

Fundamentos de Programación

Tema 3: La programación estructurada

Contenidos

1.- Introducción	2
2.- La estructura secuencial.....	3
3.- La estructura condicional	3
4.- La estructura iterativa	6
5.- El bucle while	9
6.- El bucle for	20
7.- La estructura iterativa anidada	32
8.- El for mejorado	36

Tema

3

LA PROGRAMACIÓN ESTRUCTURADA

La programación estructurada apareció a finales de los años 60 y es un conjunto de tres reglas que sirven para controlar el flujo de ejecución de los programas. Se ha demostrado que cualquier programa puede hacerse usando solamente esas tres reglas.

1.- Introducción

Los primeros programas informáticos estaban escritos en lenguajes en los que los **números de línea** eran muy importantes, porque era posible dirigir el programa hacia uno u otro número de línea. Por ejemplo, aquí tenemos un sencillo programa escrito en el lenguaje Basic (1964):

```

10 INPUT "Cuál es su nombre:"; NN$
20 PRINT "Bienvenido al 'asterisquero' "; NN$
25 PRINT
30 INPUT "con cuántos asteriscos inicia [Cero sale]:"; N
40 IF N<=0 THEN GOTO 200
50 AS$=""
60 FOR I=1 TO N
70   AS$=AS$+"*"
80 NEXT I
90 PRINT "AQUI ESTAN:"; AS$
100 INPUT "Desea más asteriscos:"; SN$$
110 IF SN$="" THEN GOTO 100
120 IF SN$<>"S" OR N$<>"s" THEN GOTO 200
130 INPUT "CUANTAS VECES DESEA REPETIRLOS [Cero sale]:"; VECES
140 IF VECES<=0 THEN GOTO 200
150 FOR I=1 TO VECES
160   PRINT AS$;
170 NEXT I
180 PRINT
185 REM A repetir todo el ciclo (comentario)
190 GOTO 25
200 END

```

En este ejemplo podemos ver cómo la palabra **goto** servía para “redirigir” la ejecución del programa a la línea cuyo número indicamos.

Aunque este enfoque resulta sencillo para programas muy pequeños, para programas grandes es horrible, ya que se genera código imposible de mantener. Además, el hecho de que el programa “de saltos” continuamente de un lugar a otro hace que el programador se vuelva loco a la hora de encontrar los errores.

Para solucionar estos problemas, apareció la **programación estructurada**, que elimina la palabra goto e introduce tres estructuras con las que podemos realizar cualquier programa: la estructura secuencial, la estructura condicional y la estructura interactiva. Todos los lenguajes que usamos hoy (Java, C, C++, PHP, JavaScript, etc) usan estas tres estructuras.

2.- La estructura secuencial

Es una norma que, sin saberlo, ya hemos usado en nuestros primeros programas, porque es el comportamiento por defecto de la programación estructurada.

La estructura secuencial nos dice que las líneas de un programa se ejecutan una tras otra desde su comienzo hasta su finalización, de manera que solo cuando termina la ejecución de una línea, se pasa a la ejecución de la siguiente.

Recordemos uno de los primeros programas del curso:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         System.out.println("Lunes");
4.         System.out.println("Martes");
5.         System.out.println("Miércoles");
6.         System.out.println("Jueves");
7.         System.out.println("Viernes");
8.     }
9. }
```

El programa comienza en la línea 3 y se escribe en pantalla la palabra “Lunes”. Solamente cuando esta acción se ha completado, el ordenador baja de línea hacia la siguiente, donde se le ordena escribir la palabra “Martes”. Esto se repite línea a línea hasta llegar a la última, donde se escribe “Viernes” y el programa finaliza porque ya no hay más líneas.

En este ejemplo vemos como el programa ha bajado “secuencialmente” desde la línea 3 hasta la 7 de una en uno. Ahí tenemos la estructura secuencial.

1. La estructura secuencial es el comportamiento por defecto de los programas de ordenador.
2. Consiste en que las líneas del programa se ejecutan de una en una, de arriba hacia abajo, desde la primera hasta la última.

3.- La estructura condicional

En el tema 1 ya hemos estudiado la estructura condicional, que consiste en utilizar la palabra **if** para hacer que el ordenador **tome una decisión** según el valor de alguna **condición de tipo boolean**.

Si lo recordamos, vemos que había varias posibilidades:

1. Estructura condicional simple: Un solo if

```
1. int edad=new Scanner(System.in).nextInt();
2. if(edad>=18){
3.     System.out.println("Eres mayor de edad");
4. }
```

2. Estructura condicional compuesta: Un if con un else

```
1. int edad=new Scanner(System.in).nextInt();
2. if(edad>=18){
3.     System.out.println("Eres mayor de edad");
4. }else {
5.     System.out.println("Eres un menor");
6. }
```

3. Estructura condicional múltiple: Un if con varios else-if y un else alternativo

```
1. int edad=new Scanner(System.in).nextInt();
2. if(edad<18){
3.     System.out.println("Eres un menor");
4. }else if(edad<=65){
5.     System.out.println("Eres un trabajador");
6. }else if(edad<100){
7.     System.out.println("Eres un jubilado");
8. }else {
9.     System.out.println("Eres un centenario");
10. }
```

4. Estructura condicional anidada: Cualquier estructura condicional dentro de otra

```
1. int edad=new Scanner(System.in).nextInt();
2. if(edad<18){
3.     if(edad<11) {
4.         System.out.println("Vas a colegio");
5.     }else {
6.         System.out.println("Vas al instituto");
7.     }
8. }
```

Además del if, también hay otra estructura condicional llamada **bloque switch-case**, (o simplemente “bloque switch”) que cuando se puede aplicar, tiene la ventaja de que se reduce la cantidad de código que hay que escribir.

3.1.- El bloque switch-case

Es una estructura que realmente no aporta nada nuevo a los programas puesto que su efecto puede conseguirse con el if.

El bloque switch-case sirve para comprobar si el valor de una variable¹ coincide con un valor constante y ejecutar varias líneas de código en caso afirmativo.

Por ejemplo, supongamos que tenemos que hacer un programa que indique la cantidad de dinero que recibe una persona según su número de aciertos en un concurso que tiene 10 preguntas, de acuerdo a esta tabla:

Número de aciertos	7	8	9	10
Premio (€)	60	75	100	200

Para hacer este programa usando el bloque switch-case escribimos el siguiente código:

¹ En Java el bloque solo se pueden comprobar variables de tipo **int**, **long**, **byte**, **short**, **char** y **String**.

```

1. public class Programa{
2.     public static void main(String[] args){
3.         // creamos las variables necesarias
4.         int aciertos=0;
5.         int premio=0;
6.         // preguntamos al usuario su cantidad de aciertos
7.         aciertos=new Scanner(System.in).nextInt();
8.         // según el valor guardado en "aciertos", damos uno u otro premio
9.         switch(aciertos){
10.            case 7:
11.                premio=60;
12.                break;
13.            case 8:
14.                premio=75;
15.                break;
16.            case 9:
17.                premio=100;
18.                break;
19.            case 10:
20.                premio=200;
21.                break;
22.            default:
23.                premio=0;
24.        }
25.    }

```

Como vemos, dentro de la palabra **switch** se encierra entre paréntesis la variable cuyo valor se va a examinar. Solo es posible poner los tipos numéricos enteros, el char y el String.

A continuación, aparece un **case** con cada valor para el que haya que hacer algo. Dentro del case se escriben las líneas que se ejecutarán si el valor de la variable del switch coincide con el valor que acompaña la palabra case. Cada case termina con la palabra **break**, que hará que el programa salga del bloque switch-case. Esta palabra es opcional, y si no se pone (a veces se olvida por error), el programa se meterá en el siguiente case.

Por último, el bloque termina con una sección **default**, que es opcional, y en ella se indican las acciones que se realizarán si el valor de la variable no se ajusta a ningún case. La palabra default no tiene por qué ponerse al final del switch-case (aunque es lo habitual). Es posible ponerlo en cualquier lugar de la estructura, como por ejemplo, antes de los case o mezclado con ellos. Aunque en el ejemplo que hemos puesto no lleva break, también se le puede poner.

Es muy importante entender que en los case solo se pueden poner valores individuales de la variable, como por ejemplo “aciertos igual a 7”. En ningún caso es posible poner un rango o cualquier otra condición booleana, como por ejemplo, “aciertos entre 4 y 7”.

La versión del bloque switch que hemos estudiado aquí es el “switch clásico” que encontramos en lenguajes como C, C++, Java, etc. Sin embargo, en los últimos años el bloque switch se ha puesto de moda y han aparecido lenguajes donde tiene nuevas características que lo hacen más útil. El lenguaje Java reformó en **Java 13** (septiembre de 2019) su bloque switch dando lugar al “**switch mejorado**” (*enhanced switch*), que estudiaremos en otro tema.

Ejercicio 1: Realiza un programa en el que haya 5 archivos mp3 y se muestre en pantalla una lista numerada con los títulos de las 5 canciones (por ejemplo: 1 – Jackson.mp3). El usuario introducirá un número del 1 al 5 y se usará el bloque switch-case para reproducir la canción elegida por el usuario.

4.- La estructura iterativa

La estructura iterativa es la parte de la programación estructurada más difícil de dominar.

La estructura iterativa nos permite repetir un trozo de código fuente todas las veces que queramos, de forma que no tengamos que darle a “copiar y pegar”.

Vamos a poner un ejemplo donde nos vamos a encontrar en una situación donde unas líneas de nuestro programa van a tener que repetirse para poder cumplir nuestro objetivo. Supongamos que en una librería tenemos estas dos clases:

Sensor	Alarma
+ Sensor() // crea un sensor encendido + boolean detectarPersona() // devuelve true si hay alguien en ese instante + boolean comprobarOff() // devuelve true si el usuario ha apagado el sensor	+ Alarma() // crea una alarma + void emitirSonido() // emite un único sonido

Queremos hacer un programa en el que un sensor detecte continuamente si hay personas. En caso de que haya alguien, la alarma emitirá 5 sonidos.

Con lo que hemos estudiado hasta ahora, realmente no hay forma perfecta de hacer el programa. Lo más que podemos hacer es esto:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // creo la alarma y el sensor
4.         Alarma a = new Alarma();
5.         Sensor s = new Sensor();
6.         // vemos si el sensor detecta presencia humana
7.         boolean presencia = s.detectarPersona();
8.         if(presencia){
9.             // si hay alguien, la alarma emite 5 sonidos
10.            a.emitirSonido();
11.            a.emitirSonido();
12.            a.emitirSonido();
13.            a.emitirSonido();
14.            a.emitirSonido();
15.        }
16.    }
17. }
```

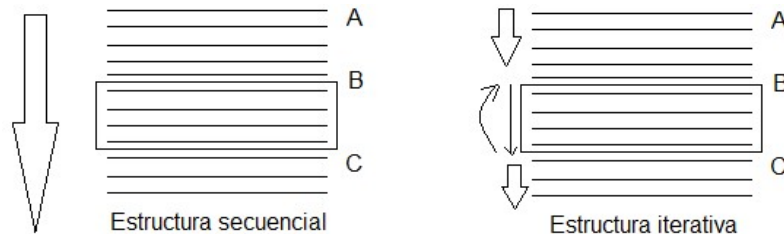
Pero este programa no es satisfactorio, porque:

5. Hemos tenido que “copiar y pegar” cinco veces la línea 10. Lo ideal es que le digamos al ordenador que tiene que repetir cinco veces esa línea.
6. El método detectarPersona solo nos detecta si hay alguien en el instante en que se llama a ese método. En otras palabras, el programa actualmente solo comprobaría si hay alguien, la alarma sonaría o no según el caso, y después finalizaría. Deberíamos ser capaces de que las líneas 7 a 15 se repitan continuamente.

Antes de estudiar cómo programar las repeticiones en informática, primero tenemos que estudiar cómo pueden ser esas “repeticiones”.

4.1.- Tipos de repeticiones

La estructura iterativa es la estructura de la programación estructurada que nos permite que una sección cualquiera del código fuente se repita todas las veces que queramos. Cuando se aplica la estructura iterativa, la sección afectada por ella se repite automáticamente, y así evitamos tener que "copiar y pegar" la misma parte muchas veces.



Con la estructura secuencial, las secciones A, B y C se ejecutan una tras otra
Con la estructura iterativa, la sección B está repitiéndose todo lo que queramos

Las dos preguntas que siempre tendremos que hacernos cuando vayamos a aplicar la estructura iterativa son:

- **¿Qué es lo que se debe repetir?**
- **¿Cuántas veces se debe producir la repetición?**

La respuesta a la primera pregunta dependerá del programa que estemos haciendo. Con la estructura iterativa podemos repetir **cualquier sección** del código fuente del programa, sin ningún tipo de restricción. Por lo tanto, podremos utilizar la estructura iterativa donde queramos, en cualquier parte del programa. En cada situación deberemos saber qué es lo que se espera que haga nuestro programa y si detectamos que debe haber un proceso que se repita continuamente, encerrarlo en una estructura iterativa.

Para responder a la segunda pregunta, distinguiremos tres tipos de repeticiones:

1. **Determinadas:** Son las situaciones en las que conocemos el número exacto de veces que debe producirse la repetición. Por "número exacto" queremos decir que hay una variable que guarda la cantidad de veces que debe repetirse algo.
2. **Indeterminadas:** Son las situaciones en las que no conocemos el número exacto de veces que debe producirse la repetición, pero sabemos que hay una condición booleana cuyo valor indica si las repeticiones deben continuar o finalizar.
3. **Determinadas con interrupción:** Son una mezcla de las dos anteriores. Son situaciones en las que conocemos un número previsto de veces que debe producirse la repetición, pero también hay una condición booleana que puede hacer las repeticiones finalicen antes de que se haya alcanzado el número que nos habíamos planteado. Es decir, sabemos que hay un número máximo de repeticiones, pero pueden pararse antes (por eso se llama con interrupción).

Ejercicio 2: Estudia estas situaciones, indicando qué es lo que se repite y el tipo de repetición.

- a) Mario debe dar 10 saltos, uno tras otro.
- b) Ponte a hacer flexiones hasta que el profesor te diga que pares.
- c) El usuario va introduciendo números por el teclado hasta que pone el 0.
- d) Escribe en la pizarra 100 veces “En clase no se habla”.
- e) Una caja tiene 50 artículos. Los vamos sacando y miramos el precio de cada uno.
- f) Un teléfono emite un sonido hasta que el dueño lo coge o la persona que llama cuelga.
- g) Tengo que resolver 20 ecuaciones de segundo grado.
- h) Tenemos que mostrar un mensaje en la pantalla tantas veces como ha introducido por teclado el usuario.
- i) Utilizo mi grabadora de CDs hasta que se rompe.
- j) Tengo que andar 15km. Al terminar un kilómetro, paro un poco y sigo hasta que llego.
- k) Voy al supermercado y empiezo a comprar cosas de forma aleatoria hasta que se me acabe el dinero.
- l) Hay que transferir por FTP los 5MB de un fichero, byte a byte.
- m) Tengo que grabar 1000 CDs con mi grabadora, mientras no se rompa.
- n) La alarma debe estar funcionando hasta que el dueño la apague.
- o) La tanda de penaltis se termina cuando uno de los equipos marca 5 goles.
- p) Debes hacer los 40 ejercicios uno tras otro hasta que éstos se acaben o el profesor te diga que pares. Además, por cada ejercicio mal, el profesor te añade 3 ejercicios más.
- q) Se presenta un menú de opciones al usuario. El usuario elige una y cuando ésta finaliza se vuelve al menú. Esto se repite hasta que el usuario selecciona la opción “Salir”.

4.2.- Los bucles

La estructura iterativa consiste en la posibilidad de repetir una sección del código fuente de un programa. Sin embargo, cada lenguaje de programación la incorpora de forma diferente según el tipo de repetición: algunos usan una palabra reservada para indicar todas las repeticiones, otros lenguajes usan palabras diferentes según el tipo de repetición, etc.

Los **bucles** son las palabras que usamos en un lenguaje de programación de alto nivel para realizar los distintos tipos de repeticiones que hemos visto. En Java vamos a estudiar los tres ² tipos de bucles más importantes:

- 4. **while:** Es un bucle con el que puede realizarse **todo tipo de repetición**. Su punto fuerte está en la realización de las **repeticiones indeterminadas**.
- 5. **do while:** Es una variante del while que funciona de manera muy parecida a él y que también sirve para hacer repeticiones indeterminadas.
- 6. **for:** Es un bucle que simplifica algunas repeticiones que aparecen muchísimo, como por ejemplo, las **repeticiones determinadas con, y sin interrupción**.

² Está demostrado que los tres bucles son equivalentes. Cualquier cosa que se pueda realizar con uno, también será posible hacerlo con los otros dos, aunque con mayor esfuerzo.

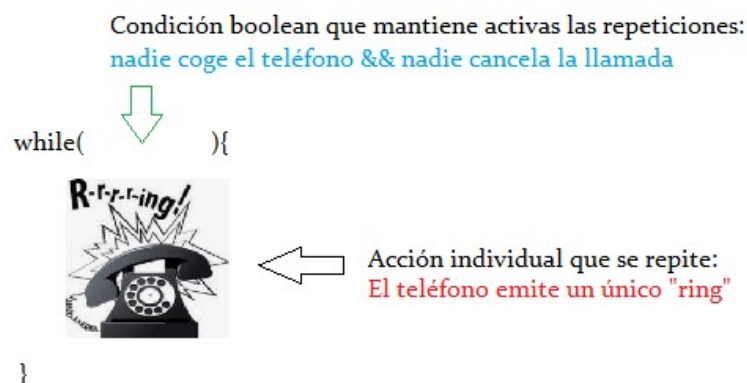
5.- El bucle while

El bucle **while**³ sirve para repetir indefinidamente una parte del código fuente, mientras que un valor **boolean** llamado **condición** se mantenga a true. Por este motivo, es ideal para hacer una **repetición indeterminada**, ya que son su ámbito natural.

En Java el bucle while se realiza escribiendo la palabra reservada **while** seguida por unos paréntesis que encierran cualquier cosa boolean (puede ser una variable boolean, una comparación, una llamada a método que devuelve boolean, etc). A continuación, se escriben entre llaves todas las líneas que se repetirán mientras el boolean que está encerrado entre paréntesis valga true:

```
1. while ( /* cualquier cosa boolean */ ) {  
2.     /* aquí vendrían todas  
3.     las líneas de código  
4.     que se repetirían mientras  
5.     la condición valga true */  
6. }
```

Por ejemplo, el caso del teléfono que suena mientras nadie lo coja o se cancele la llamada se programaría así:



Cuando el programa llega a la palabra while, se comprueba la condición boolean y si vale true (el teléfono sigue colgado y no se cancela la llamada), se ejecutarán las líneas de su interior (el teléfono emite un único sonido "ring"). A continuación, el programa vuelve a la palabra while y comprueba otra vez la condición, repitiendo el proceso. Solamente cuando la condición vale false (el teléfono se descuelga, o alguien cancela la llamada), el programa abandonará el bucle y continuará su ejecución.

Es muy importante asegurarse que en algún momento la condición booleana que hay dentro de while pueda tomar el valor "false" para que las repeticiones terminen y el programa pueda continuar. Si esto no sucede, el programa quedará bloqueado dando vueltas para siempre en un **bucle infinito**.

Vamos a ver varios ejemplos informáticos de uso del bucle while:

³ while = mientras

Ejemplo 1: Comenzaremos terminando el programa de ejemplo de la alarma, de forma que el sensor esté constantemente comprobando si hay personas en la casa, hasta que el usuario apague el sensor.

Comenzaremos analizando el enunciado: el programa debe comprobar si hay alguien y en caso afirmativo, sonará la alarma. Todo se repetirá hasta que se apague el sensor.

- ¿Hay algo que deba repetirse? Si, detectar personas, hacer que la alarma suene si hay personas y ver si el sensor se ha apagado (esto se necesita para saber si hay que terminar las repeticiones, si no se hace, el programa se repetiría para siempre).
- ¿Cuántas veces? indeterminado, porque duran hasta que se apaga el sensor.

Como las repeticiones son indeterminadas, el bucle que mejor viene para programar esto es el while. Siempre hay dos formas posibles de usar el bucle while: una “lenta pero segura”, que funciona siempre, y otra más rápida, aunque más propensa a cometer errores.

En este ejemplo vamos a utilizar la forma “lenta pero segura”, que consiste en crear una variable boolean llamada “repetir” que valdrá true mientras queramos que se repita todo. Cuando queramos parar las repeticiones, asignaremos false a esa variable.

Por tanto, tomamos el código fuente del ejemplo y creamos una variable boolean llamada “repetir” que se encargará de controlar las repeticiones. A continuación, encerramos el resto del programa en un bucle while que tenga dentro la variable “repetir”, así:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // creo la alarma y el sensor
4.         Alarma a = new Alarma();
5.         Sensor s = new Sensor();
6.         boolean repetir=true; // esta variable mantiene activas las repeticiones
7.         while(repetir){
8.             // vemos si el sensor detecta presencia humana
9.             boolean presencia = s.detectarPersona();
10.            if(presencia){
11.                // si hay alguien, la alarma emite 5 sonidos
12.                a.emitirSonido();
13.                a.emitirSonido();
14.                a.emitirSonido();
15.                a.emitirSonido();
16.                a.emitirSonido();
17.            }
18.            // ahora detectamos si la alarma debe seguir sonando
19.            if(s.comprobarOff()) {
20.                // si el sensor se ha apagado entramos en el if y paramos las repeticiones
21.                repetir = false;
22.            }
23.        }
24.    }
25. }
```

En amarillo está destacado lo más importante del ejemplo:

7. En la línea 6 creamos la variable booleana “repetir”. Mientras esta variable tenga guardado el valor true, las líneas que hay dentro del while se repetirán.
8. La línea 7 ordena al programa que repita las líneas 8 a 22 mientras la variable “repetir” tenga guardado el valor true

9. Las líneas 8 a 22 realizan la acciones que se repiten individualmente, que en este caso consiste en detectar presencia humana, emitir cinco sonidos en caso afirmativo y luego comprobar si el sensor se ha apagado
10. La línea 21 es muy importante, ya que indicamos que se paren las repeticiones si en la línea 20 se detecta que el sensor se ha apagado. En todo bucle while hecho de esta forma “lenta pero segura” habrá un momento donde tengamos que escribir “repetir=false” para no formar un bucle infinito.

Cuando este programa llega a la línea 7, se mira el valor de la variable “repetir”. Como vale true, se ejecuta el interior del bucle. Al llegar a la línea 20, pueden pasar dos cosas:

- El usuario no ha apagado el sensor → El programa se salta el if y se va a la línea 23, que es donde termina el while. Entonces, el programa vuelve otra vez a la línea 7 y como “repetir” sigue valiendo true, volverá a ejecutarse el interior del bucle.
- El usuario ha apagado el sensor → El programa entra en la línea 21 y la variable “repetir” se pone a false. Al salir del if el programa llega a la línea 23, que es donde termina el while. De ahí, vuelve a la línea 7, pero como ahora “repetir” vale false, entonces ya no se ejecuta el bloque interior y de ahí se pasa a la línea 24, en la que termina el programa.

Por tanto, escribiendo **while(repetir)** conseguimos repetir de forma indeterminada las acciones del interior del bucle. Cuando queremos parar las repeticiones, lo que hacemos es asignar false a la variable repetir. Como hemos dicho, esta forma “lenta pero segura” de hacer el bucle while servirá siempre.

Ejemplo 2: Vamos a hacer de nuevo el ejemplo anterior, pero de otra forma. El bucle while que hemos visto antes “lento pero seguro” y **siempre funcionará**⁴. Sin embargo, en la práctica es posible escribir bastantes menos líneas si dentro del while escribimos directamente la condición boolean que mantiene activas las repeticiones. En este caso, esa condición es comprobar que el sensor no haya sido apagado.

Por tanto, podemos reescribir el ejemplo anterior eliminando la variable “repetir” y comprobando directamente en la condición del while que el sensor no ha sido pulsado (esa sería la condición que, de ser true, mantiene activas las repeticiones).

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // creo la alarma y el sensor
4.         Alarma a = new Alarma();
5.         Sensor s = new Sensor();
6.         // lo que hay dentro del while se repite mientras s.comprobarOff() devuelva false
7.         while(s.comprobarOff()==false){ // también lo podemos escribir así: !s.comprobarOff()
8.             // vemos si el sensor detecta presencia humana
9.             boolean presencia = s.detectarPersona();
10.            if(presencia){
11.                // si hay alguien, la alarma emite 5 sonidos
12.                a.emitirSonido();
13.                a.emitirSonido();
```

⁴ Algunas situaciones de problemas reales solo pueden resolverse usando la forma “lenta pero segura”

```

14.         a.emitirSonido();
15.         a.emitirSonido();
16.         a.emitirSonido();
17.     }
18. }
19. }
20. }

```

Este programa sigue haciendo lo mismo de antes. Se ejecutan las líneas 8 a 17 y al llegar a la línea 17, se vuelve a la línea 7, en la que se comprueba directamente si el sensor se ha apagado. En caso afirmativo, el programa abandona el bucle y termina. En caso negativo, vuelve a repetirse el bucle.

Como vemos, el programa así es mucho más corto y eficiente, aunque a cambio, escribir la condición es un poco más difícil que antes y se introduce la posibilidad de equivocarnos (antes siempre se escribía `while(repetir)`, con lo que nunca nos equivocaríamos).

Normalmente la condición `s.comprobarOff()==false` la escribiremos `!s.comprobarOff()`

Ejemplo 3: Supongamos que queremos hacer un programa en el que el usuario escriba números de forma indefinida hasta que introduzca el cero.

Como siempre, analizamos el programa antes de empezar para ver sus repeticiones:

- ¿Qué se repite? Pedir un número al usuario y comprobar si es un cero (así se detecta el momento de parar las repeticiones. Siempre debe haber un momento en el que detectemos que hay que parar las repeticiones).
- ¿Cuántas veces? Indeterminado, porque es hasta que el usuario introduzca un cero, y eso no podemos saber cuándo será.

Como en este ejemplo también tenemos repeticiones indeterminadas, lo realizaremos con el bucle `while`. Si nos decidimos por hacerlo de la forma “lenta pero segura”, crearemos una variable boolean `repetir` a la que daremos el valor `true` y que usaremos como condición del `while`. Dentro del bucle, detectaremos el momento en que el usuario introduzca un 0 para asignar `false` a la variable `repetir`.

El código sería así:

```

1.  public class Programa{
2.      public static void main(String[] args){
3.          // creamos la variable que controla las repeticiones (forma "lenta pero segura")
4.          boolean repetir=true;
5.          while(repetir){
6.              // pedimos un número al usuario
7.              int n=new Scanner(System.in).nextInt();
8.              // si el número es 0, paramos las repeticiones
9.              // y en caso contrario, lo mostramos en pantalla
10.             if(n==0){
11.                 repetir=false;
12.             }else{
13.                 System.out.println("Has escrito el número "+n);
14.             }
15.         }
16.     }
17. }
18. }

```

Esta forma "lenta pero segura" funciona siempre pero podemos hacerlo con meno líneas comprobando en el while si el usuario ha escrito un 0. Para eso, deberemos crear antes del while la variable con el número que va a leer el usuario, ya que necesitamos tener esa variable disponible en la línea 6. Además, debemos poner a esa variable un **valor inicial que no sea 0**, porque si no, **el programa se saltaría el bloque while justo al llegar a él**.

```
1. public class Programa{
2.     public static void main(String[] args){
3.         /* Creamos una variable para guardar el número que introduzca el usuario.
4.         Como esa variable sale en la condición del while,
5.         le tenemos que dar un valor que nos asegure que el programa entre en el while */
6.         int n=1;
7.         // repetimos, mientras que el número sea distinto de 0
8.         while(n!=0){
9.             // actualizamos la variable n con el número que introduzca el usuario
10.            n=new Scanner(System.in).nextInt();
11.            // Mostramos el número en pantalla
12.            System.out.println("Has escrito el número "+n);
13.        }
14.    }
15. }
```

Como vemos, el bucle while se ha reducido bastante, aunque para eso hemos tenido que escribir una condición algo más compleja, y además, hemos tenido que tener la precaución de dar un valor inicial a "n" apropiado para que el programa pueda entrar en el while la primera vez que llega a él.

Por tanto, las conclusiones que podemos extraer a la hora de hacer un bucle while son:

16. La forma "lenta pero segura" siempre nos funcionará, aunque escribiremos más.

- **El bucle while siempre se escribe igual: while(repetir) {**
- **En algún momento del interior del while tendremos que poner: repetir=false;**

17. La forma "rápida" es más eficiente, pero tenemos que tener cuidado con:

- **Escribir correctamente la condición**
- **Dar a las variables que aparezcan en la condición un valor inicial que nos asegure que el programa entrará en el while la primera vez.**

Para evitar el problema de asignar a las variables un valor compatible para entrar en el while, es posible usar una variante del bucle while llamada **bucle do-while**, que consiste en comprobar la condición **al final** del bucle, en lugar de al principio.

Nuestro programa de ejemplo se haría así:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // Creamos una variable para guardar el número que introduzca el usuario
4.         int n=0;
5.         // repetimos las líneas que hay dentro del do
6.         do{
7.             // actualizamos la variable n con el número que introduzca el usuario
8.             n=new Scanner(System.in).nextInt();
9.             // Mostramos el número en pantalla
10.            System.out.println("Has escrito el número "+n);
11.        }while(n!=0); // en caso de que el número sea distinto de 0, continuamos repitiendo
12.    }
13. }
```

De esta forma, al comprobar la condición de repetición al final del bucle, no tenemos que preocuparnos del valor inicial que le demos a la variable.

Sin embargo, el bucle do-while tiene una pequeña diferencia con el while, y es que las líneas del interior del bucle se ejecutan al menos una vez. Observa que en el while puede ocurrir que las líneas del interior no lleguen a ejecutarse nunca, debido a que la condición para iniciar las repeticiones se comprueba al principio.

Esto hace que el bucle while sea más seguro, porque su condición está actuando como una protección ante líneas que tal vez no deban comenzar a repetirse. Con el do-while nos lanzamos siempre a ejecutar al menos una vez las líneas interiores, lo que, si no hacemos comprobaciones, en algunos casos podría dar lugar a errores.

En general, el uso de **while** está más extendido que el do-while, aunque ambos son equivalentes.

Ejemplo 4: Vamos a hacer ahora un programa que pida números por teclado al usuario hasta que éste introduzca un 0. En ese momento el programa mostrará la suma de dichos números.

En este ejemplo aparece una de las principales dificultades de la programación, y es la necesidad de tener que crear una variable auxiliar que no aparece explícitamente mencionada en el enunciado del programa.

Si nos fijamos en el enunciado, el programa es muy similar al del ejercicio anterior, pero al terminar tenemos que mostrar en pantalla la suma de los números. ¿Cómo podemos hacer esto si tras cada repetición el número introducido “se pierde”? ¿Cómo es posible poder sumar todos los números?

La solución consiste en ir construyendo la suma poco a poco, conforme se van introduciendo números y no querer calcularla al final del programa (esto ahora mismo sería imposible, porque cada número introducido por el usuario se pierde tras cada iteración).

Por tanto, crearemos una variable llamada “suma” que inicialmente valdrá 0, y cada vez que el usuario escribe un número, incrementaremos la variable “suma”. De esa forma, la variable “suma” tendrá la suma de todos los números introducidos hasta el momento. Al final del programa, bastará con mostrar por pantalla su valor.

El análisis del programa sería:

- a) ¿Qué se repite? Pedir un número al usuario y añadir dicho número a la variable "suma"
- b) ¿Cuántas veces? Indeterminado, porque se finaliza cuando el usuario introduce un 0.

De la forma “lenta pero segura” lo escribiríamos así:

```

1. public class Programa{
2.     public static void main(String[] args){
3.         // Creamos una variable para guardar la suma de los números que se introduzcan
4.         int suma=0;
5.         // hacemos repeticiones indeterminadas de la forma "lenta pero segura"
6.         boolean repetir=true;
7.         while(repetir){
8.             // pedimos un número al usuario
9.             int n=new Scanner(System.in).nextInt();
10.            // Si el número es distinto de 0 añadimos a la variable suma ese número
11.            // y si no, paramos las repeticiones
12.            if(n!=0){
13.                suma += n;
14.            }else {
15.                repetir=false;
16.            }
17.        }
18.        System.out.println("La suma de todos los números introducidos es "+suma);
19.    }
20. }

```

Pero quedaría mucho mejor si lo escribimos de la forma “rápida”, aunque para eso tenemos que crear antes del bucle la variable que guarde el número que introduzca el usuario, y además le tenemos que dar un valor inicial distinto de 0 para que pueda entrar en el while (otra opción sería usar un bucle do-while)

```

1. public class Programa{
2.     public static void main(String[] args){
3.         // Creamos una variable para guardar la suma de los números que se introduzcan
4.         int suma=0;
5.         // Creamos la variable para guardar el número que introduce el usuario
6.         int n=1;
7.         while(n!=0){
8.             // pedimos un número al usuario
9.             n=new Scanner(System.in).nextInt();
10.            // incrementamos la variable "suma" con ese número
11.            suma += n;
12.        }
13.        System.out.println("La suma de todos los números introducidos es "+suma);
14.    }
15. }

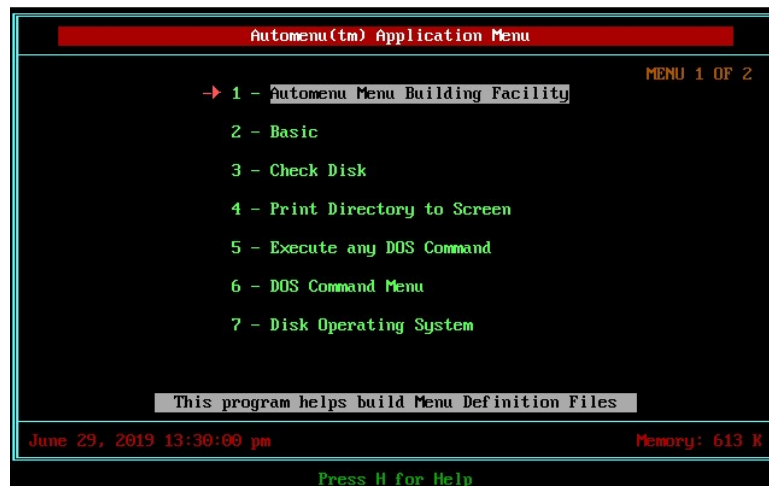
```

Ejemplo 5: Supongamos que estamos haciendo un programa para un concesionario y se muestra en pantalla un menú con estas opciones:

1. Comprar coche
2. Ver lista de coches vendidos
3. Salir

Queremos hacer que el programa esté continuamente funcionando hasta que el usuario pulse la opción "salir".

Este tipo de situación fue muy típica antiguamente en las aplicaciones de MS-DOS, antes de la aparición de Windows, y hoy la podemos encontrar en las aplicaciones de terminal para Linux.



Como venimos haciendo en todos los ejemplos, analizamos primero el programa para ver qué es lo que repite y su tipo de repetición:

- ¿Qué se repite? Mostrar el menú al usuario, leer una tecla con la opción deseada y según la tecla pulsada realizar la acción correspondiente. Si se pulsa la opción "salir", las repeticiones finalizan.
- ¿Cuántas veces se repite? Indeterminado, porque no sabemos cuando el usuario pulsará la opción "salir".

Para hacer el programa vamos a crear una variable que almacene la opción seleccionada por el usuario. En caso de que esa variable tome el valor 1 o 2, realizaremos las acciones necesarias para vender un coche o mostrar la lista de coches vendidos, pero si la variable vale 3, pararemos las repeticiones. Es clásico usar el bloque switch-case para detectar el valor de la variable y programar cada una de las opciones.

Comenzamos viendo como sería la forma "lenta pero segura" de resolverlo:

```

1. public class Programa{
2.     public static void main(String[] args){
3.         // hacemos un while "lento pero seguro"
4.         boolean repetir=true;
5.         while(repetir){
6.             // muestro el menú
7.             System.out.println("Seleccione una opción:");
8.             System.out.println("1.- Comprar coche");
9.             System.out.println("2.- Ver lista de coches vendidos");
10.            System.out.println("3.- Salir");
11.            // pedimos la opción al usuario
12.            int opcion=new Scanner(System.in).nextInt();
13.            // comprobamos qué ha introducido y actuamos en consecuencia
14.            switch(opcion){
15.                case 1:
16.                    // aquí se escribiría código para vender un coche
17.                    break;
18.                case 2:
19.                    // aquí se escribiría código para ver la lista de coches vendidos
20.                    break;
21.                case 3:
22.                    // paramos las repeticiones
23.                    repetir=true;
24.            }
25.        }
26.    }
27. }

```


Y como ya sabemos, también lo podemos hacer con la forma “rápida”:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // creamos la variable para guardar la elección del usuario
4.         int opcion=0;
5.         // repetimos mientras que el usuario no pulse la tecla 3
6.         while(opcion!=3){
7.             // muestro el menú
8.             System.out.println("Seleccione una opción:");
9.             System.out.println("1.- Comprar coche");
10.            System.out.println("2.- Ver lista de coches vendidos");
11.            System.out.println("3.- Salir");
12.            // pedimos la opción al usuario
13.            int opcion=new Scanner(System.in).nextInt();
14.            // comprobamos qué ha introducido y actuamos en consecuencia
15.            switch(opcion){
16.                case 1:
17.                    // aquí se escribiría código para vender un coche
18.                    break;
19.                case 2:
20.                    // aquí se escribiría código para ver la lista de coches vendidos
21.                    break;
22.            }
23.        }
24.    }
25. }
```

En este caso, el valor que le podemos dar a la variable “opción” es cualquier número entero menos el 3, ya que sería el único caso en que el while no entraría por primera vez. Por supuesto, también se podría hacer con un do-while.

Ejemplo 6: Vamos a hacer un programa que muestre por pantalla 50 veces la frase "En clase no se juega".

- ¿Qué se repite? Mostrar por pantalla la frase "En clase no se juega"
- ¿Cuántas veces? Exactamente 50, por tanto, son **repeticiones determinadas**.

Como ya dijimos, hacer una repetición determinada con el bucle while no es lo mejor porque es más fácil hacerlo con el bucle "for", como veremos más adelante. De todas formas, vamos a ver cómo se resolvería el ejercicio con un while.

Para hacer el programa tenemos que crear una variable que nos cuente las veces que llevamos escrita la frase, y que cuando dicho número sea 50, se terminen las repeticiones. Esta variable se suele llamar **contador**, porque va contando las repeticiones que se van haciendo.

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // hacemos una variable que cuente el número de repeticiones que llevamos
4.         int numeroRepeticiones=0;
5.         // hacemos un while "lento pero seguro"
6.         boolean repetir=true;
7.         while(true){
8.             System.out.println("En clase no se juega");
9.             // incrementamos en una unidad la variable "numeroRepeticiones"
10.            numeroRepeticiones++;
11.            // si llegamos a 50 repeticiones, paramos
12.            if(numeroRepeticiones==50){
13.                repetir=false;
14.            }
15.        }
16.    }
17. }
```

Como siempre, podemos hacerlo más rápido de esta forma:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // hacemos una variable que cuente el número de repeticiones que llevamos
4.         int numeroRepeticiones=0;
5.         // repetimos mientras llevemos menos de 50 repeticiones
6.         while(numeroRepeticiones<50){
7.             System.out.println("En clase no se juega");
8.             // incrementamos en una unidad la variable "numeroRepeticiones"
9.             numeroRepeticiones++;
10.        }
11.    }
12. }
```

Ejercicio 3: Realiza un programa que genere números enteros aleatorios entre 0 y 10 y los muestre por pantalla. El programa terminará cuando el número generado sea el 10.

Ejercicio 4: Realiza un programa que pregunte al usuario "¿Eres mayor de edad (si/no)?". Si el usuario escribe "Si", el ordenador mostrará un mensaje "El usuario es mayor de edad". Si el usuario escribe "No", el ordenador mostrará "El usuario es menor de edad". Si el usuario escribe cualquier otra cosa, el ordenador mostrará "No te entiendo" y volverá a hacer la pregunta hasta que el usuario escriba lo correcto.

Ejercicio 5: Según el reglamento de la federación de caza, un cazador no puede cazar más piezas de las que se permiten en un día. Queremos hacer un programa que nos lleve la cuenta de piezas cazadas e indique cuando se ha excedido el límite. Para ello primero se leerá por teclado el límite del día y a continuación los valores de las piezas cazadas en el orden que se obtienen. El programa imprimirá un mensaje en el momento en que el límite haya sido excedido. Después de que cada pieza ha sido registrada, el programa mostrará el número total de piezas que se llevan hasta ese momento cazadas. Ejemplo:

```
¿Cuántas piezas se pueden cazar hoy? 30
--- Empezamos la caza
Introduzca el número de piezas cazadas
10
Usted lleva cazadas 10 piezas de 30 posibles
Introduzca el número de piezas cazadas
5
Usted lleva cazadas 15 piezas de 30 posibles
Introduzca el número de piezas cazadas
20
El número máximo de piezas ha sido excedido
Pulse una tecla para finalizar
```

Ejercicio 6: Usa la Consola DAW para hacer un programa que pregunte al usuario por la ruta de una imagen y la ponga como imagen de fondo. El programa entonces preguntará "¿Desea poner otra imagen?". Si el usuario pulsa la tecla S, se repetirá el proceso, y si pulsa cualquier otra tecla, finalizará.

Ejercicio 7: Usa la Consola DAW para realizar un programa que pregunte al usuario "¿Cuántos círculos desea dibujar?". El usuario escribirá un número entero (si el número es incorrecto o negativo, el programa dirá "Número incorrecto" y finalizará sin hacer nada más) y la pantalla se borrará y se dibujarán tantos círculos como haya indicado el usuario. Cada círculo tendrá un color con valores RGB aleatorios entre 0 y 255, y sus coordenadas también serán aleatorias entre (0,0) y la resolución de la pantalla.

Ejercicio 8: Usa la clase **StringTokenizer** para hacer un programa que pregunte al usuario "Escriba una frase" y nos muestre en una línea diferente cada palabra de la frase y además, el número total de palabras encontradas.

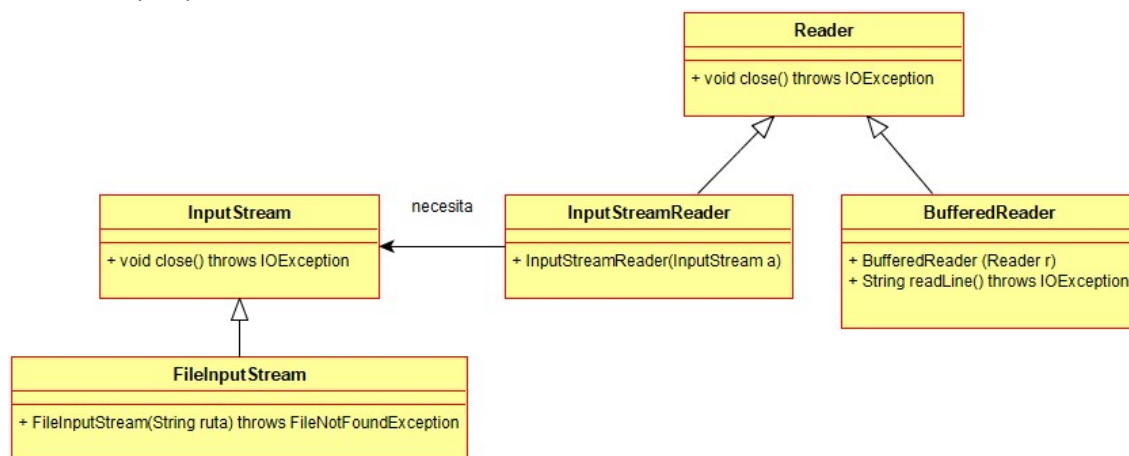
Ejercicio 9: Realiza un programa que calcule de forma aproximada la raíz cuadrada de un número positivo N introducido desde el teclado usando el *Método de Newton-Raphson*, que consiste en lo siguiente:

- Creamos dos variables de tipo double llamadas S y A, y les damos valor inicial uno.
- Actualizamos S con la siguiente fórmula:

$$S = \frac{1}{2} \left(A + \frac{N}{A} \right)$$

- Si el valor absoluto de la diferencia entre S y A es menor de 0.000001 entonces ya hemos terminado y la raíz cuadrada aproximada es S. En caso contrario, le damos a A el valor de S y repetimos el proceso a partir del punto anterior.

Ejercicio 10: Utiliza las clases y métodos de la librería de entrada y salida que se indican a continuación para hacer un programa que pregunte por teclado la ruta de un archivo de texto (.txt) y a continuación, use un **BufferedReader** para leer una por una todas sus líneas y mostrarlas por pantalla a la vez.



Ejercicio 11: Utiliza las clases **PrintWriter** y **BufferedReader** de la librería de entrada y salida para realizar un programa que muestre al usuario este menú de opciones que se repite continuamente hasta que se pulsa la opción de salir:

1. Añadir nuevo alumno
2. Ver los datos de un alumno
3. Salir

Al pulsar cada opción, sucede esto:

- Opción 1: El programa preguntará el dni, nombre, dirección y teléfono de un alumno y guardará estos tres últimos datos en líneas diferentes en un archivo de texto llamado **(poner el dni).txt**
- Opción 2: El programa pregunta un dni y recupera del archivo **(poner el dni).txt** el nombre, dirección y teléfono del alumno, mostrándolos por pantalla.
- Opción 3: El programa finalizará.

Ejercicio 12: Realiza un programa que pida una por una, las notas de los alumnos comprendidas entre 0 y 10 sin decimales. Cuando el usuario introduzca un -1, se mostrará la mayor nota, la menor, y la nota media de todos ellos.

Ejercicio 13: Usa las clases **LocalDate** y **DateTimeFormatter** para hacer un programa que pregunte al usuario un número de mes (entre 1 y 12) y un año. El programa mostrará por pantalla todos los días de ese mes, con el siguiente formato: *“día/mes/año y día de la semana”*

```
Número del mes
11
Número del año
2018
01/11/2018 jueves
02/11/2018 viernes
03/11/2018 sábado
```

6.- El bucle for

Si recordamos el ejemplo del programa de la alarma y el sensor, en la última versión que hicimos se nos quedó esto:

```
13. public class Programa{
14.     public static void main(String[] args){
15.         // creo la alarma y el sensor
16.         Alarma a = new Alarma();
17.         Sensor s = new Sensor();
18.         boolean repetir=true; // esta variable mantiene activas las repeticiones
19.         while(repetir){
20.             // vemos si el sensor detecta presencia humana
21.             boolean presencia = s.detectarPersona();
22.             if(presencia){
23.                 // si hay alguien, la alarma emite 5 sonidos
24.                 a.emitirSonido();
25.                 a.emitirSonido();
26.                 a.emitirSonido();
27.                 a.emitirSonido();
28.                 a.emitirSonido();
29.             }
30.             // ahora detectamos si la alarma debe seguir sonando
31.             if(s.comprobarOff()) {
32.                 // si el sensor se ha apagado entramos en el if y paramos las repeticiones
33.                 repetir = false;
34.             }
35.         }
36.     }
37. }
```

Aquí podemos ver que estamos repitiendo una línea de código cinco veces, lo cual nos lleva a un caso de repeticiones determinadas. Como ya sabemos, “copiar y pegar” es una mala práctica de programación que debemos evitar siempre, así que debemos modificar el

programa para que el ordenador repita la acción individual a.emitirSonido() exactamente cinco veces.

Usando la técnica del contador que vimos en el ejemplo 6 del apartado while, podríamos escribir un bucle while para automatizar las repeticiones, así:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         // creo la alarma y el sensor
4.         Alarma a = new Alarma();
5.         Sensor s = new Sensor();
6.         boolean repetir=true; // esta variable mantiene activas las repeticiones
7.         while(repetir){
8.             // vemos si el sensor detecta presencia humana
9.             boolean presencia = s.detectarPersona();
10.            if(presencia){
11.                // creo una variable que cuente las repeticiones
12.                int contador=0;
13.                // repetimos mientras el contador no llegue a 5 (sea 0,1,2,3,4)
14.                while(contador<5){
15.                    a.emitirSonido();
16.                    contador++;
17.                }
18.            }
19.            // ahora detectamos si la alarma debe seguir sonando
20.            if(s.comprobarOff()) {
21.                // si el sensor se ha apagado entramos en el if y paramos las repeticiones
22.                repetir = false;
23.            }
24.        }
25.    }
26. }
```

Si observamos el código, hay tres líneas marcadas con colores:

- Crear la variable contador
- Comprobar que el contador no ha llegado al tope de las repeticiones determinadas
- Incrementar el contador

El bucle **for** es un bucle diseñado para escribir en una sola línea esas tres cosas, por lo que se acorta mucho la línea para escribir las repeticiones determinadas.

Con el bucle for, la parte de las repeticiones se queda así:

```
int contador=0;
while(contador<5){
    a.emitirSonido();
    contador++;
}

➡

for(int contador=0; contador<5; contador++){
    a.emitirSonido();
}
```

Por tanto, el for en realidad no es más que una forma especial de escribir un bucle while donde la creación del contador, la condición que controla las repeticiones y el incremento del contador están en una sola línea.

Lo más habitual será llamar al contador i, por lo que el código del ejemplo de la alarma quedará finalmente así:

```

1. public class Programa{
2.     public static void main(String[] args){
3.         // creo la alarma y el sensor
4.         Alarma a = new Alarma();
5.         Sensor s = new Sensor();
6.         boolean repetir=true; // esta variable mantiene activas las repeticiones
7.         while(repetir){
8.             // vemos si el sensor detecta presencia humana
9.             boolean presencia = s.detectarPersona();
10.            if(presencia){
11.                // repetimos 5 veces la emisión del sonido (i toma valores 0,1,2,3,4)
12.                for(int i=0;i<5;i++){
13.                    a.emitirSonido();
14.                }
15.            }
16.            // ahora detectamos si la alarma debe seguir sonando
17.            if(s.comprobarOff()) {
18.                // si el sensor se ha apagado entramos en el if y paramos las repeticiones
19.                repetir = false;
20.            }
21.        }
22.    }
23. }

```

El funcionamiento es como si fuese un while. Cuando el programa llega al for, ejecuta la parte amarilla, que creará una variable “i” que solo vive dentro del for. A continuación, comprueba la condición azul, y si se cumple ejecuta las líneas de código del interior del bloque. Cuando se terminan, se ejecuta la parte de color verde (el incremento del contador) y el programa vuelve a la línea 12, donde se comprueba nuevamente la condición azul. El proceso se repite hasta que dicha condición deja de cumplirse.

Vamos a ver más ejemplos de utilización del bucle for:

Ejemplo 7: Vamos a repetir el ejemplo 6, pero usando el bucle for. El código que hicimos en aquel momento fue este:

```

38.         // hacemos una variable que cuente el número de repeticiones que llevamos
39.         int numeroRepeticiones=0;
40.         // repetimos mientras llevemos menos de 50 repeticiones
41.         while(numeroRepeticiones<50){
42.             System.out.println("En clase no se juega");
43.             // incrementamos en una unidad la variable "numeroRepeticiones"
44.             numeroRepeticiones++;
45.         }

```

Ahora simplemente lo reescribimos usando el bucle for, que como ya hemos visto, es más apropiado para las repeticiones determinadas.

```

1.         for(int i=0;i<50;i++){
2.             System.out.println("En clase no se juega");
3.         }

```

Como podemos ver, el resultado es mucho más corto e intuitivo.

Ejemplo 8: Vamos a repetir el ejemplo anterior, pero numerando cada una de las frases que se muestran en pantalla.

Para añadir la numeración, lo único que tenemos que tener en cuenta es que dentro del for podemos hacer uso de la variable “i” que nos lleva la cuenta del número de repeticiones. Sería así:

```

1.     for(int i=0;i<50;i++){
2.         System.out.println(i+"-En clase no se juega");
3.     }

```

Si lo ejecutamos, veremos que al lado del mensaje se nos muestra el número de repetición, y que éstas empiezan por 0 y terminan en 49. Eso es así porque la variable "i" empieza en 0 y la condición "i<50" lo que nos dice es que las repeticiones duran mientras i sea menor (pero no igual) que 50. O sea, para i=0,1,2,...,49 se muestra el mensaje en la pantalla, pero la última vez (cuando i=50) la condición no se cumple y ya se sale del for.

Por último, si queremos que la numeración esté entre 1 y 50, tenemos dos posibilidades:

1. Sumamos uno a la variable contadora, cuando vayamos a mostrarla por pantalla:

```

a)     for(int i=0;i<50;i++){
b)         System.out.println((i+1)+"-En clase no se juega");
c)     }

```

2. Hacemos que la variable contadora comience en 1 y termine en 50

```

1.     for(int i=1;i<=50;i++){
2.         System.out.println(i+"-En clase no se juega");
3.     }

```

En ambos casos ahora la numeración comienza en 1 y termina en 50. No hay forma mejor que otra. Lo importante es resolver el problema con las herramientas que tengamos a nuestra disposición.

Ejemplo 9: Queremos modificar el programa del ejemplo anterior, de manera que cuando el programa vaya por la mitad, el programa muestre un mensaje de aviso por pantalla.

Para hacer esto nuevamente hay que utilizar la variable contadora "i", de manera que cuando llegue a la mitad de las repeticiones (25 si numeramos de 1 a 50), se muestre un mensaje por pantalla. Sería así:

```

1.     public class Programa{
2.         public static void main(String[] args){
3.             for(int i=1;i<=50;i++){
4.                 if(i==25){
5.                     System.out.println(i+"En clase no se juega");
6.                 }
7.             }
8.         }
9.     }

```

Ejercicio 14: Haz un programa que dibuje en la pantalla 8 rectángulos cuyas coordenadas, dimensiones y color sean aleatorios.

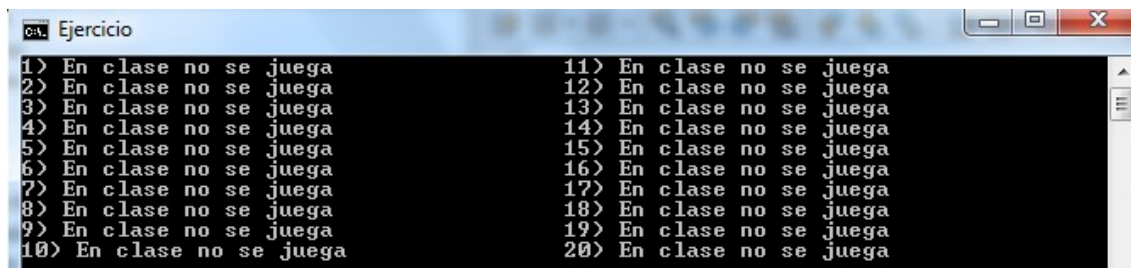
Ejercicio 15: Usa la Consola DAW para realizar un programa que muestre 20 veces por pantalla la frase "En clase no se juega" cada una en renglón diferente. Las veces impares se alinearán a la izquierda y las pares a la derecha.

```

1> En clase no se juega      2> En clase no se juega
3> En clase no se juega      4> En clase no se juega
5> En clase no se juega      6> En clase no se juega
7> En clase no se juega      8> En clase no se juega
9> En clase no se juega      10> En clase no se juega
11> En clase no se juega     12> En clase no se juega
13> En clase no se juega     14> En clase no se juega
15> En clase no se juega     16> En clase no se juega
17> En clase no se juega     18> En clase no se juega
19> En clase no se juega     20> En clase no se juega

```

Ejercicio 16: Usa la Consola DAW para realizar un programa que muestre 20 veces por pantalla la frase "En clase no se juega" de forma que las primeras 10 veces aparezcan pegadas a la izquierda y las siguientes 10 veces aparezcan pegadas a la derecha.



```

Ejercicio
1> En clase no se juega      11> En clase no se juega
2> En clase no se juega      12> En clase no se juega
3> En clase no se juega      13> En clase no se juega
4> En clase no se juega      14> En clase no se juega
5> En clase no se juega      15> En clase no se juega
6> En clase no se juega      16> En clase no se juega
7> En clase no se juega      17> En clase no se juega
8> En clase no se juega      18> En clase no se juega
9> En clase no se juega      19> En clase no se juega
10> En clase no se juega     20> En clase no se juega

```

Ejercicio 17: Realiza un programa que pregunte por teclado una dirección IP y un número entero. El programa usará la clase **InetAddress** para realizar la cantidad de pings indicada a la dirección IP, midiendo el tiempo empleado en hacer cada uno (clases **Instant** y **Duration**), y mostrará el resultado de esta forma:

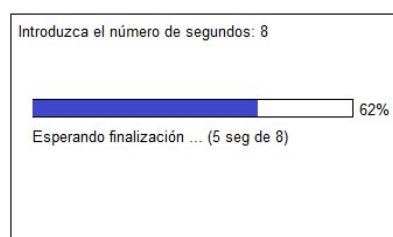
```

Haciendo ping a www.google.es [216.58.201.163]
Respuesta desde 216.58.201.163: tiempo=15ms
Respuesta desde 216.58.201.163: tiempo=17ms

```

Ejercicio 18: Usa la Consola DAW y las clases de Java Time para realizar un programa que muestre en el centro de la pantalla la hora, minutos y segundos actuales. Durante medio minuto, a cada segundo la pantalla se borrará y se volverá a imprimir la hora actual, dando la apariencia de un reloj.

Ejercicio 19: Usa la Consola DAW para hacer un programa que pregunte al usuario un número de segundos. El programa finalizará cuando pase la cantidad de segundos introducida y durante ese tiempo se dibujará y actualizar en la pantalla una barra de progreso como esta:



Sugerencia: Espera de uno en uno los segundos introducidos por el usuario. Al finalizar cada segundo dibuja la barra de progreso con los valores actualizados. Calcula con una regla de tres la medida en píxeles de la parte completada de la barra.

Ejercicio 20: Realiza un programa que calcule la suma de todos los números de esta lista:

2,9,4,5,6,1,2,3,45,2,1,2,65,3,2,6,1,2,3,5,2,2,6,67,11,67,3,2,1,7,8,5,3,27,8,93,1,6

Ejercicio 21: Realiza un programa que cree la lista de números del ejercicio anterior. A continuación, el programa preguntará al usuario: “Escribe un número” y nos mostrará en pantalla cuántas veces aparece el número en la lista y si el número está o no en la lista.

Ejercicio 22: Realiza un programa que muestre el nombre de cada alumno y su nota media

Alumno/a	Programación	Base de datos	Sistemas	Nota FOL
Pepe	9	5	4	3
Ana	4	8	6	5
Juan	2	5	10	2
Guillermo	0	0	5	0
Roberta	10	10	10	9

Ejercicio 23: Consulta la documentación del “**Java Collection Framework**” y busca la interfaz **List<T>**, que representa una lista dinámica (es decir, la lista empieza vacía y se le pueden añadir y quitar objetos). Una vez que la tengas y la hayas leído, haz el siguiente programa:

- El programa comenzará creando un objeto **List<String>** y lo rellenará con un objeto de la clase que más se usa en la práctica para implementar **List<T>**.
- El programa mostrará este menú de opciones, que se repetirá hasta que el usuario pulse la opción de salir.
 - Añadir mensaje a la lista
 - Consultar el número de mensajes de la lista
 - Consultar mensaje
 - Comprobar mensaje
 - Consultar todos los mensajes
 - Borrar toda la lista
 - Salir
- Lo que hay que hacer en cada opción es:
 - Pide al usuario que introduzca una frase y la añadirá al **List<String>**
 - Muestra en pantalla el número de elementos que hay en el **List<String>**
 - Cuando se selecciona, pueden pasar dos cosas
 - Si la lista está vacía, se mostrará “No hay mensajes” y volverá al menú.
 - Si la lista no está vacía, el programa preguntará el número del mensaje que se desea consultar y el usuario deberá introducir un número entre

0 y la última posición válida de la lista. Si se introduce un número fuera de ese rango, el programa avisará de ello. En caso contrario, se mostrará el mensaje indicado.

4. Pedirá al usuario que escriba un mensaje y el programa mostrará si el mensaje está o no en la lista. En caso de que si esté, se mostrará también su número de posición.
5. Mostrará todos los mensajes de la lista, cada uno en una línea con el formato “El mensaje de la posición ... es ...”
6. Borra todos los mensajes de la lista

Ejercicio 24 : Haz un programa que pregunte por teclado el número de un mes (entre 1 y 12) y haga lo siguiente:

- Si el número no está entre 1 y 12, se mostrará un mensaje de error y finalizará.
- El programa creará un `List<LocalDate>` y lo rellenará con objetos `LocalDate` para todos los días del mes cuyo número se ha pasado como parámetro.
- Se mostrará por pantalla cada elemento de la lista y también el tamaño de la lista.

Ejercicio 25 : Consulta el pdf de arrays y haz un programa en el que haya un array rellenado con 5000 ceros. A continuación el programa modificará las posiciones impares y guardará un 1 en ellas. Al terminar, se mostrará el array en pantalla indicando el número que hay en cada posición.

Muy importante: No es lo mismo un array que un `ArrayList`. Cuando hablamos de “array” nos referimos a las listas que estudiamos en el tema 1. Cuando hablamos de `ArrayList`, nos referimos a la clase del Java Collection Framework que implementa la interfaz `List`

Ejercicio 26 : Haz un programa que pida 6 palabras al usuario y las guarde en un array. El programa deberá recorrer el array y mostrar por pantalla cuántas palabras contienen la letra “a”.

6.1.- Repeticiones determinadas con interrupción

Como vimos al principio del tema las repeticiones determinadas con interrupción son una mezcla donde unas repeticiones determinadas pueden ser interrumpidas en función de una condición booleana. Es decir:

3. Repeticiones indeterminadas → hay una cantidad desconocida de repeticiones
4. Repeticiones determinadas → hay una cantidad exacta de repeticiones
5. Repeticiones determinadas con interrupción → hay cantidad máxima de repeticiones, que puede alcanzarse o no, según una condición booleana.

Por tanto, la principal característica de las repeticiones con interrupción es que, además de un tope máximo de repeticiones, tenemos una condición que si vale false, las repeticiones deberán detenerse aunque no se haya llegado a la cantidad máxima prevista inicialmente.

Para hacerlas podemos usar el while y el for, aunque es un poco más sencillo con este último. En el siguiente ejemplo vamos a ver cómo realizarlas usando ambas posibilidades.

Ejemplo 8: Usando la librería de Mario, vamos a hacer un programa en el que haya un cañon en la pantalla que dispare a Mario cinco veces. Tras cada disparo, el programa preguntará al usuario si desea dejar de disparar.

Podemos hacer este programa fácilmente usando la versión “lenta pero segura” del bucle while, teniendo en cuenta que hay dos momentos en los que se pueden interrumpir las repeticiones:

```
1. // creo un cañon y un muñeco a su derecha
2. Cañon c = new Cañon(120,90);
3. Mario m = new Mario(620,90);
4. // hacemos una variable que nos va a llevar la cuenta de los disparos realizados
5. int contador=0;
6. // hacemos un bucle while "lento pero seguro"
7. while(repetir){
8.     // el cañon dispara a las coordenadas donde está Mario
9.     c.disparar(m.getX(),m.getY());
10.    // preguntamos al usuario si desea detener los disparos
11.    System.out.println("¿Desea detener los disparos?");
12.    String respuesta=new Scanner(System.in).nextLine();
13.    if(respuesta.equals("no")){
14.        // si el usuario escribe "no", incrementamos el contador de disparos
15.        contador++;
16.        // si el número de disparos es igual a 5, detenemos las repeticiones
17.        if(contador==5){
18.            repetir=false;
19.        }
20.    }else {
21.        // si el usuario escribe "si", detenemos las repeticiones
22.        repetir=false;
23.    }
24. }
```

Como vemos, este programa cumple el objetivo requerido, pero como siempre, podemos reestructurar la forma “fácil pero segura” para escribir menos. Basta observar que “repetir=false” debe cumplirse cuando se da una de las condiciones respuesta.equals(“si”) o contador==5. Por tanto, usando la operación && podemos mejorar el código así:

```
1. // creo un cañon y un muñeco a su derecha
2. Cañon c = new Cañon(120,90);
3. Mario m = new Mario(620,90);
4. // creo una variable que guarde la respuesta del usuario
5. String respuesta="";
6. // hacemos una variable que nos va a llevar la cuenta de los disparos realizados
7. int contador=0;
8. // hacemos un bucle while "lento pero seguro"
9. while(contador<5 && respuesta.equals("no")){
10.    // el cañon dispara a las coordenadas donde está Mario
11.    c.disparar(m.getX(),m.getY());
12.    // preguntamos al usuario si desea detener los disparos
13.    System.out.println("¿Desea detener los disparos?");
14.    respuesta=new Scanner(System.in).nextLine();
15.    contador++;
16. }
```

Escribiéndolo de esa forma, podemos identificar los tres elementos que nos dan lugar al bucle for:

```
1. // creo un cañon y un muñeco a su derecha
2. Cañon c = new Cañon(120,90);
3. Mario m = new Mario(620,90);
4. // creo una variable que guarde la respuesta del usuario
5. String respuesta="";
6. // hacemos un bucle while "lento pero seguro"
7. for(int i=0; i<5&&respuesta.equals("no"); i++){
8.     // el cañon dispara a las coordenadas donde está Mario
9.     c.disparar(m.getX(),m.getY());
10.    // preguntamos al usuario si desea detener los disparos
11.    System.out.println("¿Desea detener los disparos?");
12.    respuesta=new Scanner(System.in).nextLine();
13. }
```

Por tanto, aquí vemos como la programación de las repeticiones determinadas con interrupción en el for nos lleva a incluir dos condiciones en su parte central:

- La comprobación de que el contador sea menor del número máximo de veces que se pueden repetir las acciones.
- La comprobación de que un factor externo pueda interrumpir las repeticiones.

Ejercicio 27: Un profesor hace diez exámenes y calcula la nota de la asignatura haciendo la nota media, pero si alguna de las notas es suspenso, entonces la calificación de la asignatura es suspenso. Realiza un programa que pida una por una las notas y muestre al final la nota de la asignatura. El programa deberá detenerse si alguno de los exámenes está suspenso.

Ejercicio 28: Realiza un programa en el que haya una variable llamada "contraseña" iniciada con el valor que tú quieras. A continuación el programa pedirá al usuario que introduzca la contraseña. El usuario solo tiene 5 intentos para ponerla bien. En caso de que el usuario la acierte, el programa dirá "Acceso permitido". Si se agotan los intentos, se mostrará "Acceso denegado". En todo momento el programa mostrará el número de intentos restantes.

Ejercicio 29: Supongamos que un diccionario español – inglés tiene las palabras de la tabla. Realiza un programa que pregunte por teclado al usuario una palabra en español y nos salga su palabra traducida al inglés.

hola	casa	caballo	hacha	manzana	ventana	tortuga	ratón	mesa
hello	house	horse	axe	apple	window	turtle	mouse	table

6.2.- Generación de números

Aunque los usos prácticos del bucle for que hemos estudiado son muy útiles, ahora vamos a estudiar el más importante de todos, que es la generación de números.

Si observamos los ejemplos que hemos puesto del bucle for, siempre tenemos una variable contadora "i" que va cambiando en cada iteración. Por ejemplo:

```

1. for(int i=0;i<10;i++){
2.     System.out.println("En clase no se juega");
3. }

```

En este bucle, la variable “i” toma el valor inicial 0 y en cada iteración va cambiando hasta que llega a 10. Por tanto, a lo largo de todo el bucle, la “i” ha tomado los valores 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. Podemos decir que *“el for ha generado los números de 0 a 9”*.

Generar números consiste en hacer un for cuyo contador vaya tomando un número diferente en cada iteración.

Hay muchísimas situaciones en las que es necesario generar números para resolver un problema. Los números que se generan pueden ir de uno en uno hacia adelante, como acabamos de ver, pero también podemos generarlos por ejemplo, de dos en dos, así:

```

1. for(int i=0;i<10;i+=2){
2.     // genera los números 0,2,4,8
3. }

```

Incluso es posible generarlos al revés. Cambiaríamos los valores inicial y final y utilizaríamos los operadores de decremento para hacer que la variable contadora disminuya:

```

1. for(int i=10;i>=0;i--){
2.     // genera los números 10,8,6,4,2,0
3. }

```

De hecho, podemos hacer cosas más complejas, como por ejemplo, esta:

```

1. for(double d=5.3;d>0;d-=1.2){
2.     // genera los números 5.3, 4.1, 2.9, 1.7, 0.5
3. }

```

Ejemplo 9: Queremos mostrar por pantalla todos los números pares que hay entre 56 y 128.

Este problema se resuelve generando los números pedidos y mostrándolos por pantalla. Una primera forma de hacerlo sería generando todos los números entre 56 y 128 y hacer un “if” que detecte cuáles son pares, para imprimirlos por pantalla. Sería así:

```

1. for(int i=56;i<=128;i++){
2.     if(i%2==0){
3.         System.out.println("Número generado: "+i);
4.     }
5. }

```

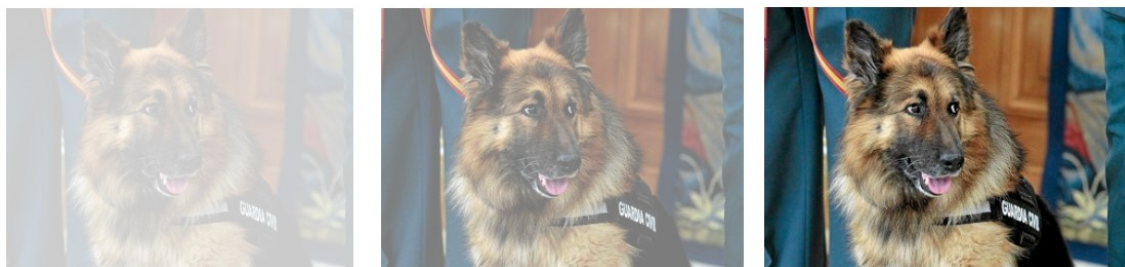
Pero una opción más eficiente sería hacer que la variable contadora “i” se incremente de dos en dos, así:

```

1. for(int i=56;i<=128;i+=2){
2.     System.out.println("Número generado: "+i);
3. }

```

Ejemplo 10: Queremos hacer un programa con la consola DAW en el que una imagen de fondo aparezca poco a poco en la pantalla (*efecto fade-in*).



A primera vista no parece un problema de repeticiones y mucho menos, de generar números. Sin embargo, analizándolo minuciosamente veremos que sí. ¿Cómo se hace que la imagen aparezca poco a poco? Observa que consiste en dibujar muchas veces la imagen, cada vez un poco más clara. Por tanto, ahí tenemos la repetición: tenemos que dibujar la imagen muchas veces. ¿Cuántas? desde que no se ve (su transparencia⁵ es 0), hasta que se ve completamente (su transparencia es 1).

Por tanto, estamos ante un problema de generar números. Tenemos que generar valores para la transparencia comprendidos entre 0 y 1. Por ejemplo, ese valor inicial puede ser 0 e incrementarse en 0.0002 cada vez (cuanto más pequeño sea el incremento más suave será la transición). Además, haremos una pausa de 1 milisegundo entre cada cambio de transparencia para controlar la velocidad del efecto.

```

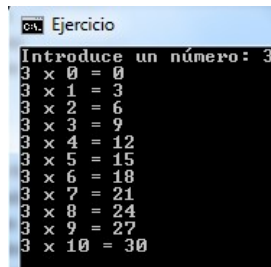
1. Consola c = new Consola();
2. CapaFondo cf = c.getCapaFondo();
3. try{
4.     BufferedImage foto=ImageIO.read(new File("foto.jpg"));
5.     FondoImagen fondo=new FondoImagen(foto);
6.     fondo.setEscalado(true);
7.     cf.setFondo(fondo);
8.     // generamos números de tipo float entre 0 y 1 incrementándose en 0.0002
9.     for(float alfa=0.0f; alfa<=1.0f; alfa+=0.0002f){
10.        cf.setAlfa(alfa);
11.        Thread.sleep(1);
12.    }
13. }catch(IOException | InterruptedException e){
14.     c.getCapaTexto().println(e.getMessage());
15. }

```

Ejercicio 30: Realiza un programa que pregunte al usuario una cantidad en segundos, borre la pantalla y con letras grandes y en el centro de la pantalla se muestre una cuenta atrás que se actualiza cada segundo. Al llegar a 0, el programa finalizará.

Ejercicio 31: Realiza un programa que pida un número entero al usuario y muestre por pantalla la tabla de multiplicar de dicho número.

⁵ La transparencia se denomina **canal alfa**, y en Java es un número de tipo float entre 0 y 1, por lo que para escribir su valor deberemos añadirle al número la letra **f** (por ejemplo, float a = 0.0f)



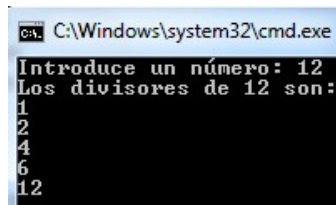
```
ca. Ejercicio
Introduce un número: 3
3 x 0 = 0
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

Ejercicio 32: Calcula la suma de los cuadrados de los números comprendidos entre 5 y 13:

$$5^2+6^2+7^2+8^2+9^2+10^2+11^2+12^2+13^2$$

Ejercicio 33: Realiza un programa que pregunte una frase al usuario. El programa mostrará cada letra de la frase en un renglón diferente.

Ejercicio 34: Haz un programa que pregunte un número al usuario y nos muestre la lista de todos sus divisores.



```
C:\Windows\system32\cmd.exe
Introduce un número: 12
Los divisores de 12 son:
1
2
4
6
12
```

Sugerencia: Genera todos los números desde 1 hasta el número y comprueba si cada número generado es un divisor.

Ejercicio 35: Haz un programa que pregunte un número al usuario y nos muestre por pantalla si es primo o no. De las dos formas que hay de hacer el ejercicio, elige la que más fácil te sea:

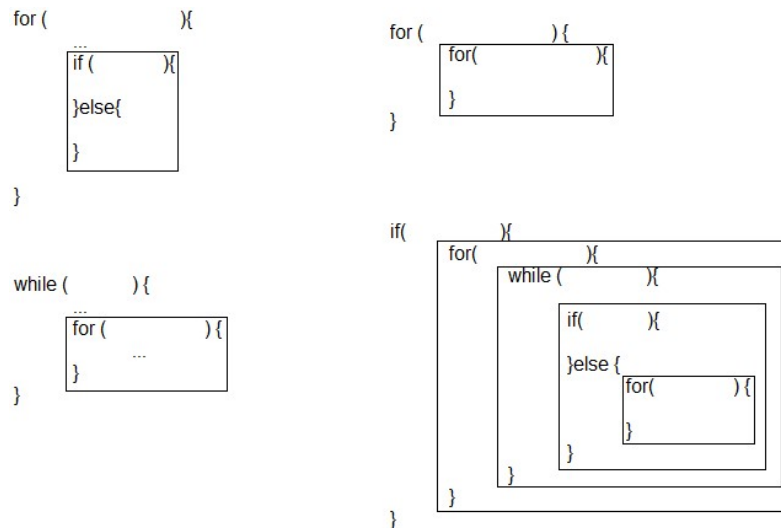
1. Se generan todos los números desde 2 hasta el que hay antes del número. Si alguno de los números generados es un divisor, entonces ya sabemos que el número no es primo. En ese caso, se interrumpe la generación de números y hemos terminado. Si por el contrario no hay divisores, entonces el número es primo.
2. Se generan todos los números desde 2 hasta el que hay antes del número y se cuenta en una variable auxiliar el número de divisores que se van encontrando. Si al finalizar el recorrido dicha variable es 0 el número es primo.

Ejercicio 36: Realiza un programa que pregunte al usuario una cadena de caracteres con su NIF. El programa deberá analizarla y decir si se trata de un NIF correcto o incorrecto. Un NIF correcto está formado por 8 dígitos y una letra. Los dígitos deben estar comprendidos entre 0 y 9 y la letra debe calcularse tal y como se indica en un ejercicio del tema anterior.

7.- La estructura iterativa anidada

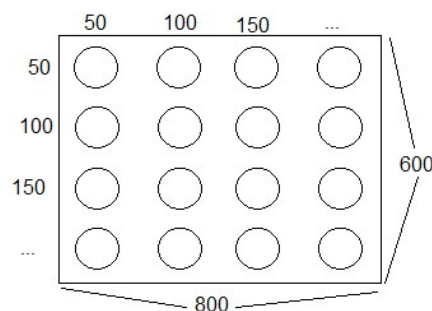
Al igual que ocurriría con la estructura condicional, la estructura iterativa también puede anidarse con cualquier otra estructura de la programación estructurada. Esto significa que dentro de una estructura iterativa podemos incluir un "if", un "for" o un "while". No hay ninguna restricción a la cantidad de estructuras que se pueden anidar.

Algunos ejemplos de estructuras anidadas:



Será la lógica de cada programa quien nos diga de forma natural qué estructuras debemos anidar.

Ejemplo 10: Queremos usar la clase **Graphics** para hacer un dibujo formado por círculos de 30 píxeles de radio (se indican por fuera las coordenadas de los círculos):



Hay dos formas de hacer este dibujo:

1. **forma bruta:** Consiste en dibujar uno por uno todos los círculos calculando sus coordenadas, pero esto es muy pesado, poco práctico y de difícil mantenimiento:

```
1. g.drawOval(50,50,30,30);
2. g.drawOval(100,50,30,30);
3. g.drawOval(150,50,30,30);
4. // etc
```

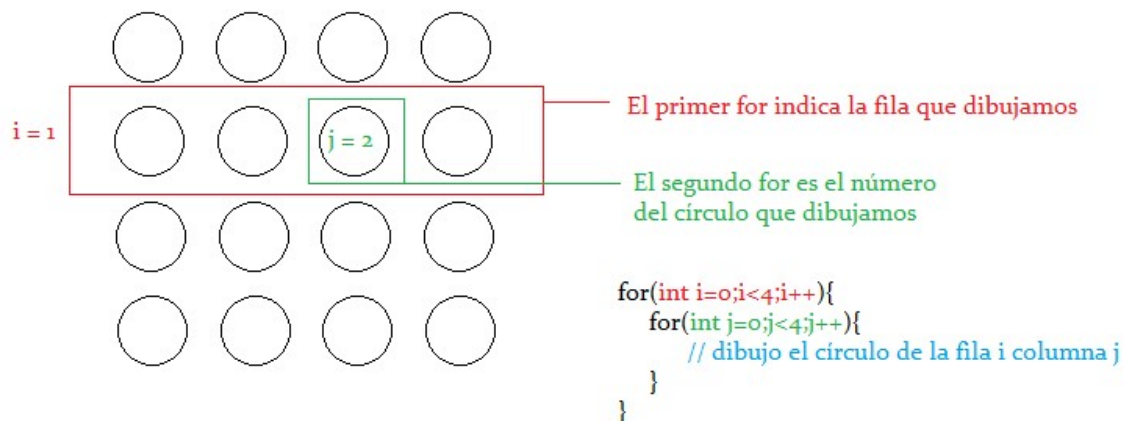

- **forma inteligente:** Nos damos cuenta de que el dibujo está formado por cuatro filas, y a su vez, cada fila está formada por cuatro círculos. Por tanto, comenzamos a hacer un bucle for que dibuje 4 filas:

```
1. // este for va a servir para dibujar las 4 filas
2. for(int i=0;i<4;i++){
3.     // aquí dibujaremos una fila de 4 círculos
4. }
```

Dentro de este for, hay que dibujar 4 círculos. Por tanto, dentro, debemos realizar un nuevo for que dibujará cada uno de los círculos.

```
1. // este for va a servir para dibujar las 4 filas
2. for(int i=0;i<4;i++){
3.     // aquí dibujamos una fila de 4 círculos
4.     for(int j=0;j<4;j++){
5.         // aquí dibujamos el círculo de la fila "i" y columna "j"
6.     }
7. }
```

Gráficamente es como si hiciéramos el dibujo línea a línea de arriba abajo, y dentro de cada línea hiciéramos los círculos de izquierda a derecha.



Ya solo nos queda usar el método drawOval para dibujar el círculo de la fila "i" y columna "j". ¿Qué coordenadas le ponemos a ese círculo? Si miramos el dibujo que hay que pintar, vemos que el centro del círculo de la fila "i" está en la coordenada 50*i, y el de la columna "j" está en la coordenada 50*j. El resultado final sería:

```
1. // este for va a servir para dibujar las 4 filas
2. for(int i=0;i<4;i++){
3.     // aquí dibujamos una fila de 4 círculos
4.     for(int j=0;j<4;j++){
5.         // aquí dibujamos el círculo de la fila "i" y columna "j" y radio 30
6.         g.drawOval(50*i,50*j,30,30);
7.     }
8. }
```

También podría haberse hecho de otra forma diferente creando dos variables al principio llamadas "x" e "y" que serían las coordenadas del primer círculo. Realizamos los mismos bucles que antes, y cuando dibujamos un círculo incrementamos la coordenada "x" en 50 píxeles. Es importante tener en cuenta que al terminar una fila, debemos restablecer la coordenada "x" con su valor inicial y bajar la coordenada "y".

```

1. // estas variables guardan las coordenadas del centro
2. // del círculo que se va a pintar
3. int x=100;
4. int y=100;
5. // este for va a servir para dibujar las 4 filas
6. for(int i=0;i<4;i++){
7.     // aquí dibujamos una fila de 4 círculos
8.     for(int j=0;j<4;j++){
9.         // dibujamos un círculo de radio 30 en las coordenadas (x,y)
10.        g.drawOval(x,y,30,30);
11.        // movemos la coordenada x unos 50 píxeles a la derecha
12.        x+=50;
13.    }
14.    // en este punto hemos dibujado una fila, así que bajamos la y unos 50 píxeles
15.    y+=50;
16.    /* no se nos debe olvidar volver a poner la x otra vez a la izquierda del dibujo
17.       porque vamos a dibujar una nueva línea */
18.    x=100;
19. }

```

Ejercicio 37: Realiza un programa que genere un documento PDF en el que se muestren todas las tablas de multiplicar del 1 al 10, cada una en una página diferente.

Ejercicio 38: Realiza un programa que pida diez notas por teclado y al final muestre su nota media. Si alguna nota introducida es incorrecta (negativa o superior a 10) el programa la preguntará nuevamente.

Ejercicio 39: Repite el ejercicio 22 de esta forma:

- Crea una lista para guardar los nombres de los alumnos
- Crea una tabla para guardar las notas de los alumnos, de tal forma que la fila 0 guarde las notas del alumno que ocupe la posición 0 de la lista y así sucesivamente.

☞ Recuerda que una tabla es una lista de listas y podemos crearla así:

1	2	3
4	5	6

```

1. int[][] numeros = {
2.     {1,2,3},
3.     {4,5,6}
4. };

```

- Haz un for para recorrer la lista de alumnos y después de mostrar el nombre de un alumno, haz un for anidado que recorra la fila de la tabla donde están sus notas y calcule su nota media.

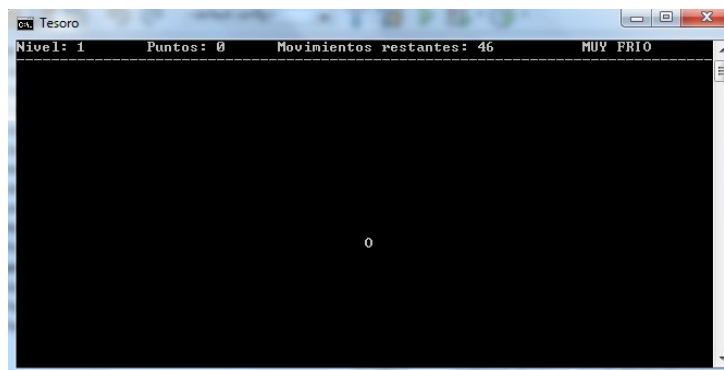
Ejercicio 40: En este ejercicio vas a hacer un programa que pida por teclado al usuario el nombre de un día y muestre tu horario de clases de ese día. Sigue estos pasos:

- Crea una lista para guardar los nombres de los días de la semana
- Crea una lista de String para guardar las horas (por ejemplo “16:00 – 16:55”)
- Crea una tabla 5 x 6 con las asignaturas que tienes cada día (cada columna es un día).
- Pide por teclado al usuario el nombre de un día
- Obtén la posición que tiene el día introducido dentro de la lista del apartado a)
- Recorre la columna correspondiente a ese día y muestra las asignaturas, poniendo también la hora correspondiente a cada una.

Ejercicio 41: El juego del tesoro

Usa la capa de texto de la ConsolaDAW para realizar un sencillo juego que consiste en encontrar un tesoro oculto, de esta forma:

- El jugador parte con 0 puntos, nivel 1 y tiene 80 movimientos disponibles. La posición inicial del jugador es el punto de coordenadas (40,15)
- El ordenador calcula las coordenadas (x,y) del tesoro de forma aleatoria, teniendo en cuenta el número máximo de filas y columnas de la capa de texto.
- El jugador es una letra O situada en las coordenadas del jugador, mientras que el tesoro no se ve. La pantalla muestra los puntos, nivel y movimientos disponibles.



- El jugador usa las siguientes teclas para moverse por la pantalla:

W	Arriba
A	Izquierda
D	Derecha
S	Abajo

- Cada vez que se mueve el jugador, disminuye en uno su número de movimientos disponibles. Si éste llega a cero, se acaba la partida y se muestra en pantalla con el signo \$ la posición del tesoro.
- Al moverse el jugador, en la parte superior de la pantalla aparecerán las palabras MUY FRIO, FRIO, CALIENTE o QUEMANDO, según sea la distancia desde el tesoro hasta el jugador, que se calcula con la siguiente fórmula:

$$d = \sqrt{(x_j - x_t)^2 + (y_j - y_t)^2}$$

Donde suponemos que la posición del jugador es (x_j, y_j) y la del tesoro es (x_t, y_t) . Tras calcular la distancia buscamos en la tabla la palabra correspondiente:

Distancia	Palabra
Mayor de 35	Muy frio
Entre 15 y 35	Frio
Entre 5 y 15	Caliente
Menos de 5	Quemando

- Si el jugador se mueve y acierta con la posición del tesoro, se informará al usuario y se pasará al siguiente nivel. Los puntos se incrementarán con el número de movimientos que le queden. El número de movimientos disponible se incrementará en 10 unidades al pasar al siguiente nivel.

Ejercicio 42: Haz un programa que pregunte al usuario el número total de filas y el número total de columnas, y cree una tabla de números, en la que cada número es la suma de su número de fila y número de columna. Aquí tienes un ejemplo si elegimos 3 filas y 3 columnas:

```

run:
Número de filas
3
Número de columnas
3
0 1 2
1 2 3
2 3 4

```

Ejercicio 43: Haz un programa que cree una tabla de 4 filas y 3 columnas rellena con booleanos aleatorios. El programa mostrará la tabla en pantalla y después la recorrerá y nos dirá cuántos true y cuántos false hay en la tabla.

8.- El for mejorado

En programación una situación que aparece muchas veces es que tenemos un array y tenemos que visitar todos sus elementos para hacer algo con ellos. Por ejemplo, mostrarlos por pantalla. En general, esto se llama “**recorrido**” del array.

En los ejercicios que hemos hecho ya hemos tenido que recorrer arrays. Por ejemplo, el siguiente programa crea un array con los nombres de los días laborables y los muestra por pantalla.

```

1. String[] laborables={"lunes","martes","miércoles","jueves","viernes"};
2. // la variable i va a guardar las posiciones que quiero recorrer (0,1,2,3,4,5,6)
3. for(int i=0;i<laborables.length;i++){
4.     // obtengo el día de la posición i y lo muestro en pantalla
5.     System.out.println(laborables[i]);
6. }

```

No solo los arrays pueden ser recorridos. Como hemos visto en los ejercicios, la interfaz `List<T>` también representa una lista⁶ que puede ser rellena con objetos del tipo `T`. Por ejemplo, lo anterior también lo podíamos haber hecho así:

⁶ Los arrays son estructuras estáticas, porque su tamaño no cambia. En cambio `List<T>` es una lista dinámica, porque le podemos añadir y borrar elementos. Usar arrays es más eficiente que `List<T>`, pero esta última admite muchas más posibilidades, por lo que tiende a usarse más.

```

1. List<String> laborables=Arrays.asList("lunes","martes","miércoles","jueves","viernes");
2. // genero las posiciones i=0,1,2,3,4,5,6
3. for(int i=0;i<laborables.size();i++){
4.     // por cada i, obtengo el día de la posición i y lo muestro en pantalla
5.     String dia=laborables.get(i);
6.     System.out.println(dia);
7. }

```

Como hacer un recorrido de un array o un List<T> es algo muy habitual, en Java 5 (2004) apareció el “**for mejorado**” (*enhanced for*), cuya finalidad es simplificar los recorridos de los arrays y los List<T>.

El for mejorado nos proporciona una forma sencilla de hacer un recorrido de inicio a fin de un array o un List<T>. El for del ejemplo anterior lo haríamos así:

```

1. List<String> dias=Arrays.asList("lunes","martes","miércoles","jueves","viernes");
2. // hago un for mejorado que recorre la lista de días laborables
3. // la variable "dia" se actualiza cada iteración con un nuevo día de la semana
4. for(String dia : laborables){
5.     System.out.println(dia);
6. }

```

Como podemos observar, los componentes del for mejorado son:

for(String dia : laborables) {
 ↑ ↑ ↑
tipo de los datos que hay en la lista día de la semana recorrido lista de días

En general, usar un for mejorado es mucho más sencillo que hacer un bucle for tradicional que primero genere las posiciones y luego extraiga los datos de la lista. Sin embargo, el for mejorado tiene algunos inconvenientes:

- Solo sirve para realizar recorridos hacia adelante.
- No es posible realizar recorridos con interrupción. Por ejemplo, a veces puede ser que nos interese visitar los datos de la lista hasta encontrar alguno que nos interese y luego ya no seguir visitando más. Para hacer eso necesitaríamos interrumpir las repeticiones, y eso no se *debe*⁷ hacer con el for mejorado.
- No es posible acceder a la posición del dato que estamos recorriendo, por lo que no podremos hacer bucles en los que necesitemos ese dato.

Ejercicio 44: Con el bloc de notas haz un archivo llamado **palabras.txt** y rellénalo con las 10 palabras que tú quieras. A continuación, haz un programa que abra ese archivo y lea todas las palabras del archivo, guardándolas en un **List<String>**. Por último, recorre ese List<String> usando un for mejorado y muestra cada palabra en pantalla.

⁷ Es posible detener un for mejorado usando la palabra **break**, pero su uso se desaconseja (en la Universidad es motivo de suspenso) porque cuando se usa mal da lugar a código de difícil mantenimiento.