Notas Java - 2. Operadores y Control de Flujo

Cristo Alvarado

11 de julio de 2025

Resumen

Esta sesión profundiza en el manejo de estructuras de control y operadores en Java, abordando ciclos avanzados (`do while`), mecanismos de interrupción (`break/continue`), arreglos (unidimensionales/multidimensionales) y diseño de métodos (parámetros, retornos, sobrecarga). Se integrará teoría con ejercicios prácticos de validación de entradas, algoritmos de filtrado y manipulación de matrices. El alumno fortalecerá competencias en diseño modular y gestión eficiente de flujos programáticos. Revise previamente sintaxis de ciclos y operadores lógicos.

Índice

§1 `do while` y Ejemplos de Ciclos **2**
§1.1 `do while` 2
§1.2 Ejemplos y Ejercicios 3
§2 `break/continue` y Arreglos **5**
§2.1 `break/continue` 5
§2.2 Arreglos 6
§2.3 Ciclo sobre Arreglo 6
§2.4 Arreglos Multidimensionales 7
§3 Métodos **8**
§3.1 Creación de Métodos 8
§3.2 Llamar un Método 9
§3.3 Parámetros de un Método 9
§3.4 Valores de Retorno (Return Values) 10

Lista de Códigos

1 Sintaxis `do while` 2
2 Diferente ejecución entre `while` y `do while`. 3
3 Misma ejecución de `while` y `do while`. 3

4 Programa que imprime números primos entre 1 y 100. . . . 3
5 Programa que imprime un triángulo de tamaño $n = 3$. . . . 4
6 Programa que recibe entero entre 0 y 10. . . . 4
7 Programa que sale de un ciclo `for` al llegar al `4`. . . . 5
8 Programa usando el ciclo `for` que no imprime `4`. . . . 5
9 Declaración de un arreglo de algún tipo de dato. . . . 6
10 Declaración de un arreglo con valores dados. . . . 6
11 Ejemplo Arreglos. . . . 6
12 Acceder a variable arreglo. . . . 6
13 Recorrer arreglo. . . . 6
14 Recorrer arreglo con `for-each`. . . . 7
15 Sintaxis `for-each`. . . . 7
16 Arreglo Bidimensional. . . . 7
17 Acceder Elementos de Arreglo Bidimensional. . . . 7
18 Ciclo `for` en Arreglo Bidimensional. . . . 7
19 Ciclo `for-each` en Arreglo Bidimensional. . . . 8
20 Método dentro de `Main`. . . . 8
21 Método dentro de `Main`. . . . 9
22 Método llamado varias veces. . . . 9
23 Parámetro en un Método. . . . 10
24 Método con Múltiples Parámetros. . . . 10
25 Ejemplo Método que Retorna `int`. . . . 10
26 Ejemplo Método que Retorna `int`. . . . 11
27 Sobrecarga de Métodos o Method Overloading. . . . 11

§1 `do while` y Ejemplos de Ciclos

En esta sección veremos el ciclo `do while` y algunos ejemplos de como usar los bucles en Java.

§1.1 `do while`

El ciclo `do while` es una variante del ciclo `while`. Este ciclo ejecuta el código **una vez SIEMPRE** y luego checa si la condición es verdadera, luego repite el ciclo hasta que la condición es verdadera.

```
do {
  // code block to be executed
}while(condition);
```

Código 1: Sintaxis `do while`

Observación 1.1
Después `while(condition)` siempre es necesario poner el punto y coma.

Observación 1.2
Diferencias entre `while` y `do while` son claras en el siguiente código:

```
public class Main{
    public static void main(String[] args){
        int i = 10;
        System.out.println("Ejecucion de do while:");
        do{
            System.out.println(i); //imprime el 10
            i++;
        }while(i < 5);
        System.out.println("Ejecucion de while:");
        i = 10;
        while(i < 5){
            System.out.println(i); //no imprime nada
            i++;
        }
    }
}
```

Código 2: Diferente ejecución entre `while` y `do while`.

Sin embargo, en ciertas circustancias se comportan prácticamente igual:

```
public class Main{
    public static void main(String[] args){
        int i = 0;
        System.out.println("Ejecucion de do while:");
        do{
            System.out.println(i); //imprime el 10
            i++;
        }while(i < 5);
        System.out.println("Ejecucion de while:");
        i = 0;
        while(i < 5){
            System.out.println(i); //no imprime nada
            i++;
        }
    }
}
```

Código 3: Misma ejecución de `while` y `do while`.

§1.2 Ejemplos y Ejercicios

Ejemplo 1.1 (Programa que Imprime Números Primos entre 1 y 100)
El siguiente programa imprime números primos entre 1 y 100.

```
public class Main{
    public static void main(String[] args){
        int p = 2, j;
```

```
            while(p <= 100){
                //verificar si el numero es primo
                j = 2;
                while(j < p){
                    if(p % j == 0){
                        j = -1;
                        break;
                    }
                    else j++;
                }
                if(j != -1) System.out.println("Es primo: " + p);
                p++;
            }
        }
    }
```

Código 4: Programa que imprime números primos entre 1 y 100.

Ejercicio 1.1
Modifique el siguiente código para usar `for` en vez de `while`.

Ejemplo 1.2 (**Programa que Imprime un Triángulo**)
El siguiente programa imprime un triángulo isóceles de tamaño $n = 3$:

```
public class Main{
    public static void main(String[] args){
        int n = 3;
        for(int i = 0; i <= n; i++){
            for(int j = 0; j < i; j++) System.out.print("*");
            for(int j = i+1; j < n; j++) System.out.print(" "
               );
            System.out.println("");
        }
    }
}
```

Código 5: Programa que imprime un triángulo de tamaño $n = 3$.

Ejercicio 1.2
¿Cómo haría un programa que tome como entrada un número n e imprima un triángulo isóceles de de lado n?

Ejemplo 1.3 (**Programa Que Recibe un Entero entre 0 y 10**)
El siguiente programa recibe un entero entre 0 y 10 y calcula su cuadrado, y no permite continuar a menos que el entero esté estrictamente entre 0 y 10.

```
import java.util.Scanner;

public class Main{
    public static void main(String[] args){
```

```
            int entrada = -1;
            Scanner flujo_entrada = new Scanner(System.in);
            do{
                System.out.print("Ingrese un numero entre 0 y 10:
                    ");
                entrada = flujo_entrada.nextInt();
            }while(entrada < 0 || entrada > 10);
            System.out.println("Su numero es: "+entrada);
            System.out.println("El cuadrado de su numero es: "+(
               entrada*entrada));
        }
    }
```

Código 6: Programa que recibe entero entre 0 y 10.

¿Qué pasa si no ingresamos un número entero en el programa? El código da error (la terminal muestra que ocurrió un error). Para solucionarlo debemos hacer una cosa que se llama **excepción de errores**. Más adelante veremos esto.

Ejercicio 1.3
En vez de que tome como entrada un entero, ¿cómo podríamos hacer que tome un número con decimales y calcule su cubo?

§2 `break`/`continue` y Arreglos

§2.1 `break`/`continue`

La sentencia `break` se ha usado anteriormente en la sentencia `swtich`. `break` es usado en ciclos para salir de un ciclo:

Ejemplo 2.1
En este ejemplo el ciclo `for` se detiene cuando `i == 4`:

```
    for (int i = 0; i < 10; i++) {
    if (i == 4) {
        break;
    }
    System.out.println(i);
    }
```

Código 7: Programa que sale de un ciclo `for` al llegar al 4.

La sentencia `continue` es similar a `break`, solo que `continue` termina la ejecución del código de la iteración actual y continua con la siguiente siguiente iteración en el ciclo.

Ejemplo 2.2

```
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        continue;
      }
```

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

Código 8: Programa usando el ciclo `for` que no imprime 4.

§2.2 Arreglos

Definición 2.1 (**Arreglos**)
Un **arreglo** es *usado para guardar múltiples valores de algún tipo en una sola variable.*

Los arreglos nos sirven para no tener que declarar múltiples variables para cada valor que queramos guardar. Para declarar un arreglo, usamos la siguiente notación:

```
    //tipoDato[] nombreVariable;
```

Código 9: Declaración de un arreglo de algún tipo de dato.

Para añadir valores a esta variable usamos corchetes `{}` y los valores los colocamos dentro, separados por comas:

```
    //tipoDato[] nombreVariable = {val1, val2, ... , valn};
```

Código 10: Declaración de un arreglo con valores dados.

Ejemplo 2.3 (**Ejemplos de Arreglos**)
En el siguiente código declaramos dos arreglos, uno de `String` y otro de `int`:

```
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    int[] myNum = {10, 20, 30, 40};
```

Código 11: Ejemplo Arreglos.

Para acceder a los elementos de un arreglo, colocamos corchetes después del nombre de la variable y luego, colocamos la posición en que se encuentra el valor que queremos usar:

```
    //nombreVariable[numeroPosicion]
```

Código 12: Acceder a variable arreglo.

Para conocer la posición, empezamos contando desde cero.

Ejemplo 2.4 (**Ejemplo Acceso Valor Arreglo**)
Para acceder al valor `"Ford"` del arreglo `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"}` debemos poner `cars[2]`. En este caso, `"Volvo"` está en la posición 0, `"BMW"` en la 1 y así sucesivamente.

§2.3 Ciclo sobre Arreglo

Para recorrer los elementos de un arreglo, usualmente usamos el ciclo `for`:

```
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

```
    for(int i = 0; i < cars.length; i++){
        System.out.println(cars[i]);
    }
```

Código 13: Recorrer arreglo.

Donde la propiedad `length` obtiene la longitud de nuestro arreglo. También podemos usar el `for-each` para recorrer los elementos de un arreglo:

```
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    for(String i : cars){
        System.out.println(i);
    }
```

Código 14: Recorrer arreglo con `for-each`.

La sintaxis del `for-each` es la siguiente:

```
    for(tipoVariableArreglo i : nombreArreglo){
        //bloque de codigo
    }
```

Código 15: Sintaxis `for-each`.

En este caso, para cada `i` de tipo `String` en el arreglo `cars`, imprimimos a `i`.

§2.4 Arreglos Multidimensionales

Siguiendo la idea de los arreglos, podemos hacer arreglos de arreglos, y arreglos de arreglos de arreglos, y sucesivamente, como en el siguiente código:

```
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

Código 16: Arreglo Bidimensional.

En este caso hacemos un arreglo del tipo de dato `int[]`, que es un arreglo. Para acceder a los elementos de este arreglo lo hacemos de la misma manera que como si fuese un arreglo normal, pero colocando corchetes dos veces y dos números en los corchetes:

```
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    System.out.println(myNumbers[1][2]); // Outputs 7
```

Código 17: Acceder Elementos de Arreglo Bidimensional.

Observación 2.1
También podemos usar el ciclo `for`, como se ve en el siguiente ejemplo:

```
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    for (int i = 0; i < myNumbers.length; ++i) {
      for (int j = 0; j < myNumbers[i].length; ++j) {
        System.out.println(myNumbers[i][j]);
      }
    }
```

Código 18: Ciclo `for` en Arreglo Bidimensional.

o también:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
for (int[] row : myNumbers) {
  for (int i : row) {
    System.out.println(i);
  }
}
```

Código 19: Ciclo `for-each` en Arreglo Bidimensional.

§3 Métodos

Definición 3.1 (**Método**)
Un **método** *es un bloque de código que es ejecutado cuando es llamado por algún otro método.*

Observación 3.1
El método `main()` es un método.

Se puede pasar información a un método. Esta *información es conocida como* **parámetros**. Los *métodos son usados para realizar alguna acción, estos también son conocidos como* **funciones**.

§3.1 Creación de Métodos

Un método debe ser declarado dentro de una clase. Se define el nombre del método, seguido de paréntesis `()`. Java da algunos métodos predefinidos, como vimos anteriormente, como el método `System.out.println()`, pero nosotros también podemos crear nuestros propios métodos para efectuar alguna acción.

Ejemplo 3.1
El siguiente es un método dentro de la clase `Main`:

```
public class Main {
    public static void main(String args[]){
        System.out.println("Hola Mundo");
    }
    static void myMethod() {
        System.out.println("Estoy ejecutando mi metodo");
        // codigo a ejecutar
    }
}
```

Código 20: Método dentro de `Main`.

En este ejemplo tenemos lo siguiente:

- `myMethod()` es el nombre del método.

- `static` significa que el método pertenece a la clase `Main` y no a un objeto de la clase `Main`. Luego se verá que significa esto último.
- `void` significa que este método no tiene un valor de retorno. Los valores de retorno son valores que una función puede regresar después de realizar ciertas operaciones.

§3.2 Llamar un Método

Para llamada un método en llava se escribe el nombre del método, seguido de dos paréntesis `()` y un punto y coma `;`.

Ejemplo 3.2 (Declaración y Llamada Método)

```
public class Main {
    public static void main(String args[]){
        System.out.println("Hola Mundo");
        myMethod();
    }
    static void myMethod() {
        System.out.println("Estoy ejecutando mi metodo");
    }
}
```

Código 21: Método dentro de `Main`.

Un método también puede ser llamado varias veces:

```
public class Main {
    public static void main(String args[]){
        System.out.println("Hola Mundo");
        myMethod();
        myMethod();
        myMethod();
    }
    static void myMethod() {
        System.out.println("Estoy ejecutando mi metodo");
    }
}
```

Código 22: Método llamado varias veces.

§3.3 Parámetros de un Método

Es posible pasar información a través de un método, esta información es conocida como **parámetros**. Los *parámetros actúan como variables dentro de un método.*

Los parámetros se especifican después del nombre del método, dentro de los paréntesis. Se pueden añadir cuantos parámetros uno desee, mientras estén separados por comas.

Ejemplo 3.3
Este ejemplo tiene un método que toma como parámetro un `String` y dentro del método este parámetro es llamado con el nombre de variable `nombre`. Cuando se llama al método, debemos siempre introducir este `String` o en caso contrario el sistema nos dará error.

```
public class Main {
  static void myMethod(String name) {
    System.out.println("Tu nombre es: " + name);
  }

  public static void main(String[] args) {
    myMethod("Daniel");
    myMethod("Alejandro");
    myMethod("Emilio");
  }
}
```

Código 23: Parámetro en un Método.

Ejemplo 3.4
En este ejemplo tenemos un método que recibe como entrada múltiples parámetros:

```
public class Main {
  static void myMethod(String name, int edad) {
    System.out.println("Tu nombre es: " + name + ", y tu edad
        es: " + edad);
  }

  public static void main(String[] args) {
    myMethod("Daniel", 23);
    myMethod("Alejandro", 18);
    myMethod("Emilio", 29);
  }
}
```

Código 24: Método con Múltiples Parámetros.

Observación 3.2
Dentro del método podemos hacer exactamente lo mismo que hacemos dentro del método `main()`, esto es, inicializar variables, objetos, usar `if`, `else`, `swtich`, los bucles `for`, `while`, etc...

§3.4 Valores de Retorno (Return Values)

En los ejemplos anteriores, al momento de declarar un método siempre usábamos la palabra `void` en todos los ejemplos. Esta palabra indica que el método no debe retornar valor alguno.

Si se quiere usar un método para regresar un valor, podemos usar un tipo de dato primitivo (como lo son `int`, `char`, etc...) en vez de `void`, y usar la palabra reservada `return` dentro del método:

Ejemplo 3.5
El siguiente ejemplo es un método que recibe dos parámetros y retorna un entero que es la suma de estos dos parámetros.

```
public class Main {
    static int myMethod(int x, int y) {
```

```
        return x + y;
    }

    public static void main(String[] args) {
        System.out.println(myMethod(5, 3));
    }
}
```

Código 25: Ejemplo Método que Retorna `int`.

El resultado del retorno también se puede guardar en una variable:

```
public class Main {
    static int myMethod(int x, int y) {
        return x + y;
    }

    public static void main(String[] args) {
        int z = myMethod(5, 3);
        System.out.println(z);
    }
}
```

Código 26: Ejemplo Método que Retorna `int`.

Observación 3.3
Java tiene una cosa denominada **Sobrecarga de Métodos**, que nos permite definir múltiples métodos con el mismo nombre, pero con parámetros diferentes:

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

Código 27: Sobrecarga de Métodos o Method Overloading.