

Programación

Bucles en Java, aleatorios, printf y depuración

Unidad 5

Jesús Alberto Martínez
versión 0.2



Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.
Basado en los apuntes del CEEDCV



Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención



Interesante

Unidad 5. Bucles en Java, aleatorios, printf y depuración

1	Introducción.....	3
2	Bucle for.....	4
3	Bucle while.....	5
4	Bucle do-while.....	7
5	Control dentro del bucle.....	9
5.1	Break.....	9
5.2	Continue.....	9
6	Bucles anidados.....	10
6.1	Finalización de bucles anidados.....	11
7	Ejemplos.....	12
7.1	Ejemplo 1.....	12
7.2	Ejemplo 2.....	13
8	Generación de números aleatorios.....	15
8.1	Math.random().....	15
8.2	Random.nextInt.....	15
8.3	ThreadLocalRandom.....	16
9	printf() guía rápida.....	16
9.1	Cadena con formato.....	16
9.2	Modificadores.....	16
9.3	Anchura.....	17
9.4	Precisión.....	17
9.5	Caracteres de conversión.....	17
9.6	Ejemplos.....	17
9.7	Más información.....	17
10	Depuración del código.....	18
10.1	Poner un punto de interrupción.....	18
10.2	Iniciar la depuración.....	18
10.3	Controlando la ejecución del programa.....	19
10.4	Observaciones o Expresiones.....	19
10.5	Ampliación.....	20

1 Introducción

Los bucles son estructuras de repetición, bloques de instrucciones que se repiten un número de veces mientras se cumpla una condición o hasta que se cumpla una condición.

Un bloque de instrucciones se encontrará encerrado mediante llaves {.....} si existe más de una instrucción al igual que suceden las estructuras alternativas (if... else... etc).

Existen tres construcciones para estas estructuras de repetición:

- Bucle for
- Bucle while
- Bucle do-while

Todo problema que requiera repetición puede hacerse con cualquiera de los tres, pero según el caso suele ser más sencillo o intuitivo utilizar uno u otro.

Como regla general es recomendable:



Utilizar el bucle **for** cuando se conozca de antemano el número exacto de veces que ha de repetirse el bloque de instrucciones.



Utilizar el bucle **while** cuando no sabemos el número de veces que ha de repetirse el bloque y es posible que no deba ejecutarse ninguna vez.



Utilizar el bucle **do-while** cuando no sabemos el número de veces que ha de repetirse el bloque y deberá ejecutarse al menos una vez.

Estas reglas son generales y algunos programadores se sienten más cómodos utilizando principalmente una de ellas. Con mayor o menor esfuerzo, puede utilizarse cualquiera de las tres indistintamente.

2 Bucle for

El bucle *for* se codifica de la siguiente forma:

Código	Ordinograma
<pre>for (inicialización ; condición ; incremento) { bloque acciones; }</pre>	

La cláusula **inicialización** es una instrucción que se ejecuta una sola vez al inicio del bucle, normalmente para inicializar un contador. Por ejemplo `int i = 1;`

La cláusula **condición** es una expresión lógica que se evalúa al inicio de cada iteración del bucle. En el momento en que dicha expresión se evalúe a false se dejará de ejecutar el bucle y el control del programa pasará a la siguiente instrucción (a continuación del bucle `for`). Se utiliza para indicar la condición en la que quieres que el bucle continúe. Por ejemplo `i <= 10;`

La cláusula **incremento** es una instrucción que se ejecuta al final de cada iteración del bucle (después del bloque de instrucciones). Generalmente se utiliza para incrementar o decrementar el contador. Por ejemplo `i++;` (incrementar `i` en 1).

Ejemplo 1: Bucle que muestra por pantalla los números naturales del 1 al 10:

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

En la inicialización utilizamos `int i=1` para crear la variable `i` con un valor inicial de 1.

La condición `i<=10` indica que el bucle debe repetirse mientras `i` sea menor o igual a 10.

La actualización `i++` indica que, al final de cada iteración, `i` debe incrementarse en 1.

Ejemplo 2: Programa que muestra los números naturales (1,2,3,4,5,6,...) hasta un número introducido por teclado.

```
6 public static void main(String[] args) {  
7     Scanner sc = new Scanner(System.in);  
8     int max;  
9     System.out.print("Introduce el número máximo: ");  
10    max = sc.nextInt();  
11    for (int i = 1; i <= max; i++) {  
12        System.out.println("Número: " + i);  
13    }  
14 }  
15 }  
16
```

Siendo la salida:

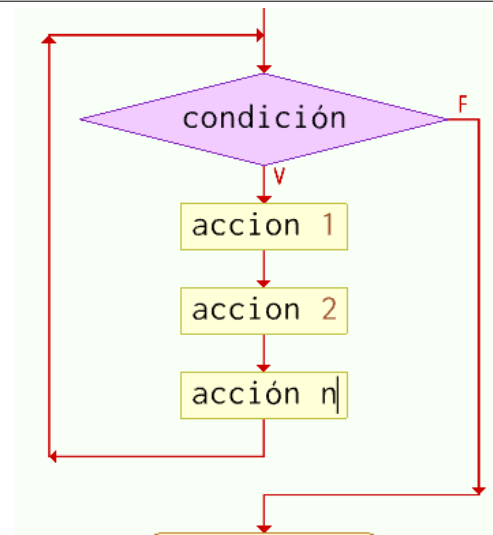
```
run:  
Introduce el número máximo: 5  
Número: 1  
Número: 2  
Número: 3  
Número: 4  
Número: 5  
BUILD SUCCESSFUL (total time: 6 seconds)
```

3 Bucle while

El bucle while se codifica de la siguiente forma:

Código	Ordinograma
--------	-------------

```
while (condición) {
    bloque acciones;
}
```



El bloque de instrucciones se ejecuta mientras se cumple una condición (mientras **condición** se evalúe a true). La condición se comprueba ANTES de empezar a ejecutar por primera vez el bucle, por lo que si se evalúa a false en la primera iteración, entonces el bloque de acciones no se ejecutará ninguna vez.

El mismo ejemplo 2 anterior hecho con un bucle while sería:

```

7   public static void main(String[] args) {
8       Scanner sc = new Scanner(System.in);
9       int max, cont;
10      System.out.print("Introduce el número máximo: ");
11      max = sc.nextInt();
12      cont = 1;
13      while (cont <= max) {
14          System.out.println("Número: " + cont);
15          cont++;
16      }
17  }
```

```

run:
Introduce el número máximo: 5
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
BUILD SUCCESSFUL (total time: 6 seconds)
```

Y la salida:

4 Bucle do-while

El bucle while se codifica de la siguiente forma:

Código	Ordinograma
<pre>do { bloque acciones; } while (condición);</pre>	<pre> graph TD Start(()) --> A1[acción 1] A1 --> A2[acción 2] A2 --> An[acción n] An --> Cond{condición} Cond -- V --> End(()) Cond -- F --> Start </pre>

En este tipo de bucle, el bloque de instrucciones se ejecuta siempre al menos una vez, y ese bloque de instrucciones se ejecutará mientras **condición** se evalúe a true.

⚡ Por ello en el bloque de instrucciones deberá existir alguna que, en algún momento, haga que *condición* se evalúe a *false*. ¡Si no el bucle no acabaría nunca!

El mismo **ejemplo 2** anterior hecho con un bucle do-while sería:

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int max, cont;  
    System.out.print("Introduce el número máximo: ");  
    max = sc.nextInt();  
    cont = 1;  
  
    do {  
        System.out.println("Número: " + cont);  
        cont++;  
    } while (cont <= max);  
}
```

Y la salida:

```
run:  
Introduce el número máximo: 5  
Número: 1  
Número: 2  
Número: 3  
Número: 4  
Número: 5  
BUILD SUCCESSFUL (total time: 6 seconds)
```


5 Control dentro del bucle

5.1 Break

La instrucción `break` ya la hemos visto en los apuntes, dentro de una sentencia `switch` es la encargada de finalizar la ejecución del bloque de instrucciones y salir del `switch`.

De forma parecida, break nos va a servir dentro de un bucle para salirnos del bucle que la contiene.

```
13 public static void main(String args[]) {
14     for (int i = 1; i <= 10; i++) {
15         if (i == 5) {
16             break;
17         }
18         System.out.print(i + " ");
19     }
20     System.out.println("\nFin del bucle");
21 }
```

Dará como resultado

```

--- exec-maven-plugin:3.0.0
1 2 3 4
Fin del bucle
-----
BUILD SUCCESS

```

Al ejecutar la instrucción break finaliza la ejecución del bucle.

5.2 Continue

La instrucción continue dentro de un bucle hace que finalice la ejecución de esa iteración (repetición) volviendo al comienzo del bucle.

```

13 public static void main(String args[]) {
14     for (int i = 1; i <= 10; i++) {
15         if (i == 5) {
16             continue;
17         }
18         System.out.print(i + " ");
19     }
20     System.out.println("\nFin del bucle");
21 }

```

Dará como resultado

```

--- exec-maven-plugin
1 2 3 4 6 7 8 9 10
Fin del bucle
-----

```

6 Bucles anidados

Un bucle anidado es un bucle que se encuentra dentro de otro bucle.

```

13 public static void main(String args[]) {
14     int tabla = 1;
15     while (tabla < 4) {
16         System.out.println("TABLA DEL " + tabla);
17         for (int i = 1; i <= 10; i++) {
18             System.out.print(i * tabla + " ");
19         }
20         System.out.println();
21         tabla++;
22     }
23 }

```

Dará como salida

```

--- exec-maven-plugin:3.0.0:exec {
TABLA DEL 1
1 2 3 4 5 6 7 8 9 10
TABLA DEL 2
2 4 6 8 10 12 14 16 18 20
TABLA DEL 3
3 6 9 12 15 18 21 24 27 30
-----

```

6.1 Finalización de bucles anidados

Cuando usamos break o continue saldremos del bucle en el que se encuentra.

```

13  public static void main(String args[]) {
14      int tabla = 1;
15      while (tabla < 4) {
16          System.out.println("TABLA DEL " + tabla);
17          for (int i = 1; i <= 10; i++) {
18              if (tabla == 2 && i == 4) {
19                  break;
20              }
21              System.out.print(i * tabla + " ");
22          }
23          System.out.println();
24          tabla++;
25      }
26  }
```

Salida:

```

--- exec-maven-plugin:3.0.0:
TABLA DEL 1
1 2 3 4 5 6 7 8 9 10
TABLA DEL 2
2 4 6
TABLA DEL 3
3 6 9 12 15 18 21 24 27 30
-----
```

Pero también podemos salir a cualquier otro bucle a través de etiquetas:

```

13  public static void main(String args[]) {
14      int tabla = 1;
15      salida_total:
16      while (tabla < 4) {
17          System.out.println("TABLA DEL " + tabla);
18          for (int i = 1; i <= 10; i++) {
19              if (tabla == 2 && i == 4) {
20                  break salida_total;
21              }
22              System.out.print(i * tabla + " ");
23          }
24          System.out.println();
25          tabla++;
26      }
27  }
```

Salida:

```

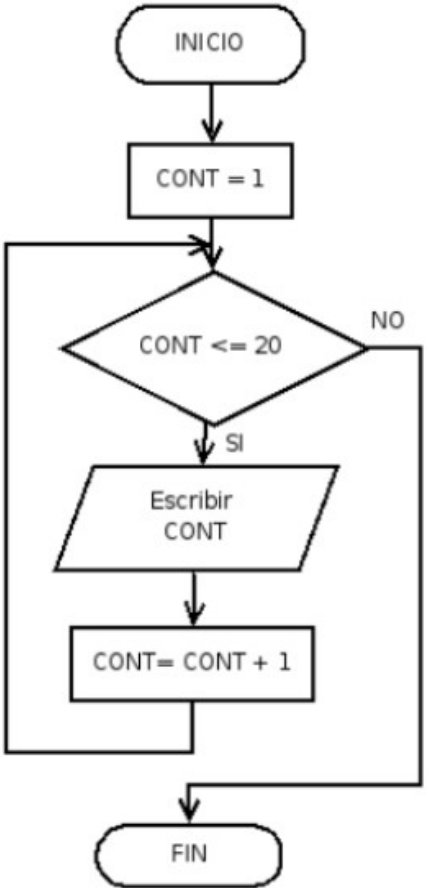
--- exec-maven-plugin:3
TABLA DEL 1
1 2 3 4 5 6 7 8 9 10
TABLA DEL 2
2 4 6
-----

```

7 Ejemplos

7.1 Ejemplo 1

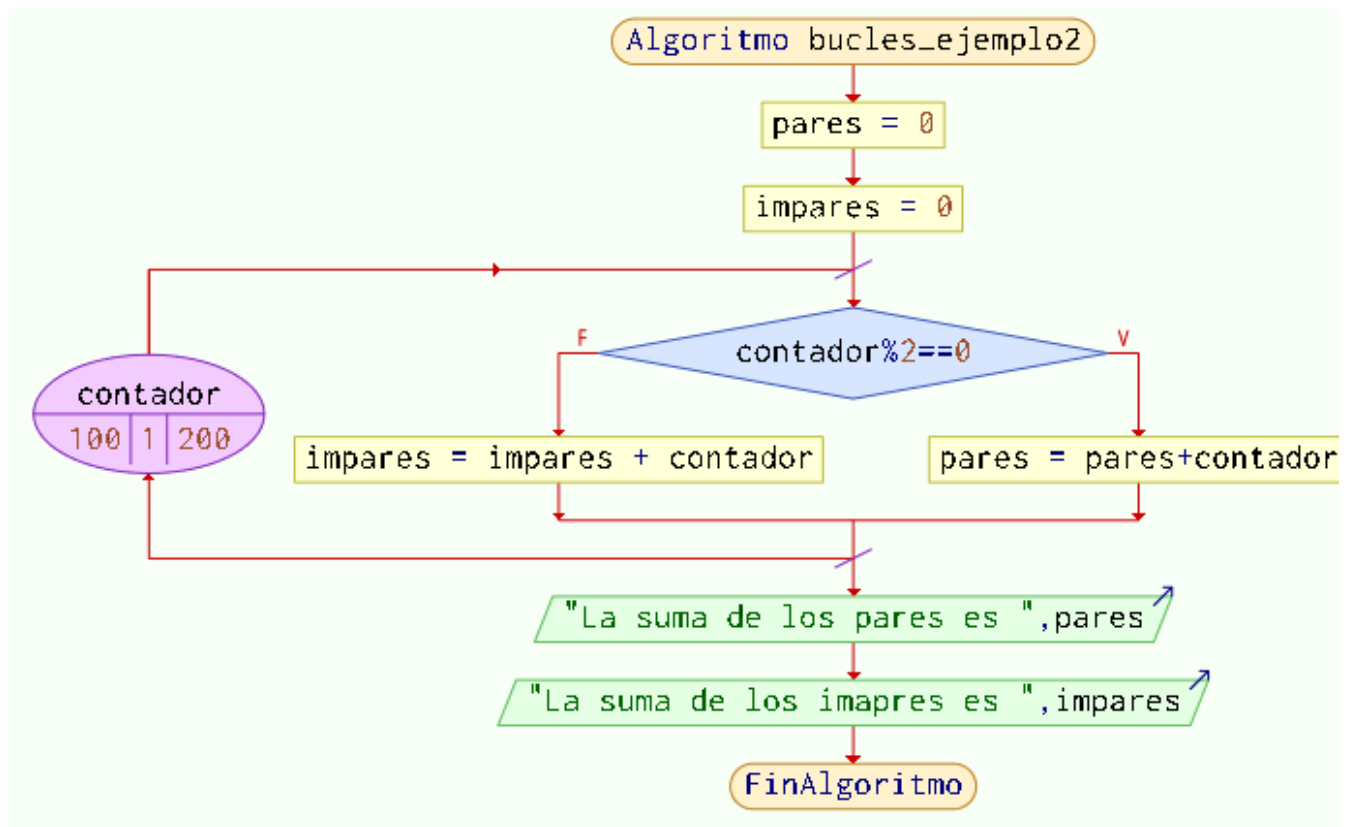
Programa que muestre por pantalla los 20 primeros números naturales (1, 2, 3... 20).

Ordinograma	Código
 <pre> graph TD INICIO([INICIO]) --> CONT1[CONT = 1] CONT1 --> Decision{CONT <= 20} Decision -- SI --> Escribir[/Escribir CONT/] Escribir --> Increment[CONT = CONT + 1] Increment --> Decision Decision -- NO --> FIN([FIN]) </pre>	<pre> 12 public class Ejercicio1 { 13 14 public static void main(String[] args) { 15 int cont; 16 17 for(cont=1; cont<=20; cont++) 18 System.out.print(cont + " "); 19 20 System.out.print("\n"); 21 } 22 } </pre> <p>Salida:</p> <pre> run: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 BUILD SUCCESSFUL (total time: 0 seconds) </pre>

7.2 Ejemplo 2

Programa que suma independientemente los pares y los impares de los números comprendidos entre 100 y 200.

Ordinograma:



Código:

```
1  ▶ public class T5Ejemplo2 {  
2  ▶  ▶ public static void main(String[] args) {  
3      int pares,impares,cont;  
4  
5      pares=0;  
6      impares=0;  
7  
8      for (cont=100;cont<=200;cont++){  
9          if (cont % 2==0)  
10             pares=pares+cont;  
11          else  
12             impares=impares+cont;  
13      }  
14  
15      System.out.println("La suma total de los pares es "+pares);  
16      System.out.println("La suma total de los impares es "+impares);  
17  }  
18 }
```

Salida:

```
▶ ↑ "C:\Program Files\Java\jdk-17.0.3.1\bin  
⚙ ↓ La suma total de los pares es 7650  
■ ⏸ La suma total de los impares es 7500
```

8 Generación de números aleatorios

En Java podemos generar números pseudo-aleatorios de varias formas.

8.1 Math.random()

Una de ellas es mediante el método random de la clase Math.

Math.random() nos devuelve un número aleatorio real comprendido entre 0.0 y 1.0, pero en general nosotros lo que vamos a necesitar es generar números aleatorios entre unos determinados límites, por ejemplo entre 0 y el 100, o entre el 1 y el 6, etc, por lo que necesitamos convertir ese número.

`Math.random()*numero`

Nos devolverá un número aleatorio entre 0 y el número, sin incluir número. Como fórmula general será, siendo min y max dos números, y min menor que max.

`Math.random()*(max-min)+min`

nos devolverá un número aleatorio entre min y max, sin incluir max.

Pero tenemos otro problema, que si necesitamos un número entero, tenemos que convertirlo a entero, por ejemplo:

```
int numero=Math.trunc(Math.random()*(max-min)+min);
```

```
int numero=(int)(Math.random()*(max-min)+min);
```

8.2 Random.nextInt

Otra forma de generar números aleatorios es a través de la clase Random y su método nextInt().

Por ejemplo, para generar un número aleatorio entre 0 y 9 usaríamos

```
Random objetoRandom = new Random();  
int numeroAleatorio=objetoRandom.nextInt(10);
```

De forma general para generar un número entre un rango dado `nextInt(max - min + 1) + min`

Para generar un número aleatorio entre 5 y 10 usaríamos

```
int numeroAleatorio=objetoRandom.nextInt(10-5+1) + 5;
```

8.3 ThreadLocalRandom

La última forma que vamos a ver es usando la clase ThreadLocalRandom, su uso general sería:

```
int numeroAleatorio = ThreadLocalRandom.current().nextInt(min, max + 1);
```

Esta forma es la más sencilla y la que se prefiere actualmente.

Para generar un número aleatorio entre 5 y 10 usaríamos

```
int numeroAleatorio = ThreadLocalRandom.current().nextInt(5, 11);
```

9 Salida con formato. printf() guía rápida

Otra forma de mostrar texto en la salida estandar es a través de printf, donde podemos controlar con mucha más precisión el formato de la salida.

System.out.printf("Cadena con formato" [,arg1,arg2,...]);

Por ejemplo, queremos mostrar la división de 3 entre 7

```
System.out.println("División de 3/7: "+ 3/7);
// Nos devuelve 0, es división entre números enteros, devuelve un número entero
System.out.println("División de 3/7: "+3.0/7.0);
// Nos devuelve 0.42857142857142855
System.out.printf("División de %d/%d: %.2f %n",3,7,3.0/7.0);
```

9.1 Cadena con formato

Está formada por literales y especificadores de formato. Los argumentos sólo se necesitan si hay especificadores de formato dentro de la cadena. Los especificadores de formato incluyen: modificadores, anchura, precisión, y carácter de conversión según la secuencia

% [modificadores] [anchura] [.precisión] carácter de conversión

9.2 Modificadores

- - : justificación a la izquierda (por defecto es a la derecha)
- + : añade el símbolo + o - a un valor numérico
- 0 : fuerza ceros a la izquierda
- , : muestra los indicadores de numeración

- : un espacio mostrará el símbolo – en números negativos

9.3 Anchura

Indica la anchura mínima de caracteres que ocupará la salida de ese campo.

9.4 Precisión

Se usa para restringir la salida. Si la conversión es sobre números reales especifica el número de decimales a mostrar. Si es una cadena de texto especifica el número de caracteres.

9.5 Caracteres de conversión

- d: entero decimal (byet, short, int, long)
- f: números reales (float, double)
- c: carácter, C la mostrará en mayúsculas
- s: cadenas de texto, S la mostrará en mayúsculas
- n: nueva línea (mejor usar %n que \n)

9.6 Ejemplos

```
System.out.printf("Total is: $%,.2f%n", dblTotal);
System.out.printf("Total: %-10.2f: ", dblTotal);
System.out.printf("% 4d", intValue);
System.out.printf("%20.10s\n", stringVal);
String s = "Hello World";
System.out.printf("The String object %s is at hash code %h%n", s, s);
```

9.7 Más información

<https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html>

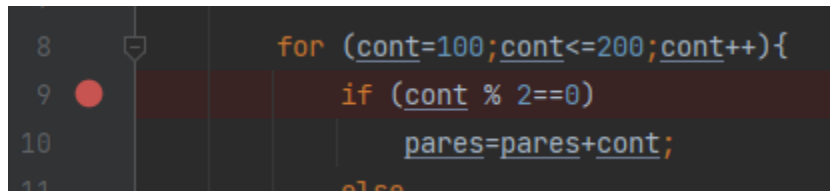
10 Depuración del código

Depurar (Debug) es ejecutar un programa de forma interactiva mientras observamos el código fuente y el valor de las variables durante su ejecución.

Un punto de interrupción (breakpoint) en el código fuente indica donde debe parar la ejecución del programa para depurar. Una vez que el programa se ha detenido puedes comprobar el valor de las variables, cambiarlas, etc.

10.1 Poner un punto de interrupción


Lo podemos establecer con el botón izquierdo del ratón en el margen izquierdo del código fuente (en el margen a la derecha de los números) también podemos pulsar CTRL-F8 cuando estamos sobre la línea donde queremos para la ejecución.

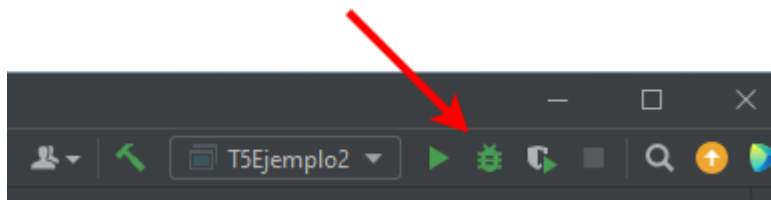


Se indica con un punto rojo a la derecha de los números de línea.

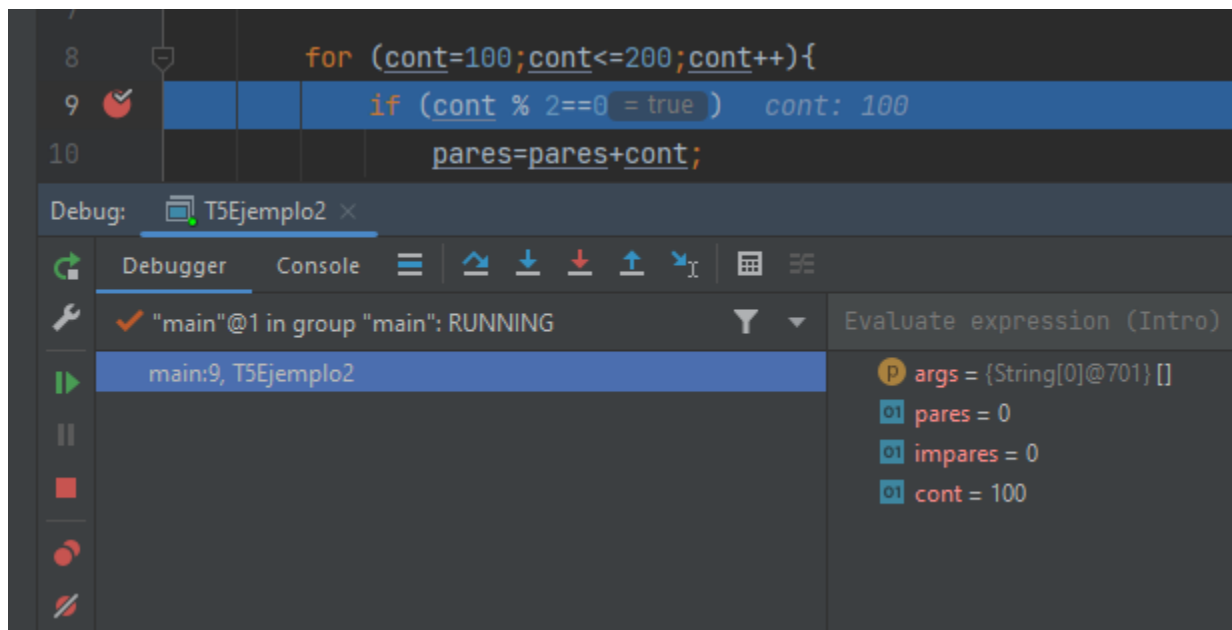
10.2 Iniciar la depuración

Para iniciar la depuración debemos haber establecido al menos un punto de interrupción y pulsar sobre el

icono de debug  o pulsar Mayus-F9, o ir al menú y seleccionar "Run → Debug".



Si la ejecución del programa pasa por un breakpoint, la ejecución del programa se detendrá, mostrará información en la pestaña Debug, y quedará a la espera de que nosotros le indiquemos qué hacer.



10.3 Controlando la ejecución del programa

En la barra de herramientas de la pestaña Debug tenemos los iconos para controlar la ejecución del programa, aunque normalmente es más fácil realizarlo por teclado.



F7 - Ejecuta la línea seleccionada y va a la siguiente línea. Si la línea seleccionada es una llamada a un método, entrará a la ejecución del método.

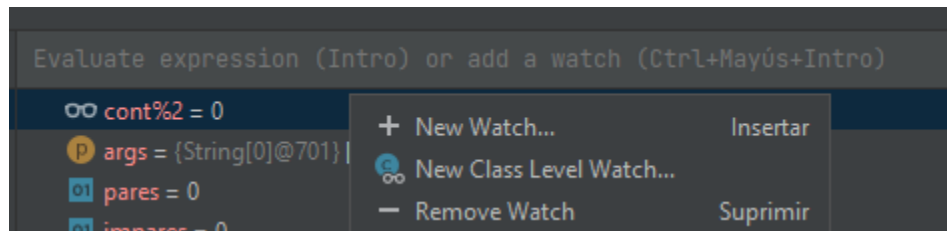
F8 - Ejecuta la línea seleccionada y va a la siguiente línea. Si la línea seleccionada es una llamada a un método no entrará dentro, simplemente lo ejecuta.

Mayúsculas F8 - Sale de la ejecución del método actual y vuelve al método que lo ha llamado.

F9 - Reanudamos la ejecución del programa hasta finalizar o encontrar un nuevo breakpoint.

10.4 Observaciones o Expresiones

Además del valor de las variables, podemos querer evaluar una expresiones, o ver continuamente el valor de la expresión. Para ello pulsaremos con el botón derecho del ratón sobre el panel donde se encuentra el valor de las variables y añadiremos un "New Watch" o evaluaremos expresión



10.5 Ampliación

Tenemos mucha más información en la documentación oficial de IntelliJ IDEA.

<https://www.jetbrains.com/help/idea/debugging-code.html>

11 Ejemplo

Vamos a jugar a un juego y a comprobar cómo vamos de matemáticas. El juego consistirá en que el ordenador generará dos números de dos cifras al azar, y nos pedirá que calculemos la resta del mayor de ellos menos el menor.

Después de contestar, nos dirá si hemos acertado o no, nos dirá cuantas hemos acertado y cuantas hemos fallado, y nos preguntará si queremos seguir jugando hasta que digamos que no.

```

4 ▶ public class EjercicioApuntes {
5 ▶   public static void main(String[] args) {
6       // Definición de variables
7       Scanner entrada=new Scanner(System.in);
8       int numero1, numero2, adivina;
9       int acertadas=0,falladas=0;
10
11       // Generamos los dos números al azar
12       numero1 = ThreadLocalRandom.current().nextInt( bound: 100);
13       numero2 = ThreadLocalRandom.current().nextInt( bound: 100);
14
15       // Ordenamos los números. En numero siempre el mayor
16       if (numero1<numero2){
17           int temporal=numero1;
18           numero1=numero2;
19           numero2=temporal;
20       }
21
22       // Preguntamos al usuario
23       System.out.printf("¿Cuánto es %d - %d? ",numero1,numero2);
24       adivina=Integer.parseInt(entrada.nextLine());
25
26       if (adivina==(numero1-numero2)){
27           acertadas++;
28           System.out.println("BIEN!! CORRECTO!");
29       }
30       else {
31           falladas++;
32           System.out.println("OH!! NO ES CORRECTO!");
33       }
34       System.out.printf("Hemos acertado %d, y hemos fallado %d\n",acertadas,falladas);
35       System.out.println("¿Quieres seguir jugando?");
36   }
37 }

```

¿Que tenemos que repetir?

```
4 public class EjercicioApuntes {
5     public static void main(String[] args) {
6         // Definición de variables
7         Scanner entrada=new Scanner(System.in);
8         int numero1, numero2, adivina;
9         int acertadas=0,falladas=0;
10        String respuesta;
11        boolean continuar=true;
12
13        while (continuar==true) { // la forma correcta sería simplemente while (continuar){
14
15            // Generamos los dos números al azar
16            numero1 = ThreadLocalRandom.current().nextInt( bound: 100);
17            numero2 = ThreadLocalRandom.current().nextInt( bound: 100);
18
19            // Ordenamos los números. En numero siempre el mayor
20            if (numero1 < numero2) {...}
21
22            // Preguntamos al usuario
23            System.out.printf("¿Cuánto es %d - %d? ", numero1, numero2);
24            adivina = Integer.parseInt(entrada.nextLine());
25
26            if (adivina == (numero1 - numero2)) {...} else {...}
27            System.out.printf("Hemos acertado %d, y hemos fallado %d\n", acertadas, falladas);
28            System.out.println("¿Quieres seguir jugando?");
29            respuesta=entrada.nextLine();
30            if (!respuesta.equals("si"))
31                continuar=false;
32            //continuar=entrada.nextLine().equals("si");
33        }
34    }
35 }
```

Otra solución

```
4 public class EjercicioApuntes {
5     public static void main(String[] args) {
6         // Definición de variables
7         Scanner entrada=new Scanner(System.in);
8         int numero1, numero2, adivina;
9         int acertadas=0,falladas=0;
10
11         while (true) { // bucle infinito, equivale a for(;;)|
12             // Generamos los dos números al azar
13             numero1 = ThreadLocalRandom.current().nextInt( bound: 100);
14             numero2 = ThreadLocalRandom.current().nextInt( bound: 100);
15
16             // Ordenamos los números. En numero siempre el mayor
17             if (numero1 < numero2) {
18                 int temporal = numero1;
19                 numero1 = numero2;
20                 numero2 = temporal;
21             }
22
23             // Preguntamos al usuario
24             System.out.printf("¿Cuánto es %d - %d? ", numero1, numero2);
25             adivina = Integer.parseInt(entrada.nextLine());
26
27             if (adivina == (numero1 - numero2)) {
28                 acertadas++;
29                 System.out.println("BIEN!! CORRECTO!");
30             } else {
31                 falladas++;
32                 System.out.println("OH!! NO ES CORRECTO! La respuesta era "+(numero1-numero2));
33             }
34             System.out.printf("Hemos acertado %d, y hemos fallado %d\n", acertadas, falladas);
35             System.out.println("¿Quieres seguir jugando?");
36             if (!entrada.nextLine().equals("si"))
37                 break;
38         }
39     }
40 }
```