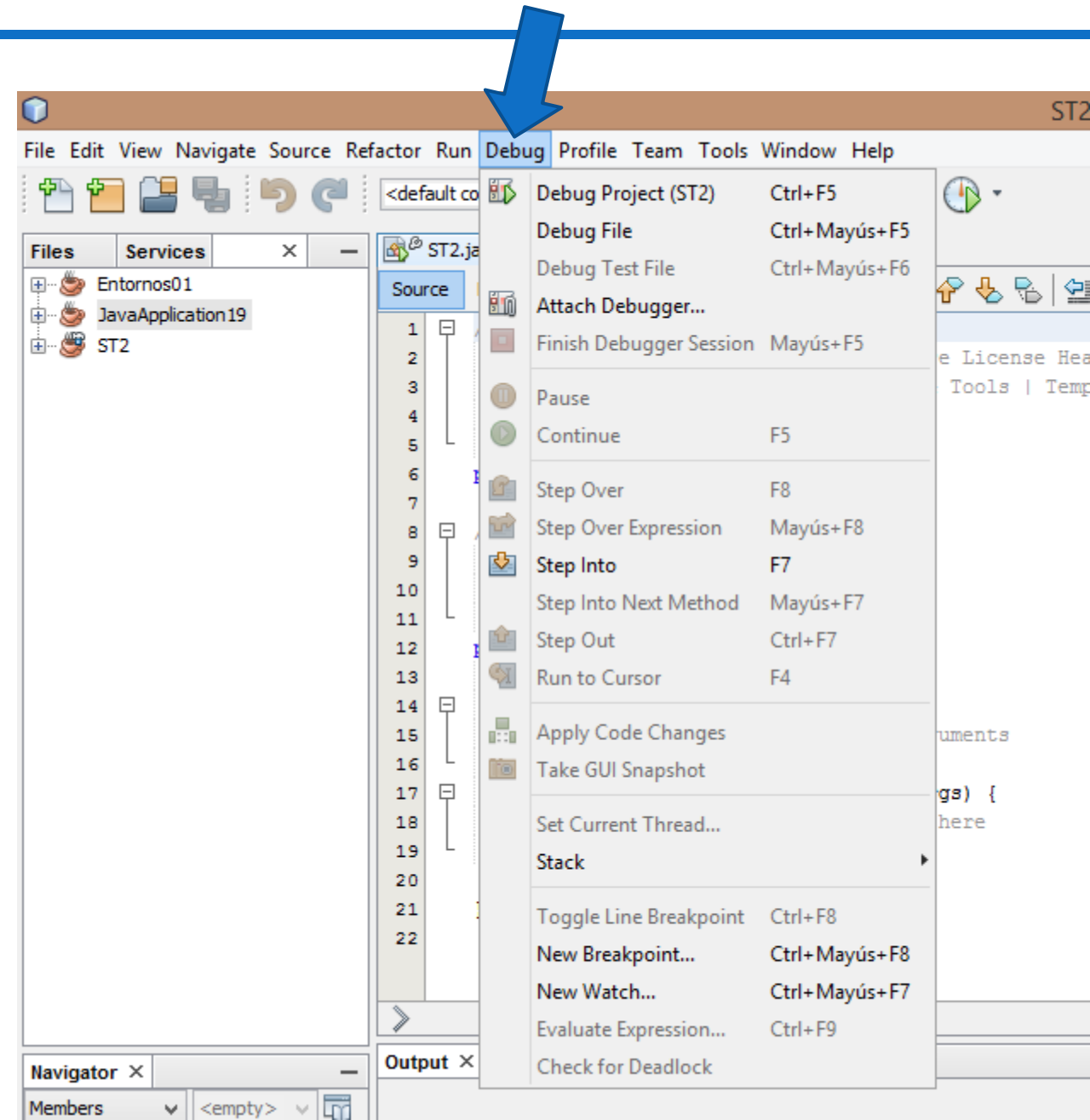
Herramientas de depuración

- Resultados
 - Se encuentra el error → se corrige y se elimina
 - No se encuentra → estudiar las posibles causas, diseñar casos de prueba para verificar la causa, repetir las pruebas para identificar los errores y corregirlos

Depurador

- **Depurador**

- Permite supervisar la ejecución de los programas, para localizar y eliminar los **bugs**.
- Condición imprescindible: debe compilarse con éxito
 - Puntos de ruptura
 - Tipos de ejecución
 - Examinadores de variables




```
public class LlenarNumeros {
```

```
    /**...3 lines */
```

```
    public static void main(String[] args) {  
        int n=5;  
        int[] tabla=new int[n];  
  
        tabla=llenar(n);  
        int suma=sumar(tabla);  
        System.out.println("La suma es: "+suma);  
    }
```

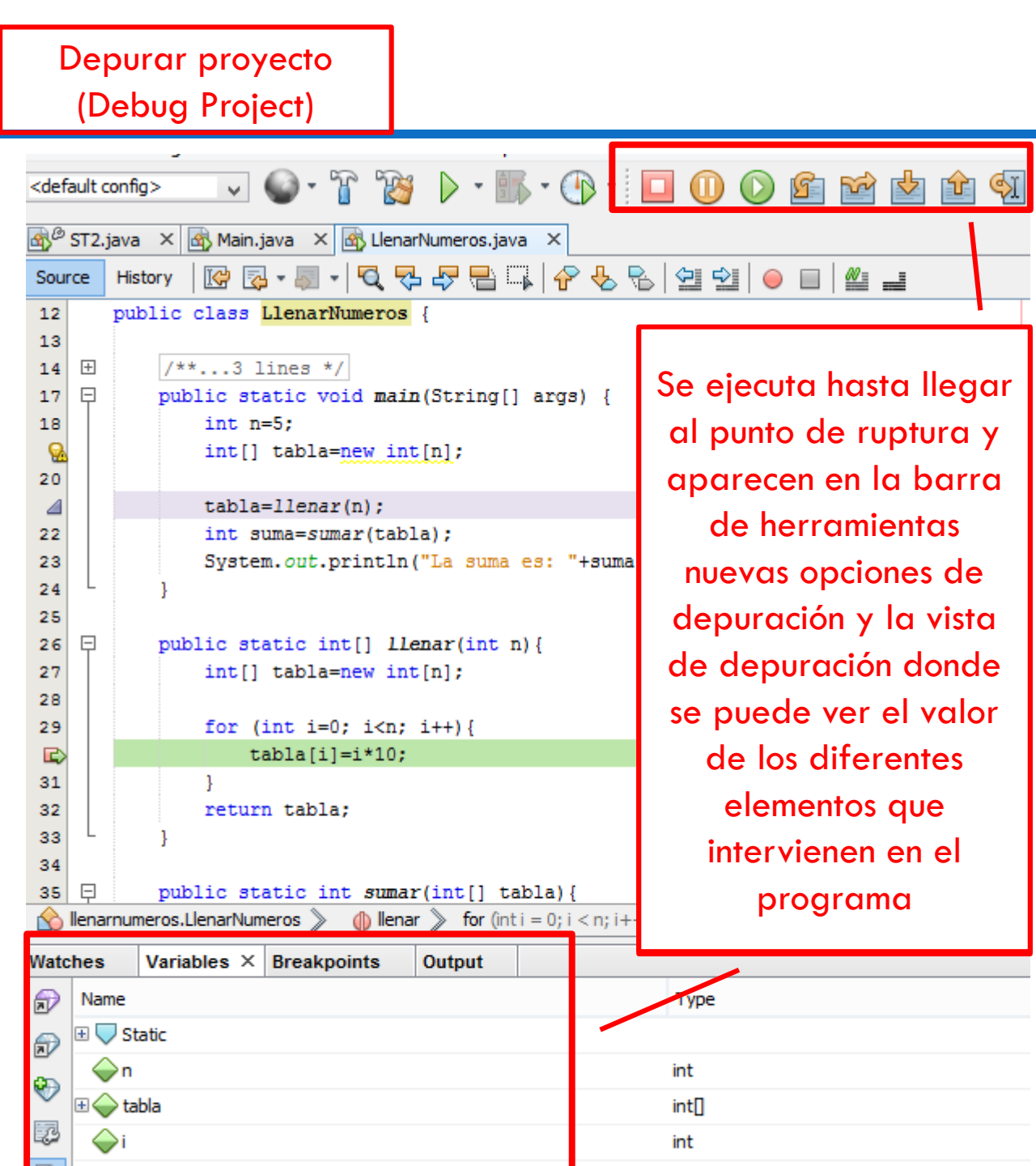
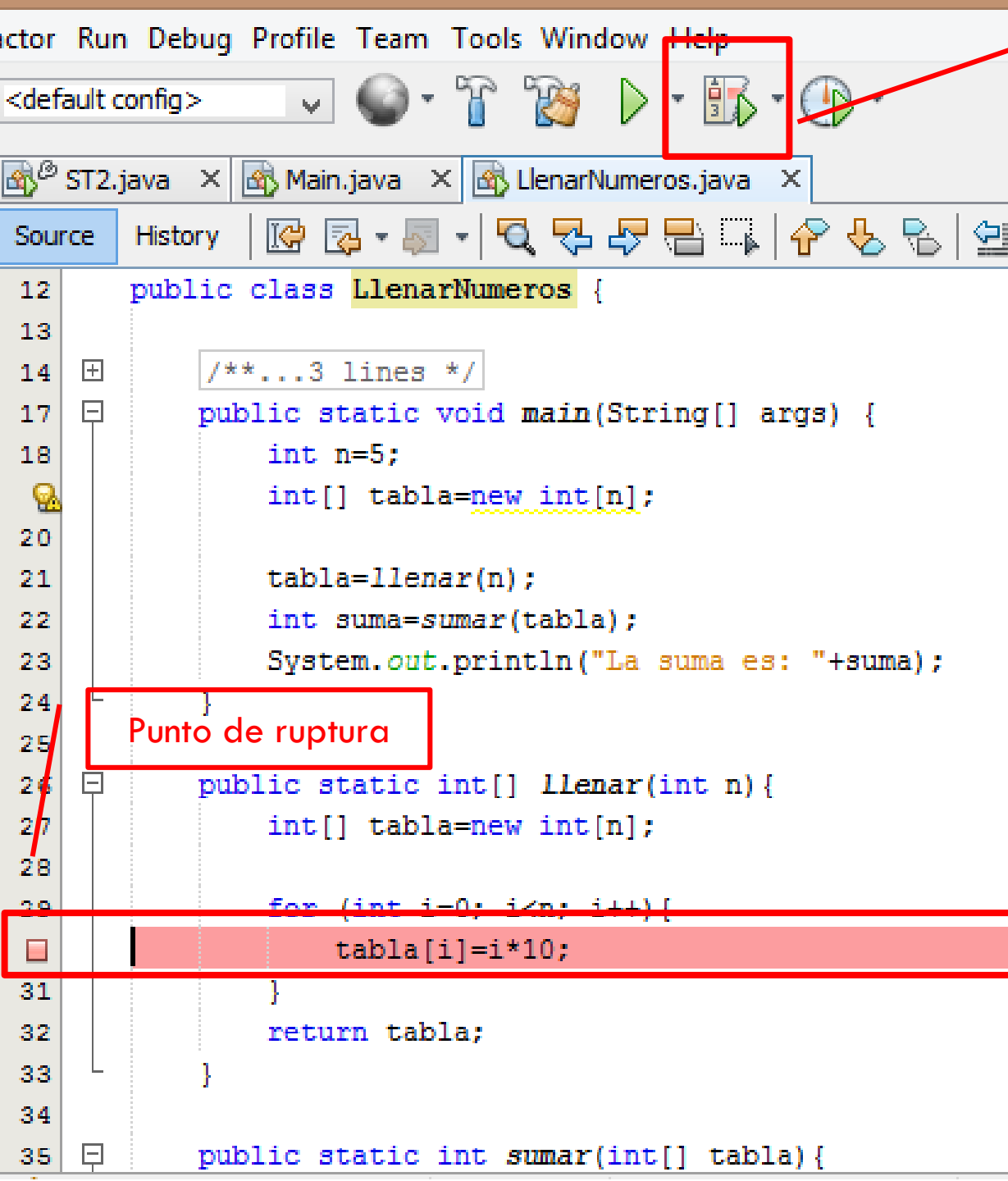
```
    public static int[] llenar(int n){  
        int[] tabla=new int[n];  
  
        for (int i=0; i<n; i++){  
            tabla[i]=i*10;  
        }  
        return tabla;  
    }
```

```
    public static int sumar(int[] tabla){  
        int suma=0;  
        int n=tabla.length;  
  
        for(int i=0; i<n; i++){  
            suma=suma+tabla[i];  
        }  
        return suma;  
    }
```

Copia el siguiente código en
Netbeans.
¿Qué hace?

Depurador. Puntos de ruptura

- **Puntos de ruptura o breakpoint y seguimiento**
 - Seleccionar la línea de código donde queremos que el programa se pare
 - Inspeccionar variables, o realizar una **ejecución paso a paso**, para verificar la corrección del código
 - Son marcadores que pueden establecerse en cualquier línea de código ejecutable.
 - Se pueden insertar varios puntos de ruptura
 - Se pueden eliminar fácilmente
 - Se establecen haciendo clic en el margen izquierdo o botón derecho > Breakpoint
 - Debug > New breakpoints



IDE interface showing a Java program being debugged. The main window displays the source code of `LlenarNumeros.java`. The program defines a `llenar` method that fills an array and a `sumar` method that calculates the sum of the array elements.

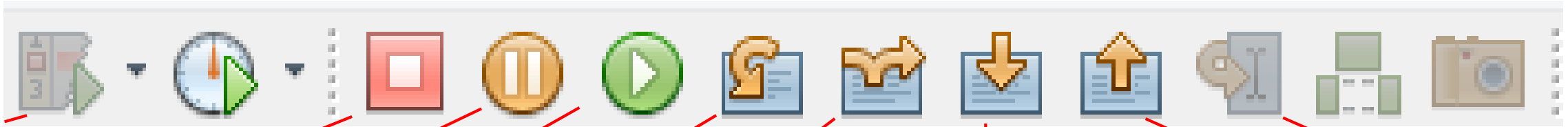
The execution is paused at line 30 of `LlenarNumeros.java`, which is highlighted in green. A red box highlights this line with the text: "Línea en la que se ha detenido la ejecución y línea que ha realizado la llamada".

The Variables window (Inspector de variables) is open, showing the current state of the program's variables. A red box highlights this window with the text: "Inspector de variables".

The Variables window displays the following variables and their values:

Name	Type	Value
Static		...
n	int	5
tabla	int[]	#73(length=5)
[0]	int	0
[1]	int	0
[2]	int	0
[3]	int	0
[4]	int	0
i	int	0

Depurador. Barra de herramientas



**Debug
Proyect**

**Finish
Debugger
Session**

Pause

Continue

Salta los
métodos

Step Over:

Ejecuta una línea de código. Si la instrucción es una llamada a un método, ejecuta el método sin entrar dentro del código del método..

Step Over

Expression: Si una expresión tiene múltiples llamadas a los métodos, se puede utilizar **Step Over Expression** y ver el valor de cada llamada de método en la expresión en la ventana Variables locales

Step Into: Si la instrucción es una llamada a un método, salta al método y continúa la ejecución por la primera línea del método.

Step Out:

Ejecuta una línea del fuente. 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

Run to Cursor:

Se ejecuta el programa hasta la instrucción donde se encuentra el cursor.

Depurador. Puntos de ruptura condicionales

- **BP Condicionales:** el programa para sólo si se cumple la condición especificada.
- Nuevo BP
- Menú contextual en el BP >> Breakpoint>>Propierties
- Condition → $i==3$
- Debug Project → ejecuta hasta que se cumple la condición

The screenshot illustrates the steps to set a conditional breakpoint in a Java IDE. A context menu is shown over a line of code in a file named `LlenarNumeros.java`. The 'Properties' option is selected. The 'Breakpoint Properties' dialog is open, showing the 'Conditions' tab with the condition `i==3` entered. The 'Watches' panel at the bottom shows the variable `i` with a value of 3.

Breakpoint Properties Dialog:

- Settings: File: `oyectos/LlenarNumeros/src/lleannumeros/LlenarNumeros.java`, Line number: 30
- Conditions: ☒ Condition: `i==3`
- Break when hit count is: Equal to
- Actions: Suspend: Breakpoint thread, Enable Group: <none>, Disable Group: <none>, Print Text: Breakpoint hit at line {lineNumber} in class {className} by thre

Watches Panel:

Name	Type	Value
Static		
n	int	5
tabla	int[]	#72(length=5)
i	int	3

Depurador. Examen y modificación de variables

- Uno de los elementos más importantes del proceso de depuración de un programa.
- NetBeans → Ventana de Inspección > modificar el contenido de las variables y continuar con la ejecución.

The screenshot shows the NetBeans IDE with a Java file named `LlenarNumeros.java`. The code is as follows:

```
25  
26 public static int[] llenar(int n){  
27     int[] tabla=new int[n];  
28  
29     for (int i=0; i<n; i++){  
30         tabla[i]=i*10;  
31     }  
32     return tabla;  
33 }  
34
```

A conditional breakpoint is set on line 29 with the condition `i==2`. The `Variables` window is open, showing the state of the program:

Name	Type	Value
Static		
n	int	5
tabla	int[]	#408(length=5)
[0]	int	0
[1]	int	10
[2]	int	0
[3]	int	0
[4]	int	0
i	int	2

A red box highlights the breakpoint condition and the variable `i` in the Variables window, with a red arrow pointing to the right.

BP condicional con `i==2`. Modifico la `i` a 4, continuo con la ejecución y el resultado es el que se muestra a continuación

The screenshot shows the NetBeans IDE with the same Java file. The code is as follows:

```
25  
26 public static int[] llenar(int n){  
27     int[] tabla=new int[n];  
28  
29     for (int i=0; i<n; i++){  
30         tabla[i]=i*10;  
31     }  
32     return tabla;  
33 }  
34
```

The `Output` window is open, showing the results of the program execution:

```
debug:  
La suma es: 30  
BUILD SUCCESSFUL (total time: 9 minutes 20 seconds)
```

A red arrow points from the `i` variable in the Variables window to the `debug:` output.

Depurador. Ejemplo

- **Ejemplo:** Escribir un programa en Java en Netbeans que resuelva la potencia de un número. Depurar y ejecutar observando la evolución de las variables en la **Ventana de Inspección**.

Depurador. Examen y modificación de variables

The image shows an IDE interface with the following components:

- File Explorer:** Shows a project structure with folders like 'Calculadora', 'coche', 'Entornos01', 'JavaApplication19', 'LlenarNumeros', 'Source Packages', 'Test Packages', 'Libraries', and 'Test Libraries'. The 'LlenarNumeros' folder is selected.
- Debugger Menu:** The 'Debug' menu is open, showing options like 'Debug Project (LlenarNumeros) Ctrl+F5', 'Debug File Ctrl+Mayús+F5', 'Debug Test File Ctrl+Mayús+F6', 'Attach Debugger...', 'Finish Debugger Session Mayús+F5', 'Pause', 'Continue F5', 'Step Over F8', 'Step Over Expression Mayús+F8', 'Step Into F7', 'Step Into Next Method Mayús+F7', 'Step Out Ctrl+F7', 'Run to Cursor F4', 'Apply Code Changes', 'Take GUI Snapshot', 'Set Current Thread...', 'Stack', 'Toggle Line Breakpoint Ctrl+F8', 'New Breakpoint... Ctrl+Mayús+F8', 'New Watch... Ctrl+Mayús+F7', 'Evaluate Expression... Ctrl+F9', and 'Check for Deadlock'.
- New Watch Dialog:** A dialog box titled 'New Watch' is open. It has a 'Watch Expression' field containing 'i*n'. The 'OK' button is highlighted.
- Code Editor:** The code editor shows the source code of 'LlenarNumeros.java'. The code is as follows:

```
17 public static void main(String[] args) {  
18     int n=5;  
19     int[] tabla=new int[n];  
20  
21     tabla=llenar(n);  
22     int suma=sumar(tabla);  
23     System.out.println("La suma es: "+suma);  
24 }  
25  
26 public static int[] llenar(int n){  
27     int[] tabla=new int[n];  
28  
29     for (int i=0; i<n; i++){  
30         tabla[i]=i*10;  
31     }  
32     return tabla;  
33 }  
34  
35 public static int sumar(int[] tabla){  
36     int suma=0;  
37     int n=tabla.length;  
38  
39     for(int i=0; i<n; i++){  
40         suma=suma+tabla[i];  
41     }  
42     return suma;  
43 }
```


A breakpoint is set on line 29. The 'Watches' tab is active, showing a table with the following data:

Name	Type	Value
i*n	int	15

Para evaluar expresiones, como por ejemplo $i*n$. La inspección de la expresión durante la ejecución paso a paso se puede realizar en la pestaña Watches.