2.2 Bucles

1. Introducción

En programación, un **bucle** o *instrucción de control repetitiva* (o *iterativa*) permite ejecutar una o más instrucciones varias veces, es decir, permite ejecutar un bloque de instrucciones repetidamente, escribiéndolas una sola vez en el programa, reduciendo de este modo el código del mismo. Cada vuelta que da el bucle se conoce como **iteración**.

Las variables que se utilizan en los bucles pueden tener distintas funciones:

- **índice**: es una variable que apunta a un recurso concreto. En los bucles, apunta al número de iteración. Se suelen utilizar por convenio los identificadores de variables *i*, *j* y *k*.
- **contador**: cuenta cuántas veces ocurre algo.
- **acumulador**: acumula operaciones parciales.

Las instrucciones básicas que permiten construir este tipo de estructuras son while, do-while y for.

2. Bucle while

El bucle while agrupa instrucciones las cuales se ejecutan continuamente mientras una condición que se evalúa sea verdadera. La condición se evalúa antes de entrar dentro del while y cada vez que se termina de ejecutar las instrucciones del while.

```
while (expresiónLógica) {
    instrucciones //se ejecutan si la expresion lógica es true
}
```

El programa se ejecuta siguiendo estos pasos:

1. Se evalúa la expresión lógica.
2. Si la expresión es verdadera ejecuta las instrucciones, sino el programa abandona la sentencia while.
3. Tras ejecutar las instrucciones, volvemos al paso 1.

```
package tema2_2_Bucles;

import java.util.Random;

public class While1 {
```

```
    public void show() {

        int number;
        Random random = new Random();

        /*
         * Se obtiene un número aleatorio entre 1 y 500.
         * Mientras dicho número sea par, continuamos
         * en el bucle while.
         */
        while ((number = random.nextInt(500) + 1) % 2 == 0) {
            System.out.println(number);
        }

    }

    public static void main(String[] args) {

        new While1().show();

    }

}
```

3. Bucle do-while

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

La única diferencia respecto al while está en que la expresión lógica se evalúa después de haber ejecutado las instrucciones. Es decir, el bucle al menos se ejecuta una vez. Los pasos son los siguientes:

1. Ejecutar instrucciones.
2. Evaluar expresión lógica.
3. Si la expresión es verdadera, volver al paso 1. Si es falsa, salir del do-while.

Ejemplo que muestra números aleatorios entre 1 y 500 hasta que salga uno múltiplo de 7:

```
package tema2_2_Bucles;

import java.util.Random;

public class DoWhile1 {

    public void show() {

        boolean exit;
        int number;
        Random random = new Random();

        do {
```

```
            number = random.nextInt(500) + 1;//Se calcula un número aleatorio
entre 1 y 500
            System.out.println(number);
            exit = (number % 7 == 0); //exit se pone a true si el número es
múltiplo de 7
        } while (!exit);// Condición de repetición: que exit sea false

    }

    public static void main(String[] args) {

        new DoWhile1().show();

    }

}
```

Si le añadimos al ejemplo anterior que solamente haya 5 posibilidades para encontrar el múltiplo de 7, eso significa que necesitamos una variable que apunte al número de iteración del bucle para poder controlar que solamente se ejecute 5 veces, es decir, necesitamos un índice:

```
package tema2_2_Bucles;

import java.util.Random;

public class DoWhile2 {

    public void show() {

        boolean exit;
        int number, i = 1;//i es un índice porque apunta al número de iteración
del bucle
        Random random = new Random();

        do {
            number = random.nextInt(500) + 1;
            System.out.println(number);
            exit = (number % 7 == 0);
            i++;
        } while (!exit && i <= 5);//Mientras exit sea false y además i sea menor
o igual a 5

        if (!exit) {
            System.out.println("No se encontró el múltiplo de 7");
        }

    }

    public static void main(String[] args) {

        new DoWhile2().show();

    }
```

```
}
```

El bucle do-while se utiliza cuando se sabe que las instrucciones del bucle se van a ejecutar al menos una vez. En el bucle while puede ser que no se ejecuten nunca si la condición es falsa desde un principio. Las peticiones de usuario se realizan con bucles do-while, ya que la petición del dato al usuario siempre se realiza, y si el usuario no introduce lo que le hemos pedido, entonces utilizamos el bucle para volvérselo a pedir.

```
package tema2_2_Bucles;

import java.util.Scanner;

public class DoWhile3 {

    @SuppressWarnings("resource")
    public void show() {

        Scanner keyboard = new Scanner(System.in);
        int number;
        do {
            System.out.print("Introduzca un número del 1 al 5: ");
            number = keyboard.nextInt();
            System.out.printf("Has introducido un %d\n", number);
        } while (number < 1 || number > 5);

    }

    public static void main(String[] args) {

        new DoWhile3().show();

    }

}
```

Veamos un ejemplo con índice, contador y acumulador:

```
package tema2_2_Bucles;

import java.util.Scanner;

public class DoWhile4 {

    @SuppressWarnings("resource")
    public void show() {

        Scanner keyboard = new Scanner(System.in);
        int number, counter = 0, sum = 0, index = 0;
        /*
         * index: es un índice porque apunta al número de iteración del bucle
         * counter: es un contador para contar el número de pares
         * sum: es un acumulador donde se acumula la suma de los números pares
         *
         */
```

```
        do {
            System.out.print("Introduzca un número ó 0 para terminar: ");
            number = keyboard.nextInt();
            if (number != 0) {
                index++;//Se incrementa el número de iteración del bucle
                System.out.printf("El número introducido en la iteración %d es
%d\n", index, number);
                if (number % 2 == 0) {//Si el número es par
                    counter++;//Se incrementa el contador
                    sum += number;//Se acumula la suma en el acumulador
                }
            }
        } while (number != 0);

        System.out.printf("En %d iteraciones se han introducido %d números pares
cuya suma vale %d", index, counter, sum);

    }

    public static void main(String[] args) {

        new DoWhile4().show();

    }

}
```

4. Bucle for

```
for(inicialización;condición;incremento){
    instrucciones
}
```

El funcionamiento es el siguiente:

1. Se ejecuta la instrucción de inicialización.
2. Se comprueba la condición.
3. Si la condición es cierta, entonces se ejecutan las instrucciones. Si la condición es falsa, se abandona el bloque for.
4. Tras ejecutar las instrucciones, se ejecuta la instrucción de incremento y se vuelve al paso 2.

Ejemplo para escribir números del 1 al 100:

```
package tema2_2_Bucles;

public class For1 {

    public void show() {

        int i;

        for (i = 1; i <= 100; i++) {//i es un índice porque apunta al número de
iteración del bucle
```

```
            System.out.println(i);
        }

    }

    public static void main(String[] args) {

        new For1().show();

    }

}
```

En el eclipse, si seleccionamos varias líneas de código, las podemos fácilmente insertar en un if o en un bucle: Menú *Source* → *Surround with*.

También es posible declarar la variable dentro del propio bucle for, pero su ámbito de vida se reduce exclusivamente al bucle for, es decir, no se conoce fuera del for ya que muere en cuanto el bucle finalice:

```
package tema2_2_Bucles;

public class For2 {

    public void show() {

        for (int i = 1; i <= 100; i++) {//se declara la variable i como int
dentro del for
            System.out.println(i);
        }
        i = 1;//Error de compilación

    }

    public static void main(String[] args) {

        new For2().show();

    }

}
```

El bucle for también admite más de una variable, en cuyo caso, en las partes de inicialización e incremento, se utiliza como separador la coma `,`:

```
package tema2_2_Bucles;

public class For3 {

    public void show() {

        for (int i = 0, j = 10; i <= j; i++, j--) {
            System.out.printf("i: %d j:%d\n", i, j);
        }
```

```
    }

    public static void main(String[] args) {

        new For3().show();

    }

}
```

Los bucles for también se pueden hacer con while o do-while pero los for tienen una sintaxis más abreviada. Veamos el ejemplo *For1* hecho con un while:

```
package tema2_2_Bucles;

public class While2 {

    public void show() {

        int i = 1;

        while (i <= 100) {
            System.out.println(i);
            i++;
        }

    }

    public static void main(String[] args) {

        new While2().show();

    }

}
```

Entonces, si todos los bucles de pueden hacer con for, while y do-while, ¿cuál debemos usar? Es importante que el programador utilice la estructura más adecuada en cada caso:

- Cómo saber cuándo hay que utilizar un for: cuando se sepa el número de veces que se va a ejecutar el bucle, es decir, cuando se conozcan el número de iteraciones de antemano o lo tengamos almacenado en alguna variable. Por motivos de legibilidad de código, es muy importante que las variables del for aparezcan en las 3 partes del for, es decir, en la inicialización, en la condición y en el incremento, ya que el programador que quiera saber cuántas iteraciones realiza el bucle, solamente con fijarse en la línea de código donde se encuentra el for, pueda saberlo. Si las variables del for no aparecen en las 3 partes, eso nos indica que es más conveniente realizar un while o un do-while.
- Si no se sabe de antemano el número de iteraciones, entonces hay que utilizar while o do-while:
    - Cómo saber cuándo hay que utilizar un do-while: cuando las instrucciones del bucle se van a ejecutar al menos una vez.

- Cómo saber cuándo hay que utilizar un while: cuando pueda ocurrir que las instrucciones del bucle no se ejecuten nunca si la condición es falsa desde un principio.

Si las variables del for no aparecen en las 3 partes, eso nos indica que es más conveniente realizar un while o un do-while. Veamos el ejemplo que muestra números aleatorios entre 1 y 500 hasta que salga uno múltiplo de 7:

```
package tema2_2_Bucles;

import java.util.Random;

public class For4 {

    public void show() {

        boolean exit = false;
        int number = 0, i;
        Random random = new Random();

        for (i = 1; !exit; i++) {//Este for no es legible, se debe sustituir por un do-while
            number = random.nextInt(500) + 1;
            System.out.printf("Iteración %d, número: %d\n", i, number);
            exit = (number % 7 == 0);
        }
        System.out.printf("El múltiplo de 7 con valor %d se ha encontrado en la iteración %d", number, i - 1);

    }

    public static void main(String[] args) {

        new For4().show();

    }

}
```

Si le añadimos al ejemplo anterior que solamente haya 5 posibilidades para encontrar el múltiplo de 7:

```
package tema2_2_Bucles;

import java.util.Random;

public class For5 {

    public void show() {

        boolean exit = false;
        int number = 0, i;
        Random random = new Random();
        /*
         * Ahora el for sí es legible porque la variable i
         * aparece en las 3 partes: en la inicialización,
```

```
         * en la condición y en el incremento
         */
        for (i = 1; i <= 5 && !exit; i++) {
            number = random.nextInt(500) + 1;
            System.out.printf("Iteración %d, número: %d\n", i, number);
            exit = (number % 7 == 0);
        }
        if (exit) {
            System.out.printf("El múltiplo de 7 con valor %d se ha encontrado en
la iteración %d", number, i - 1);
        } else {
            System.out.println("El múltiplo de 7 no se ha encontrado");
        }

    }

    public static void main(String[] args) {

        new For5().show();

    }

}
```

También es muy importante por motivos de legibilidad del código, que el único sitio donde se modifiquen los valores de las variables del for sea en la zona del incremento ya que el programador que quiera saber cuántas iteraciones realiza el bucle, solamente con ver la línea de código del for, pueda saberlo. Por ejemplo, este código no es adecuado ya que para salirse del bucle del for, se está modificando la variable *i*.

```
package tema2_2_Bucles;

import java.util.Random;

public class For6 {

    public void show() {

        int number = 0, i;
        Random random = new Random();

        for (i = 1; i <= 5; i++) {
            number = random.nextInt(500) + 1;
            System.out.printf("Iteración %d, número: %d\n", i, number);
            if (number % 7 == 0) {
                i = 5;//No se pueden modificar las variables del for, solamente
en el incremento
            }
        }
        if (number % 7 == 0) {
            System.out.printf("El múltiplo de 7 encontrado vale %d", number);
        } else {
            System.out.println("El múltiplo de 7 no se ha encontrado");
        }
```

```
    }

    public static void main(String[] args) {

        new For6().show();

    }

}
```

5. Bucle for-each

Una de las cosas que incorporó Java 5 fue el bucle for-each. Esta estructura nos permite recorrer una colección o un array de elementos de una forma sencilla, evitando el uso de iteradores o de un bucle for normal.

```
package tema2_2_Bucles;

public class ForEachArrays {

    public void show() {

        int[] grades = { 8, 7, 9 };

        for (int grade : grades) {
            System.out.printf("Nota: %d\n", grade);
        }

    }

    public static void main(String[] args) {

        new ForEachArrays().show();

    }

}
```

También se usa el for-each para recorrer los valores de un enum. Para ello, se utiliza el método *values()* de los enum que está implícitamente declarado por el compilador:

```
package tema2_2_Bucles;

public class ForEachEnum {

    public enum DayOfWeek {
        MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
    }

    public void show() {

        for (DayOfWeek d : DayOfWeek.values()) {
            System.out.println(d);
        }
```

```
    }

    public static void main(String[] args) {

        new ForEachEnum().show();

    }

}
```