Ciclos y contadores

¡Comencemos!

¿Qué aprenderás?

- Aplicar los distintos bloques repetitivos.
- Usar los bloques repetitivos para ejecutar operaciones con contadores.

Introducción

Previamente resolvimos ejercicios de ciclos, donde el fin del ciclo estaba determinado por el ingreso de un dato por parte del usuario. A continuación resolveremos problemas que requieren una cantidad determinada de ciclos o iteraciones. Por ejemplo, repetir un ciclo 10 veces, una cuenta regresiva o problemas de sumatorias.

¡Vamos con todo!

Iterar

Iterar es dar una vuelta al ciclo. Hay muchos problemas que se pueden resolver de forma mucho más eficiente iterando, por ejemplo, contar desde 0 hasta 10.

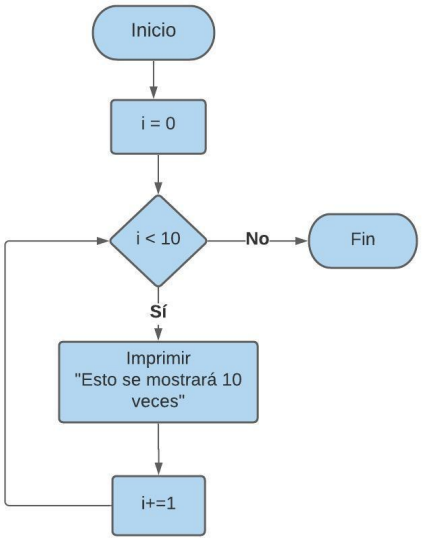

Imagen 1. Diagrama de flujo iterando 10 veces.
Fuente: Desafío Latam.

Contando con While

```
int i = 0;
while (i<10) {
    System.out.printf("Esto se mostrará 10 veces\n");
    i += 1 ; //IMPORTANTE Java
}
```

La instrucción `System.out.printf ("Esto se mostrará 10 veces\n")` se repetirá hasta que la variable `i` alcance el valor 10. Para entonces, la comparación de la instrucción `while` se evaluará como `false` y saldremos del ciclo.

En programación es una convención ocupar una variable llamada `i` como variable de iteración para operar en un ciclo.

Importante: Si no aumentamos el valor de la variable `i` entonces nunca llegará a ser igual o mayor a 10.

Que significa i+=1;

`+=` es un operador de asignación, muy similar a decir `a = 2` pero la diferencia es que con `+=` estamos diciendo "el valor anterior más 1".

```
a = 2; // aquí se asigna el valor 2 a la variable a
a += 2; /*aquí el valor de la variable aumentó en dos, y fue almacenada
nuevamente en a. Es decir ahora a = 4*/
a = a+2; //esto equivale a la sentencia anterior.
```

Transformando do a do while

Si bien con la sentencia while podemos mostrar 10 veces un mismo mensaje, también podemos usar el operador do while.

```
int i = 0;
do{
    System.out.printf("Esto se mostrará 10 veces\n");
    i += 1 ; //Importante
} while (i<10) ;
```

En este caso, se realizará al menos una vez, la operación de mostrar por pantalla y sumar 1 a la variable `i`, mientras `i` sea menor que 10. Una vez que `i` toma el valor 10, ya no se seguirá repitiendo el ciclo.

Importante: i += 1 es un incremento de 1 en 1. En este caso también se podría escribir `i++`; lo cual incrementará de 1 en 1 la variable `i`.

Operadores de asignación

La siguiente tabla muestra el comportamiento de los operadores de asignación.

Operador	Nombre	Ejemplo	Resultado
=	Asignación	a = 2	a toma el valor de 2
+=	Incremento y asignación	a += 2	a es incrementado en 2 y asignado el valor resultante
-=	Decremento y asignación	a -= 2	a es reducido en 2 y asignado el valor resultante
++	Incremento de 1 en 1	a++	Incrementa o suma 1 a la variable a
--	Decremento de 1 en 1	a--	Reduce o resta 1 a la variable a
*=	Multiplicación y asignación	a *= 3	a es multiplicado por 3 y asignado el valor resultante
/=	División y asignación	a /= 3	a es dividido por 3 y asignado el valor resultante

Tabla 1: Operadores de asignación.
Fuente: Desafío Latam.

Ejercicio guiado: La bomba de tiempo

Crearemos un algoritmo sencillo que realice una cuenta regresiva de 5 segundos.

Contar de forma regresiva es muy similar, solo debemos comenzar desde el valor correspondiente e ir disminuyendo su valor de uno en uno.

```
int i = 5;
while (i > 0) { //cuando lleguemos a cero terminamos.
    System.out.printf("%d\n",i);
    i--; //en cada iteración descontamos 1
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Para hacer que el programa espere de a 1 segundo antes de ejecutar otra sentencia, debemos agregar el paquete de `import java.util.concurrent.TimeUnit;` al inicio del programa y se utilizará el método `TimeUnit.SECONDS.sleep(1);` donde el parámetro 1 indica que la espera será de un segundo.

Incursionando en java: Try Catch

En Java los errores en tiempo de ejecución se denominan **excepciones**, y esto ocurre cuando se ejecuta un error en alguna de las instrucciones de nuestro programa. Por ejemplo, cuando se hace una división por cero o se abre mal un fichero.

Cuando se levanta una excepción, el programa se detiene en ese instante y no continua ejecutando las otras instrucciones. En este caso, se crea un objeto de una determinada clase (dependiendo del error), que nos proporcionará información sobre ese error. Todas estas clases tienen como clase padre **Throwable** (más adelante entraremos más en detalle sobre herencia, clases padres).

Java tiene un buen control de excepciones, evitando así que se detenga inesperadamente nuestro programa y, aunque se produzca una excepción, se siga ejecutando. Para ello tenemos la siguiente estructura:

```
try {
    // Instrucciones cuando no hay una excepción
} catch (TypeException ex) {
    // Instrucciones cuando se produce una excepción
} finally {
    // Instrucciones que se ejecutan, tanto si hay como si no hay
excepciones
}
```

En un bloque de código try-catch se puede omitir el catch o finally, pero no ambos.

En el ejercicio guiado: La bomba de tiempo

El método `TimeUnit.SECONDS.sleep(1);`

Si tratamos de ejecutar el método sin utilizar `try catch`, nos mostrará el siguiente error:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Unhandled exception type InterruptedException
```

Ya que este método requiere control de excepciones, por ello se escribe de la siguiente manera.

```
try {
    TimeUnit.SECONDS.sleep(1);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```