4. Estructuras de repetición

Los tipos de estructuras repetitivas o de repetición en Java son:

4.1. While

En los bucles o ciclos while, que vendrían a ser un **mientras** en pseudocódigo, las instrucciones se repiten mientras la condición sea cierta. La condición se comprueba al principio del bucle, por lo que las acciones se pueden ejecutar cero o más veces. Si la condición ya no es cierta la primera vez que se evalúe, ocurrirá que ese código ubicado dentro del bloque del bucle o ciclo while no llegará a ejecutarse.

La ejecución de un bucle o ciclo while sigue los siguientes pasos:

1. Se evalúa la **condición de salida**.

2. Si el resultado es falso (en inglés, *false*), las instrucciones no se ejecutan y el programa sigue ejecutándose por la siguiente instrucción que se encuentre a continuación del bucle o ciclo while.

3. Si el resultado es cierto (en inglés, *true*), se ejecutan las instrucciones y se vuelve al paso 1 para repetir el proceso inicial.

Ejemplo: función estática que imprime línea a línea lo que vamos leyendo del teclado.

```
public static void leerYEscribir()
	{
		/* Vamos a leer por el teclado y volverlo a mostrar por pantalla para
ver que funciona bien */
		Scanner teclado = new Scanner(System.in);
		System.out.println("Bienvenido al programa leer y escribir");
		System.out.println("Vamos a imprimir por pantalla todo lo que
escribas");
		System.out.println("para terminar escribe FIN en mayúsculas");
		// Leemos la primera cadena de texto, si fuera FIN no entraríamos en el
bucle…
		String cadenaDeTexto = teclado.next();
		while( cadenaDeTexto != "FIN")
		{
			// Imprimimos por pantalla lo leído por teclado.
			System.out.println(cadenaDeTexto);
			//…leer la siguiente línea del teclado…
			cadenaDeTexto = teclado.next();
			//…e imprimirla por pantalla.
		}
			// Si queremos imprimir el FIN también...
			System.out.println(cadenaDeTexto);
	}
```

4.2. Do-while

En los bucles o ciclos do-while, que vendrían a ser un **repetir** en pseudocódigo, las instrucciones se ejecutan mientras la condición sea cierta. La condición se comprueba al final del bucle, por lo que el bloque de instrucciones se ejecutará siempre al menos una vez. Esta es la diferencia fundamental con la instrucción while. Las instrucciones de un bucle while es posible que no se ejecuten si la condición inicialmente es falsa.

La ejecución de un bucle do-while sigue los siguientes pasos:

1. Se ejecutan las instrucciones a partir de "do {".
2. Se evalúa la condición de salida.
3. Si el resultado es *false,* el programa sigue ejecutándose por la instrucción ubicada a continuación del while.
4. Pero, si el resultado es true, se vuelve al paso 1.

Ejemplo: función que devuelve el perímetro de un círculo.

```
public static double perimetroCirculo() {
	Scanner teclado = new Scanner(System.in);
	double radio;
	do {
		System.out.print("Teclee el valor del radio > 0: ");
		radio = teclado.nextDouble();
	} while (radio <= 0);
	return 2 * Math.PI * radio;
}
```

4.3. For

En los bucles o ciclos for, que vendrían a ser un **para** en pseudocódigo, una instrucción o bloque de código con instrucciones se repite un número determinado de veces para que se cumpla la condición tantas veces como le hayamos indicado.

La estructura general de una instrucción for en Java es la siguiente:

```
for(inicialización; condición; incremento/decremento){
	instrucción 1;
	// tantas instrucciones como necesitemos ...
	instrucción N;
}
```

A continuación de la palabra for, y entre paréntesis, debe haber siempre tres zonas separadas por **punto y coma**:

Zona de inicialización	Zona de condición	Zona de incremento o decremento
Es la parte en la que las variables de control del bucle toman su valor inicial. Puede haber una o más instrucciones en la inicialización, separadas por comas. La inicialización se realiza solo una vez.	Es una expresión booleana que hace que se ejecute la sentencia o el bloque de sentencias mientras sea cierta dicha expresión. Generalmente, en la condición se compara la variable de control con un valor límite.	Es una expresión que decrementa o incrementa la variable de control del bucle.

Tabla 3.1. Zonas de for

La **ejecución de un bucle for** sigue los siguientes pasos:

1. Se inicializan las variables de control (inicialización).
2. Se evalúa la condición de salida.
3. Si la condición es cierta, se ejecutan las instrucciones. Si es falsa, finaliza la ejecución del bucle y continúa el programa por la siguiente instrucción ubicada después del bucle o ciclo for.
4. Si se ha entrado en el bucle, se actualizan las variables de control (con incremento o decremento para ir acercándonos a la condición de salida).
5. Se vuelve al punto 2.

Ejemplo: función que recorre e imprime los valores enteros de una fila unidimensional.

```
public static void recorrerFilas(int numeroFilas) {
  // Recorrer todas las filas que se le indican como parámetro
  // siendo la variable indiceFila un índice para recorrer el bucle for
      for (int indiceFila = 1; indiceFila <= numeroFilas; indiceFila++) {
  //...e imprime el valor de la fila en la que se encuentra
            System.out.println( indiceFila );
      }
}
```

4.4. Foreach o for extendido

Los bucles o ciclos for each vienen a ser un «**para cada elemento**» o un «**para extendido**» en pseudocódigo. El for each facilita el recorrido de objetos existentes en una colección dinámica de objetos sin necesidad de definir el número de elementos por recorrer. La sintaxis que se emplea es la siguiente:

```
     for (TipoObjetoARecorrer nombreVariableTemporal :
nombreDeLaColecciónDeObjetos ) {
      // Pondremos tantas instrucciones como necesitemos
      Instrucciones;
     }
```

Ejemplo:

```
   String[ ] listaNombres = {"Ana", "José", "Merxe", "Fidel", "Arancha",
"Alfredo", "Lourdes", "Raül", };
   for (String nombre: listaNombres) {
      System.out.println(nombre);
   }
```

A. Bucles o ciclos infinitos

Java permite construir bucles o ciclos infinitos, los cuales se ejecutan indefinidamente, a menos que provoquemos su interrupción con la instrucción de salto «break». Estos serían posibles ejemplos de estos tipos de bucles o ciclos infinitos:

```
for ( ; ; ) {
      instrucciones;
}
for ( ;true; ) {
      instrucciones;
}
while(true) {
      instrucciones;
}

do {
      instrucciones;
} while(true)
```

Los bucles o ciclos infinitos también suelen producirse de forma involuntaria al no establecer correctamente la condición de salida de un bucle. Es un error común que solemos cometer al iniciarnos en el mundo de la programación. Tenlo en cuenta, para que no te ocurra. De lo contrario, el programa no terminará nunca de ejecutarse y se quedará colgado dentro de su propio código, pues necesita alguna manera de salir debido a una condición que le permita acceder a una instrucción break.

B. Bucles o ciclos anidados

Los bucles o ciclos anidados son aquellos que incluyen instrucciones for, while o do-while unas dentro de otras, o incluso varios bucles o ciclos anidados de una misma instrucción. Por ejemplo, para recorrer matrices se suelen utilizar dos for anidados.

Debemos tener en cuenta que las variables de control que utilicemos deben ser distintas.

Los anidamientos de estructuras tienen que ser correctos, es decir, una estructura anidada dentro de otra lo debe estar totalmente.

Ejemplo:

```
      public    static    void    recorrerFilasYColumnas int    numeroFilas,    int
numeroColumnas)
      {
      /* Recorrer todas las filas y las columnas de un tablero de ajedrez variable
      * que recibimos como parámetro tanto el número de filas como de columnas. */
            for (int indiceFila = 1; indiceFila <= numeroFilas; indiceFila++)
            {
      /* Para cada fila que recorremos con el índice indiceFila para la primera
            * dimensión, para recorrer las columnas con otro indiceColumna
            * para la segunda dimensión. */
                  for (int indiceColumna = 1; indiceColumna <= numeroColumnas;
indiceColumna++)
                  {
            //...e imprimir el valor de la posición de la Fila y la Columna
                        System.out.println("Fila: " + indiceFila + " Columna: " +
indiceColumna);
                  }
            }
      }
```