

2.2)Corregir errores sintácticos (y documentarlos)

Errores sintácticos encontrados:

- Ausencia de cierre de instrucción con ';' :

```
7 public static void main(String[] args) {  
8     Scanner sc = new Scanner(System.in);  
9     String resp;  
10  
11     do {  
12         int num = LeerEntero(sc, "Introduzca un número: " ; //FMTS20251031 - Faltaba el ';' ;  
13     }
```

- Declaración de una variable que ya existe:

```
20 System.out.print("¿Quiere hacer otra pirámide? (s/n) ");  
21 resp = sc.next().trim().toUpperCase(); //FMTS20251031 - Sobraba la declaración de la variable porque ya está declarada al principio del código.
```

- Ausencia de cierre de Do-While:

```
22  
23     } while (resp.equals("S")); //FMTS20251031 - Faltaba cerrar la instrucción con ';' ;  
24     borrarConsola();  
25     System.out.println("¡¡¡PROGRAMA FINALIZADO!!!");  
26 }
```

- Error de sintaxis en los parámetros del for:

```
39  
40 public static void borrarConsola() {  
41     for (int i = 0; i < 5; i++ { //FMTS20251031 - Había una ',' donde debía haber un ';' ;  
42         System.out.println();  
43     }  
44 }  
45 }
```

- Nombre de variable mal escrito:

```
66     }  
67     return res; //FMTS20251031 - El nombre de la variable es 'res'.  
68 }
```

¿Qué es un error sintáctico?

Es un error en la sintaxis, es decir, la estructura que debe tener el código para estar correctamente escrito.

¿Cuándo los visualizamos?

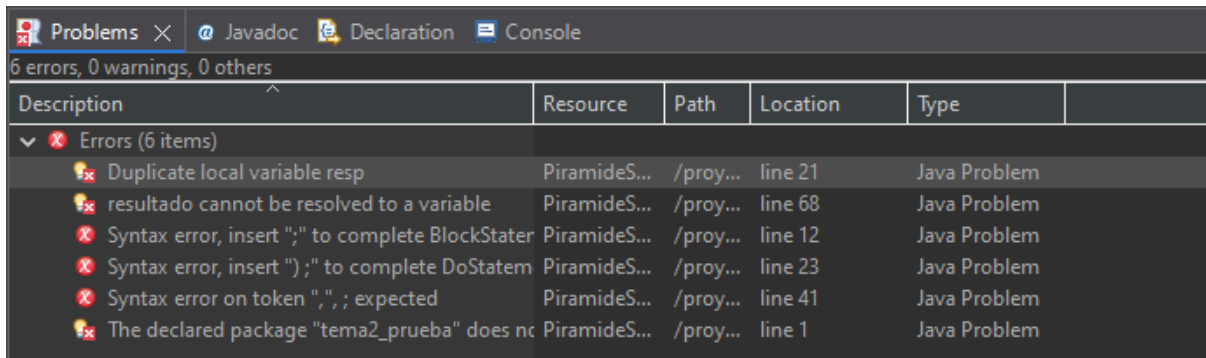
Antes de compilar el código se muestran los errores sintácticos en pantalla.

¿Podemos depurar con errores de sintaxis?

No, primero se deben corregir los errores de sintaxis para que el programa entienda las instrucciones.

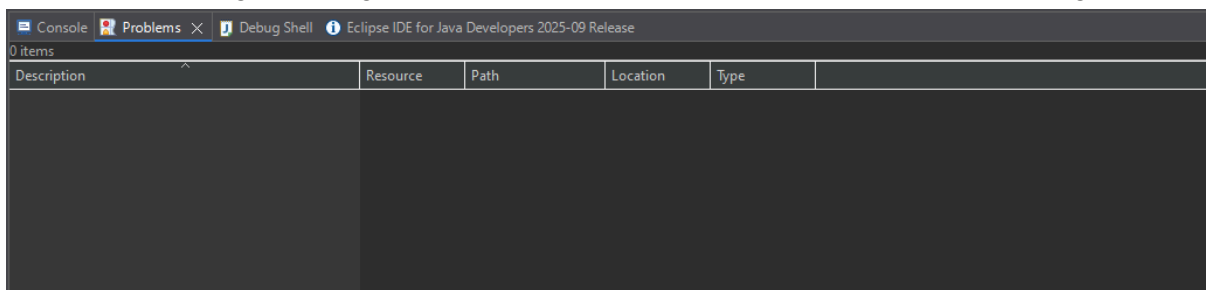
Vista Problems antes de la corrección:

En la vista 'Problems' se muestra una lista con todos los errores de sintaxis detectados por Eclipse.



Vista Problems después de la corrección:

Después de corregir el código podemos ver que la vista 'Problems' no muestra ningún error.

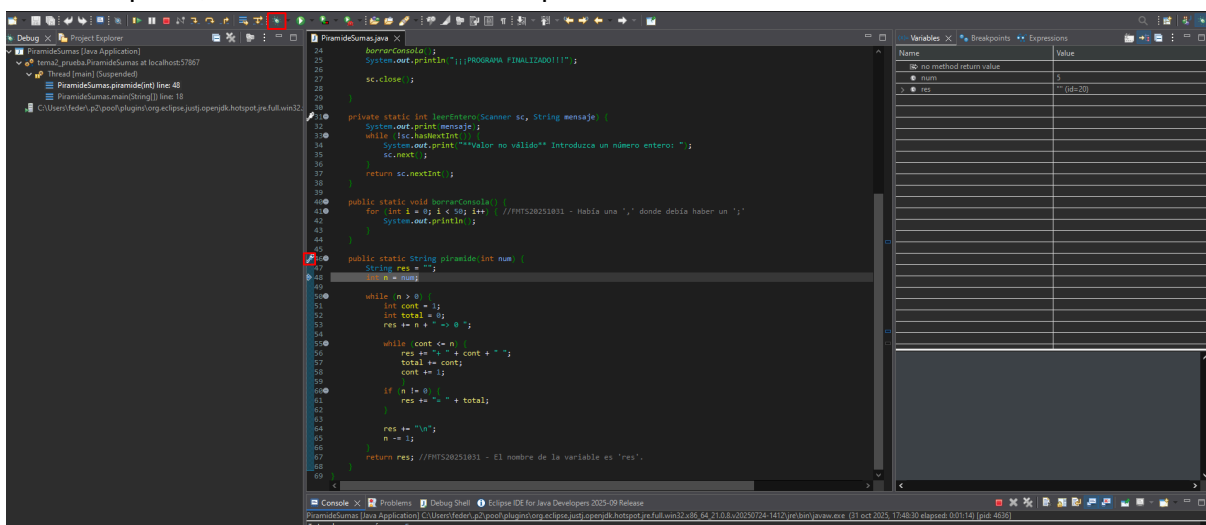


2.3) Corregir errores lógicos usando el Depurador

Mini guía de depuración:

Para depurar un programa debemos poner un breakpoint en la línea del código por la que deseamos empezar, después haremos clic sobre el botón de depurar.

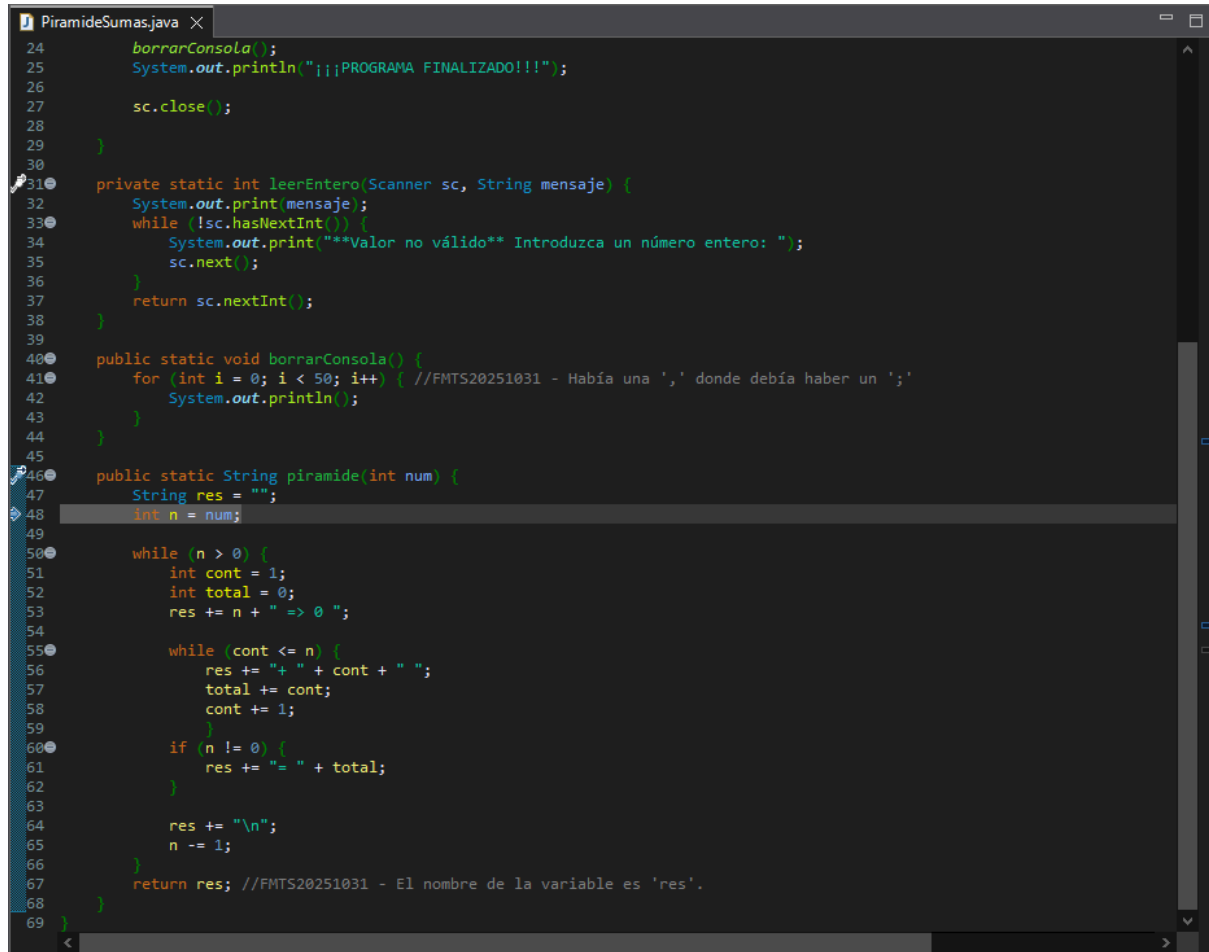
Una vez hecho esto, Eclipse nos preguntará si queremos cambiar de vista, si le decimos que sí, cambiará a la vista de depuración.



Algunas de las vistas más importantes son:

Editor de código:

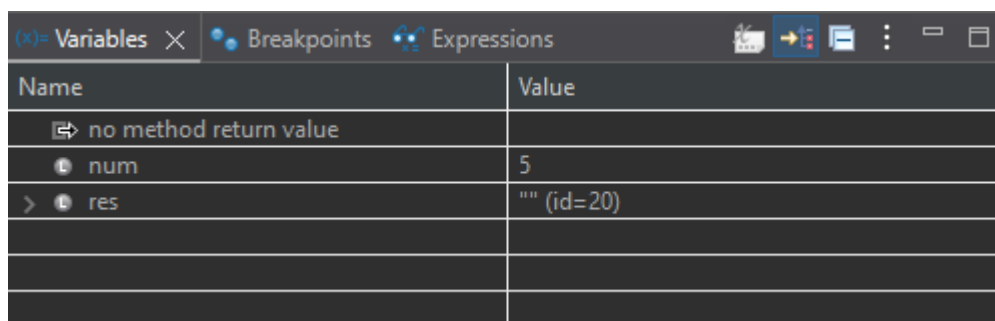
Es evidente que sin la vista del editor el IDE no sería funcional, porque no podríamos modificar código con él.



```
24     borrarConsola();
25     System.out.println("¡¡¡PROGRAMA FINALIZADO!!!");
26
27     sc.close();
28
29 }
30
31 private static int leerEntero(Scanner sc, String mensaje) {
32     System.out.print(mensaje);
33     while (!sc.hasNextInt()) {
34         System.out.print("**Valor no válido* Introduzca un número entero: ");
35         sc.next();
36     }
37     return sc.nextInt();
38 }
39
40 public static void borrarConsola() {
41     for (int i = 0; i < 50; i++) { //FMTS20251031 - Había una ',' donde debía haber un ';'
42         System.out.println();
43     }
44 }
45
46 public static String piramide(int num) {
47     String res = "";
48     int n = num;
49
50     while (n > 0) {
51         int cont = 1;
52         int total = 0;
53         res += n + " => 0 ";
54
55         while (cont <= n) {
56             res += "+" + cont + " ";
57             total += cont;
58             cont += 1;
59         }
60         if (n != 0) {
61             res += "=" + total;
62         }
63
64         res += "\n";
65         n -= 1;
66     }
67     return res; //FMTS20251031 - El nombre de la variable es 'res'.
68 }
69 }
```

Vista de Variables:

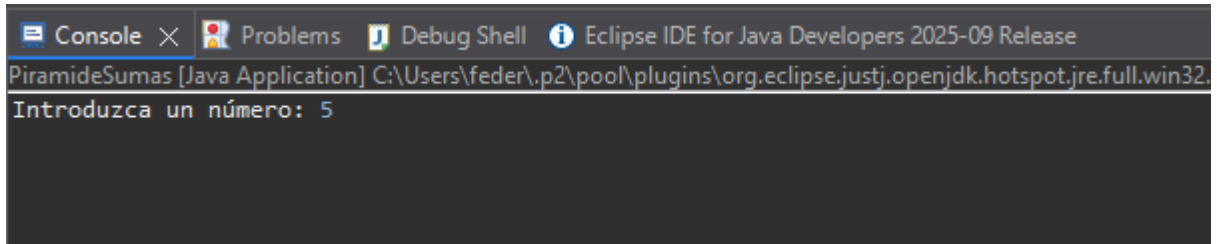
Es importante porque nos permite ver el estado de las variables existentes en cada momento.



Variables	
Name	Value
no method return value	
num	5
res	"" (id=20)

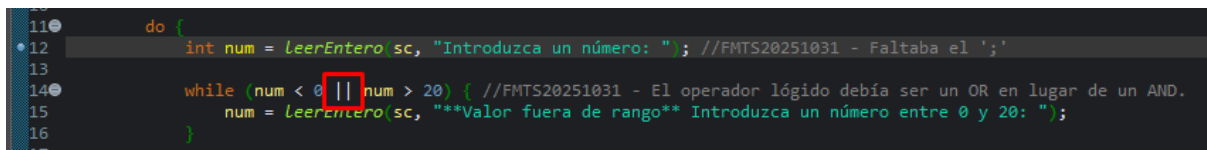
Consola:

Es necesaria para interactuar con el programa, tanto para recibir como para introducir información.

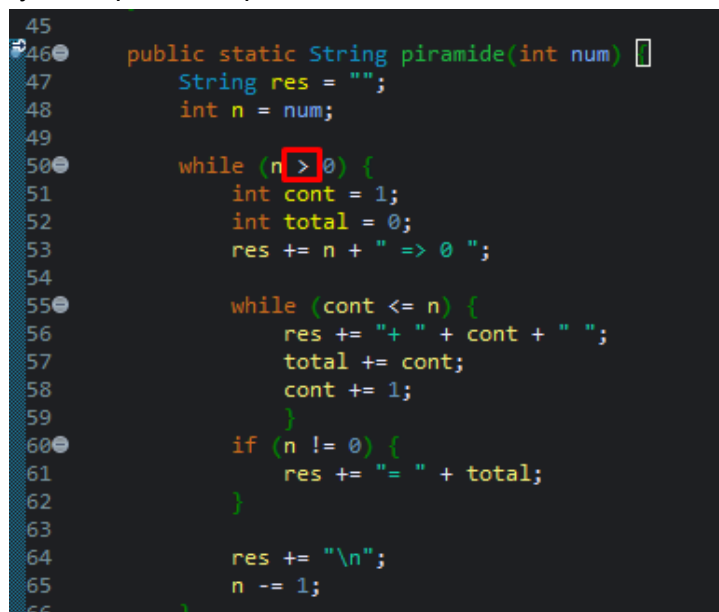


Errores lógicos encontrados:

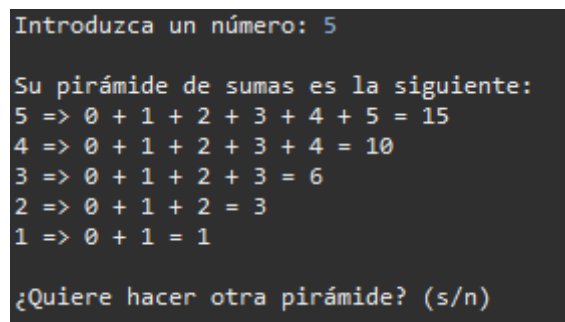
- El programa comprobaba si el número introducido es menor que cero Y mayor que 20, devolviendo siempre 'false', evitando que se muestre el mensaje de fuera de rango.



- El segundo bucle de la función piramide comprobaba si 'cont' era menor que 'n', en lugar de mayor, lo que lo atrapaba en un bucle infinito.



Una vez hecho esto podemos ver en la captura que la pirámide de sumas se muestra correctamente en consola.



¿Dónde colocaste los breakpoints y qué valores viste que confirmaron el fallo?

1. Coloqué el primer breakpoint en la línea 12 para depurar desde el principio y observar el comportamiento del programa.

```

11  do {
12  int num = LeerEntero(sc, "Introduzca un número: "); //FMTS20251031 - Faltaba el ';'
13
14  while (num < 0 || num > 20) { //FMTS20251031 - El operador lógico debía ser un OR en lugar de un AND.
15      num = LeerEntero(sc, "***Valor fuera de rango** Introduzca un número entre 0 y 20: ");
16  }
17
18  System.out.println("\nSu pirámide de sumas es la siguiente:\n" + piramide(num));
19
20  System.out.print("¿Quiere hacer otra pirámide? (s/n) ");
21  resp = sc.next().trim().toUpperCase(); //FMTS20251031 - Sobraba la declaración de la variable porque ya est
22
23  } while (resp.equals("S")); //FMTS20251031 - Faltaba cerrar la instrucción con ');'
24  borrarConsola();
25  System.out.println("¡¡¡PROGRAMA FINALIZADO!!!");
26
27  sc.close();

```

2. He colocado un segundo breakpoint al comienzo de la función piramide para analizar su comportamiento, pero como necesita el valor de num, la depuración comienza en la misma línea 12.

```

45  public static String piramide(int num) {
46  String res = "";
47  int n = num;
48
49  while (n > 0) {
50  int cont = 1;
51  int total = 0;
52  res += n + " => 0 ";
53
54  while (cont <= n) {
55  res += " + " + cont + " ";
56  total += cont;
57  cont += 1;
58  }
59
60  if (n != 0) {
61  res += " = " + total;
62  }
63
64  res += "\n";
65  n -= 1;
66
67  return res; //FMTS20251031 - El nombre de la variable es 'res'.
68  }
69  }

```

3. Se comprobaba que 'n' fuera menor que 0, no mayor, por lo que al introducir un número positivo nunca se ejecutaba el while.

The screenshot shows the IDE with the code from the previous blocks. The Variables window on the right displays the following data:

Name	Value
no method return value	
num	3
res	"" (id=28)
n	3

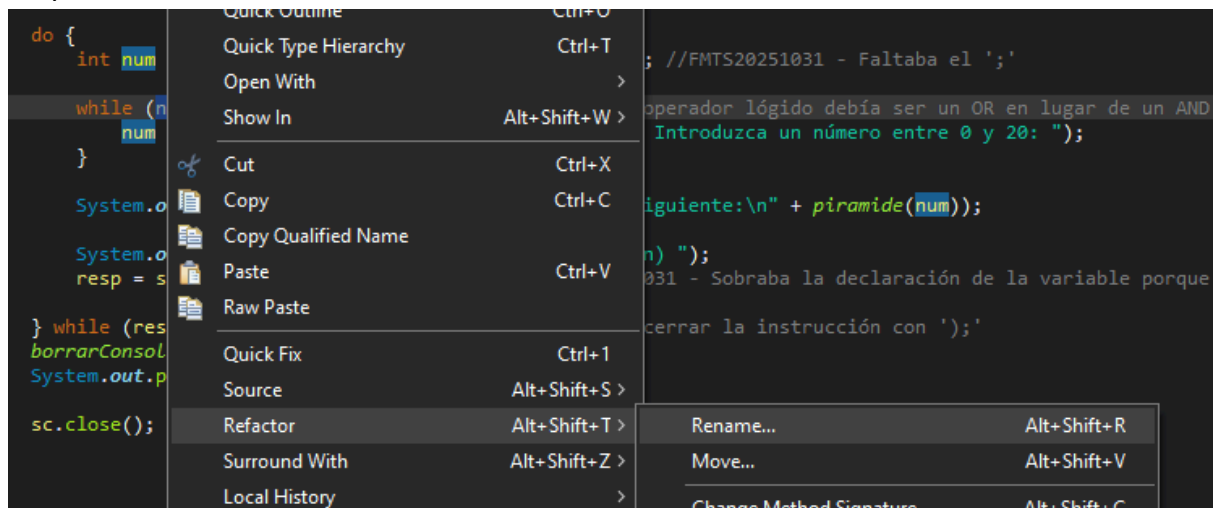
¿En qué situación debemos usar Step Into, Step Over, Step Return y Resume?

- **Step Into:** Si la línea actual llama a otro método, el depurador entra dentro de ese método y ejecuta su primera línea.
- **Step Over:** Ejecuta la línea actual completa, incluyendo llamadas a métodos, pero sin entrar en ellos.
- **Step Return:** Ejecuta el resto del método actual y regresa a la línea que lo llamó.
- **Resume:** Reanuda la ejecución sin detenerse paso a paso, hasta el siguiente breakpoint o el fin del programa.

2.4)Refactorización: renombrar

Refactorización

He utilizado la herramienta rename del IDE, lo que es útil porque ahorra tiempo y evita errores. Solo hay que seleccionar el elemento que se quiere renombrar, hacer clic derecho y elegir la opción 'Rename', una vez hecho esto solo hay que modificar el nombre y aceptar los cambios.



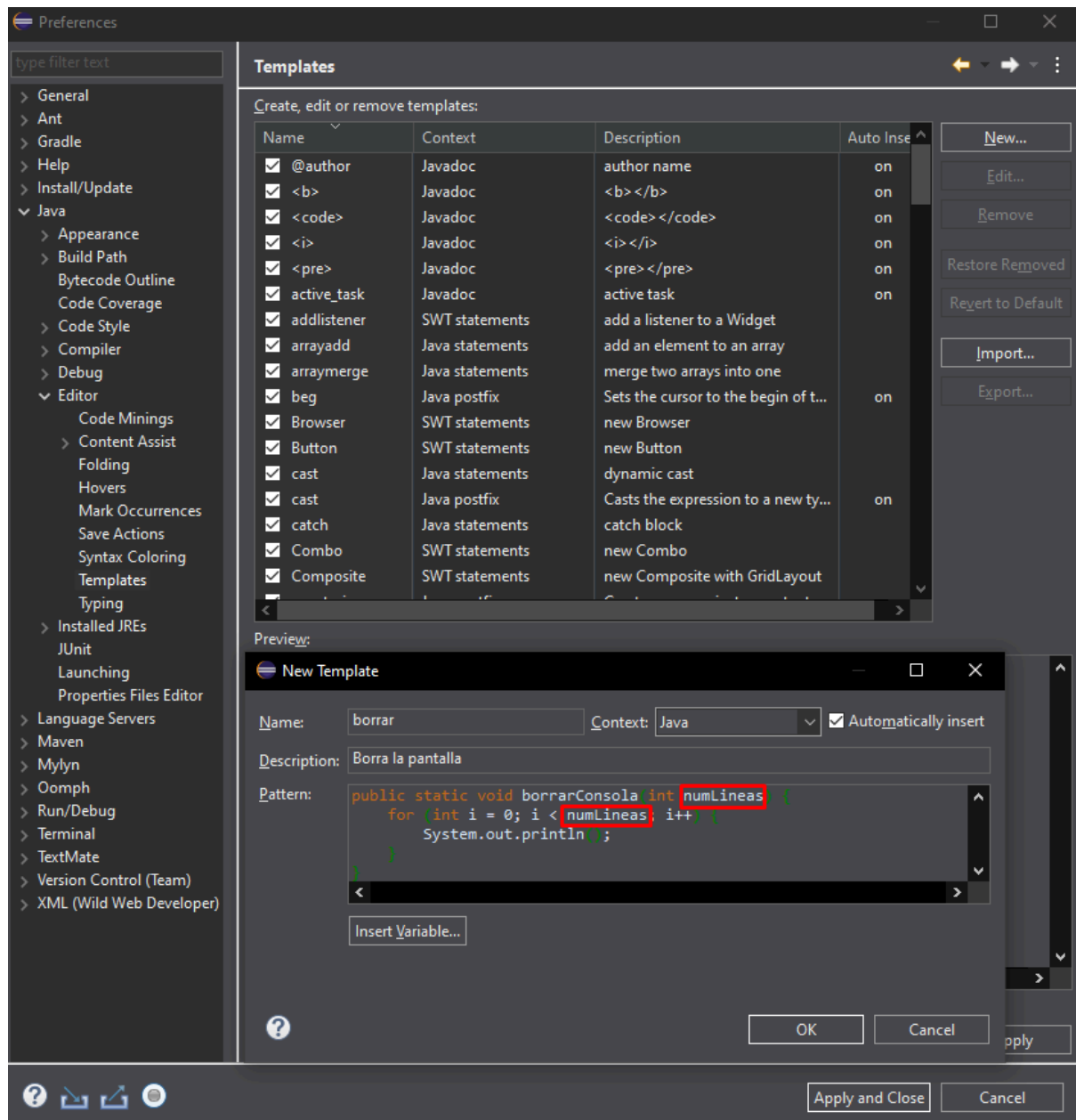
2.5)Crear una plantilla (template) “borrar”

Definición de la plantilla

Si seguimos la ruta Window > Preferences > Java > Editor > Templates y pulsamos sobre el botón 'New...' se abrirá una ventana en la que podremos definir la nueva plantilla 'borrar'.

Pasamos 'numLineas' como un parámetro de la función de la plantilla y lo utilizamos también como condición.

Para guardar la plantilla, pulsamos el botón 'OK'.



¿Para qué sirve una plantilla?

Las plantillas (templates) en Eclipse sirven para ahorrar tiempo y reducir errores al escribir código. Básicamente, son fragmentos de código predefinidos que puedes insertar automáticamente con solo escribir un atajo y presionar Ctrl + Espacio.