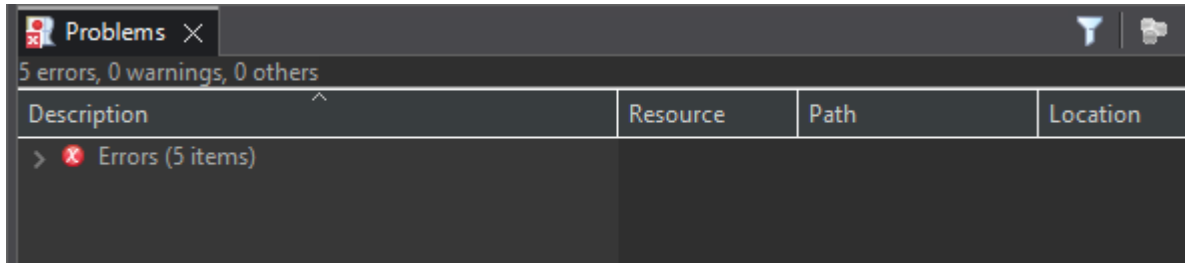
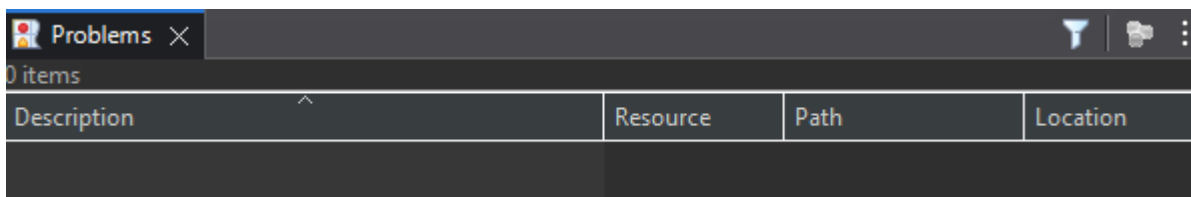


2.2 Corregir errores sintácticos (y documentarlos)

Incluyo la captura con la vista de Problems antes de la corrección



Después de la corrección



Corrección de errores:

- Falta de añadir al final de la instrucción punto y coma

```
int num = leerEntero(sc, "Introduzca un número: ")
```

Solución al fallo

```
int num = leerEntero(sc, "Introduzca un número: ");
```

- Declara la variable dos veces con String (se declara al principio del código)

```
String resp = sc.next().trim().toUpperCase();
```

Solución al fallo

```
resp = sc.next().trim().toUpperCase();
```

- Falta un paréntesis por añadir

```
while (resp.equals("S"));
```

Solución al fallo

```
while (resp.equals("S"));
```

- Error poniendo , en lugar de ; en la estructura del for

```
for (int i = 0; i < 50, i++) {
```

Solución al fallo

```
for (int i = 0; i < 50; i++) {
```

- Cambio nombre a una variable, ya que está declarada con otro

```
return resultado;
```

Solución al fallo

```
return res;
```

- **¿Qué es un error sintáctico?**

Es cuando escribimos mal, la estructura de un bucle, nos falta un punto y coma, mal declaradas las variables...

Es decir escribimos de manera incorrecta según las reglas del lenguaje que usamos

- **¿Cuándo los visualizamos?**

Normalmente los IDE, nos indican los errores de sintaxis en con un símbolo rojo además de subrayarlo en color rojo

- **¿Podemos depurar con errores de sintaxis?**

No, ya que el código da error y no se puede compilar.

Podemos depurar una vez corregimos los errores sintácticos para comprobar las condiciones, valores de las variables, si es correcto funcionamiento de un bucle...

2.3 Corregir errores lógicos usando el Depurador

- Cambio la condición, ya que es imposible que se cumpla. Cambio && por ||

```
while (num < 0 && num > 20) {
```

Solución al fallo

```
while (num < 0 || num > 20) {
```

Colocación del breakpoint para darme cuenta que nunca se cumple la condición

```
12
13 ● while (num < 0 || num > 20) {
14     num = leerEntero(sc, "**Va
15 }
16
```

- Colocación del breakpoint y corrección de los distintos fallos para depurar

```
45 ● public static String piramide(int num) {
46     String res = "";
47     int n = num;
48
49 ● while (n >= 0) { //AMC20251031 - Hay que cambiar la condición
50     int cont = 1;
51     int total = 0;
52     res += n + " => 0 ";
53
54 ● while (cont <= n) { //AMC20251031 - Faltan por añadir los corchetes
55     res += "+" + cont + " ";
56     total += cont;
57     cont += 1;
58 }
59
60 ● if (n != 0) { /*AMC20251031 - Esta condición no hay que modificarla, no se porque en el examen en
61     clase, supongo que por el tiempo y nervios puse otra condición*/
62     res += "=" + total;
63 }
64
65     res += "\n";
66     n -= 1;
67 }
68
```

Preguntas:

- **Proporciona una mini guía de cómo realizar la depuración de un programa (si te es más fácil, elige uno de los fallos lógicos y úsalo de ejemplo)**

La depuración se realiza para encontrar los distintos fallos lógicos, no los sintácticos, ya que con los fallos lógicos si que se puede ejecutar el código pero puede que no haga lo que queramos y puede fallar la lógica y el funcionamiento

Para la depuración tenemos que colocar un punto de ruptura, le damos a “Debug”, y mediante F5, F6... vamos viendo los pasos, que condiciones se cumplen y los valores de las variables

- **Agrega capturas con los puntos de ruptura y las vistas que consideres más importantes a la hora de depurar un programa y por que.**

La vista que me parece más importante es la de Variables, para ver el valor de estas y ver si el código hace su objetivo de manera correcta.

[illegible]

En las correcciones lógicas indico los puntos de ruptura que he usado para su corrección, pero vuelvo a añadir las capturas.

```
13 while (num < 0 || num > 20) { //AMC20251031 - Cor
14     num = LeerEntero(sc, "**Valor fuera de rango**
15 }
16
```

```
45 public static String piramide(int num) {
46     String res = "";
47     int n = num;
48
49     while (n >= 0) { //AMC20251031 - Hay que cambiar la condición
50         int cont = 1;
51         int total = 0;
52         res += n + " => 0 ";
53
54         while (cont <= n) { //AMC20251031 - Faltan por añadir los corchetes
55             res += "+ " + cont + " ";
56             total += cont;
57             cont += 1;
58         }
59
60         if (n != 0) { /*AMC20251031 - Esta condición no hay que modificarla, no se porque en el examen en
61             clase, supongo que por el tiempo y nervios puse otra condición*/
62             res += "= " + total;
63         }
64
65         res += "\n";
66         n -= 1;
67     }
68 }
```

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

Coloque un breakpoint en la función pirámide para ver porque esta no se mostraba, cual era el valor de las variables y ver en qué condiciones si entraba y en cuáles no

Otro breakpoint lo coloque en la condición de los valores fuera de rango para ver porque la condición estaba mal

La vista de Variables me ayuda a todos los valores de estas y entender mejor el funcionamiento y control del código

- ¿En qué situación debemos usar Step Into (F5), Step Over (F6), Step Return (F7) y Resume (F8)?

Step Into (F5) → Lo usamos para entrar en la función o método en el que nos encontramos

Step Over (F6) → Ejecuta la línea completa sin entrar en la función ni método

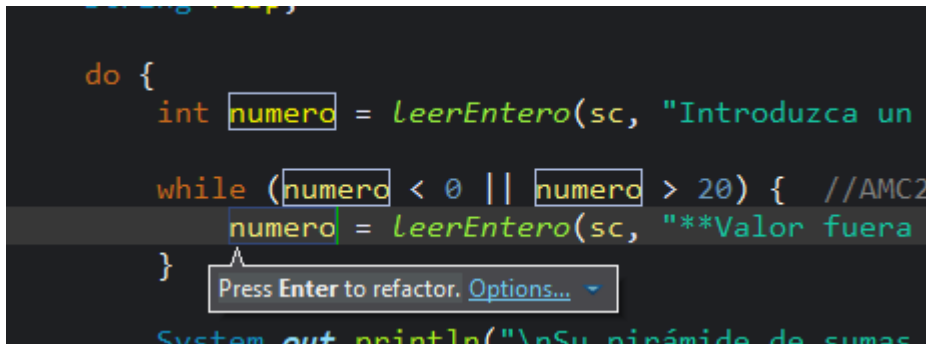
Step Return (F7) → Nos sirve para salir del método o función en la que nos encontremos

Resume (F8) → Salta al siguiente breakpoint

2.4 Refactorización: renombrar num → numero

Pasos:

Seleccionamos la variable a renombrar, clic derecho → Refactor → Rename



```

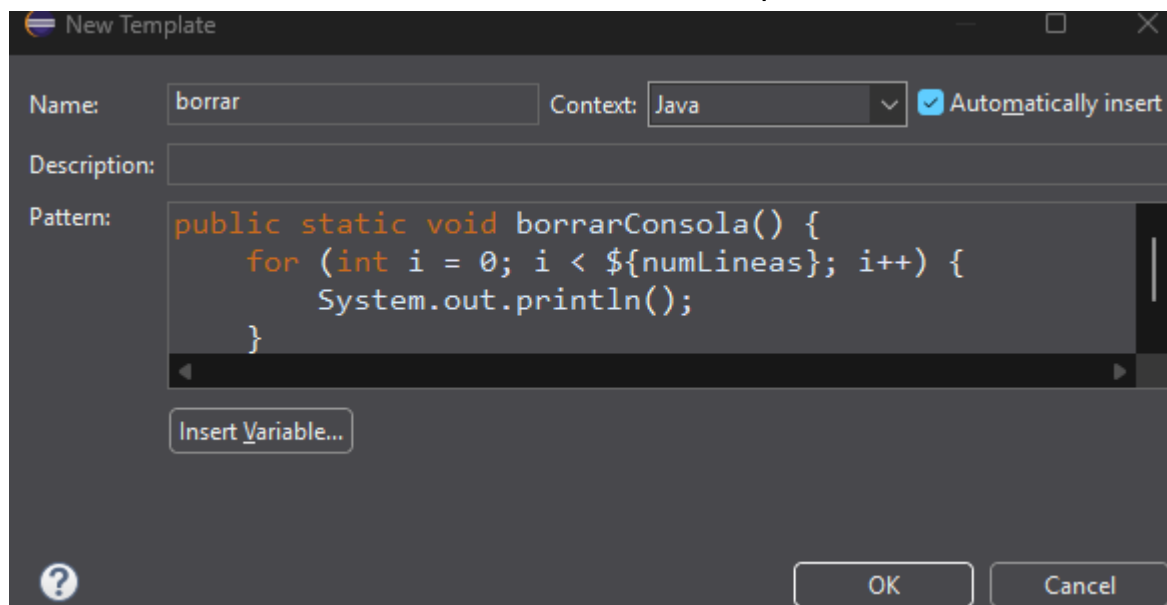
do {
    int numero = LeerEntero(sc, "Introduzca un n
    while (numero < 0 || numero > 20) { //AMC20
        numero = LeerEntero(sc, "**Valor fuera d
    }
}
System.out.println("\nSu pirámide de sumas
  
```

Lo he realizado con la herramienta de refactorización que viene en Eclipse, es una buena práctica para no tener problemas al cambiar el nombre a una variable ya que hacerlo manual en un código largo es probable que se nos olvide de renombrar alguna, de este método evitamos problemas y es mucho más rápido

2.5 Crear una plantilla (template) “borrar”

Para crear una plantilla tenemos que seguir los siguientes pasos:

Window → Preferences → Java → Editor → Templates → New



```
public static void borrarConsola() {  
    for (int i = 0; i < 25; i++) { //AMC2025  
        System.out.println();  
    }  
}
```

Las plantillas son atajos de código, que permiten insertar fragmentos de código de forma rápida. Ahorran tiempo y reducen los errores de escritura