

UNIDAD DIDÁCTICA 2: CONTROL DE FLUJO Y ARRAYS

ÍNDICE

1. Introducción.
2. Estructuras de selección.
 - 2.1. Estructura if.
 - 2.2. Estructura if-else.
 - 2.3. Estructura switch.
3. Estructuras de repetición.
 - 3.1. Bucle while.
 - 3.2. Bucle do...while.
 - 3.3. Bucle for.
4. Arrays.
 - 4.1. Declaración, instanciación e inicialización.
 - 4.2. Utilización de arrays.
 - 4.3. Arrays multidimensionales.
 - 4.4. Cadenas de caracteres.



1. Introducción

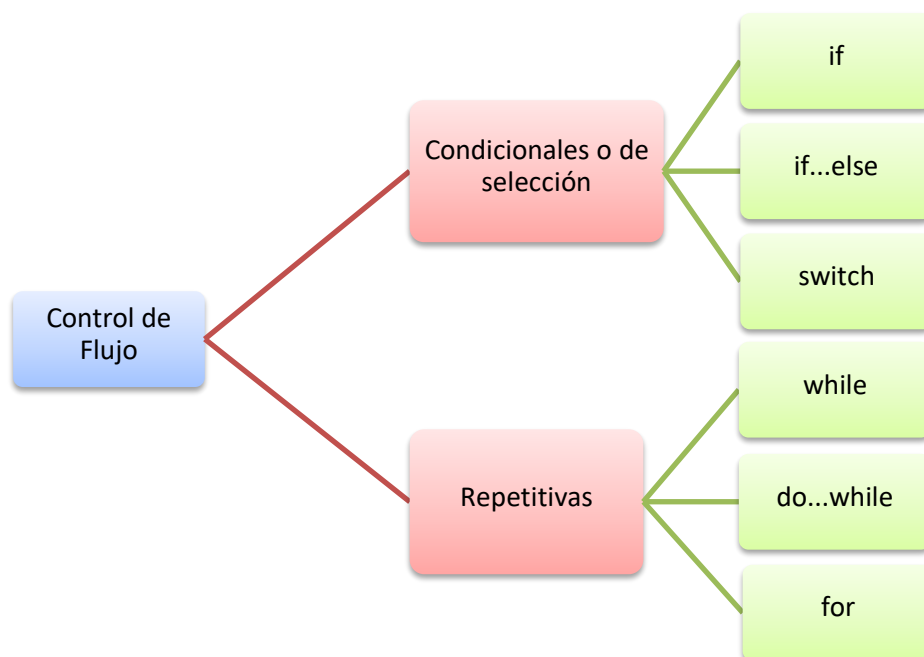
Como recordarás, en cualquier lenguaje, los tokens de distintos tipos se unían para formar expresiones que, aunque tienen sentido propio por sí mismas, no indican ninguna acción a realizar por el ordenador. La forma en que esas expresiones pasan a formar sentencias totalmente completas, con sentido pleno, y ejecutables por el ordenador, es el tema a tratar en esta unidad.

La diferencia entre una expresión (serie de variables, constantes y datos unidos por operadores, por ejemplo, $2*PI*radio$) y una sentencia es el **carácter de finalización de sentencia, que en Java es el punto y coma (;)**. Así, todas las sentencias terminarán en punto y coma.

Existen tres tipos básicos de sentencias en Java:

- **Sentencias de expresión.** Están formadas por una expresión seguida del carácter punto y coma (;).
- **Sentencias de declaración.** La declaración de una variable, en la que se le asocia al identificador de la variable un tipo, y opcionalmente un valor, que genera una sentencia, al añadirle el símbolo de terminación.
- **Sentencias de control de flujo.** Se encargan de "contener" a las otras sentencias del programa, de forma que se indique el orden en que se van a ejecutar esas sentencias, y bajo qué condiciones. Pueden considerarse como sentencias estructurales dentro del programa.

Estudiaremos en este tema las sentencias de control de flujo. Son FUNDAMENTALES para el desarrollo de cualquier programa.



Antes de escribir cualquier sentencia, es adecuado escribir algunas líneas, bien sea usando pseudocódigo o diagramas de flujo, para tener bien claro qué pretendemos y qué debe hacer nuestro programa. Veamos algunos ejemplos:

PSEUDOCÓDIGO

El pseudocódigo trata de hacer una aproximación de la codificación final de un programa sin usar un lenguaje de programación específico. Requiere, por tanto:

- Entender completa y detalladamente el problema, hacer un análisis minucioso antes del algoritmo.
- Subdividir el problema hasta la unidad más pequeña posible.
- Ordenar de forma secuencial estas unidades.

Pseudocódigo para un programa que lea dos números por teclado y muestre el resultado por pantalla:

```
Nombre Suma de dos enteros
Módulo Principal

DECLARAR VARIABLES
ENTERO num1
//Se pueden declarar varias variables en la misma línea siempre que sean del mismo tipo
ENTERO num2, suma

INICIO
Leer num1
Leer num2
suma= num1+num2
Mostrar suma//También se puede usar escribir o incluso imprimir
FIN
```

DIAGRAMAS DE FLUJO

Un Diagrama de Flujo representa la esquematización gráfica de un algoritmo, el cual muestra gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema. Se basan en la utilización de diversos símbolos para representar operaciones específicas. Se les llama diagramas de flujo porque los símbolos utilizados se conectan por medio de flechas para indicar la secuencia de operación.

Para hacer comprensibles los diagramas a todas las personas, los símbolos se someten a una normalización; es decir, se hicieron símbolos casi universales, ya que, en un principio cada usuario podría tener sus propios símbolos para representar sus procesos en forma de Diagrama de Flujo. Esto trajo como consecuencia que solo aquel que conocía sus símbolos, los podía interpretar. La simbología utilizada para la elaboración de diagramas de flujo es variable y debe ajustarse a un patrón definido previamente.

DIAGRAMA DE FLUJO PARA SOLUCIONAR PROBLEMAS

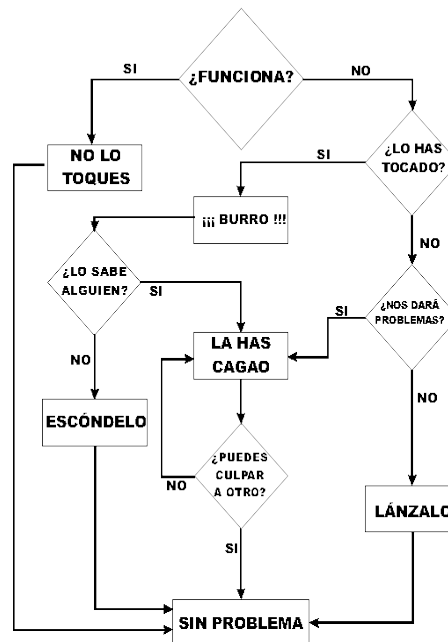
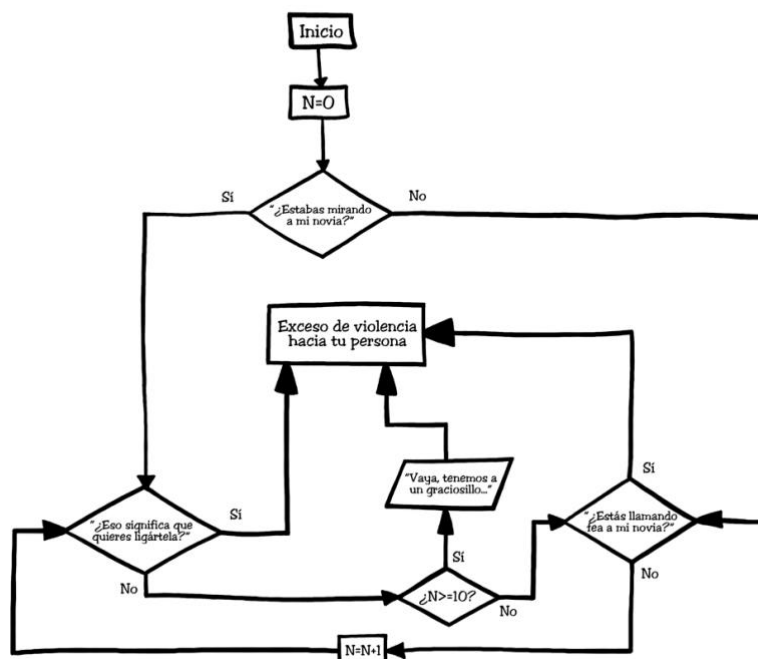


Diagrama de flujo para hacer amigos (by Sheldon Cooper)

<http://www.youtube.com/watch?v=LT6cyqPQq2E>

Por qué tengo miedo a ir a "antros de ligoteo" tales como discotecas, bares musicales, pubs y demás:



Antes de seguir mira el siguiente video

<https://www.youtube.com/watch?v=YTk65pb>

ΛΟΓΑ

2. Estructuras de Selección o condicionales

Una sentencia condicional nos permite seleccionar si una sentencia se ejecutará o no dependiendo de **si** se cumple una condición dada. Como hemos visto en el esquema anterior, existen varios tipos:

2.1. Estructura if

Permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación. Como hemos visto con el pseudocódigo, el *if* corresponde a un *si* y nos permite ver si se cumple una condición. Si esta se cumple, se ejecuta una o varias instrucciones. Si la condición no se cumple, el programa continúa su ejecución normalmente después de la llave de cierre del bloque if.



<p>Pseudocódigo:</p> <pre> Si (condición) { instrucción 1; instrucción 2; ... }</pre>	<p>Diagrama de flujo:</p>
---------------------------------------------------------------------------------------------------	---------------------------

La estructura general en lenguaje Java es:

```
if (condición)
{
    instrucción1;
    instrucción2;
    ...
}
```

La condición debe ser una expresión booleana, es decir, su evaluación **SIEMPRE** debe dar como resultado un tipo boolean (true o false). Veamos un ejemplo concreto:

```
//Ejemplo de programa con if
public class EjemploIf{
    public static void main (String args[ ]){
        int num1 = 9;
        int num2 = 4;
        if (num1 > num2){
            System.out.println (num1+" es mayor que "+num2);
        }
        System.out.println ("No sé nada más y me imprimo siempre porque estoy
        fuera de las llaves del if");
    }
}
```

Se puede poner más de una condición después del *if*, poniendo un “&&”, es decir, que las condiciones tienen que cumplirse todas, o poniendo “||”, es decir, que **solo** una de las condiciones tiene que cumplirse. En este último caso, en cuanto la primera condición se cumple, se ejecuta el código sin necesidad de evaluar la segunda.

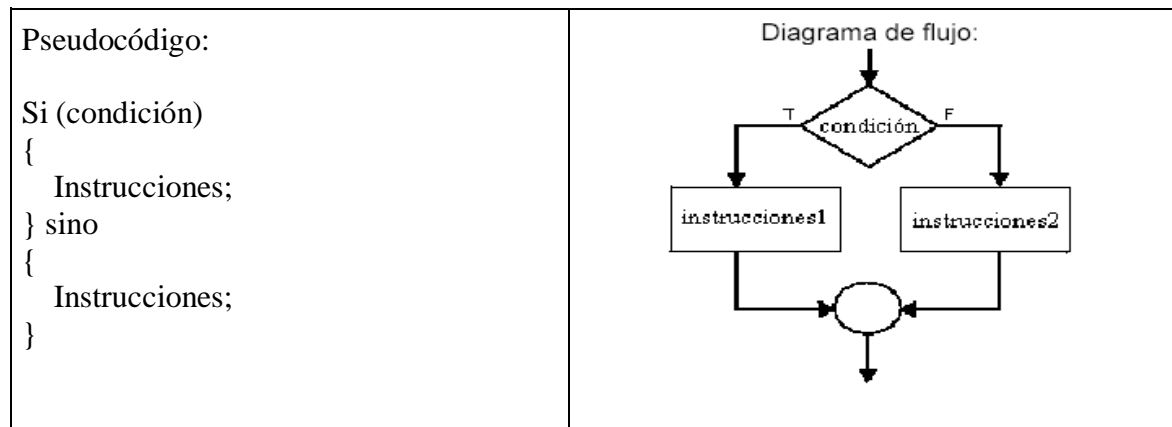
Un ejemplo de un programa que utiliza el *if con doble condición* es el siguiente:

```
//Ejemplo de un programa con un if y dos condiciones

public class EjemploIf{
    public static void main (String args[ ]){
        int num1 = 4;
        int num2 = 9;
        int num3 = 7;
        if (num3 > num1 && num3 < num2) {
            System.out.println (num1+" menor que "+num3+" menor que "+num2);
        }
    }
}
```

2.2. Estructura if-else

Es una ampliación del *if*. Nos permite elegir de entre dos opciones mutuamente excluyentes. En pseudocódigo el *if-else* corresponde a un *si-sino*.



En lenguaje Java quedaría:

```

if (condición)
{
    instrucción1;
    ...
}
else
{
    instrucción2;
    ...
}
        
```



La condición del *si* debe producir un valor booleano; si el valor es verdadero se ejecuta la *instrucción1*, si es falso se ejecuta la instrucción del *else*, es decir, la *instrucción2*.

- ✓ La instrucción del *else* puede también estar vacía, es decir, si el valor booleano de la condición del *if* es falso, no se ejecuta nada y se sigue con el programa.
- ✓ Pueden existir más de una instrucción tanto en el *if* como en el *else*.
- ✓ Es posible anidar sentencias *if*, pero hay que tener cuidado en el momento de emparejar los *else* con sus correspondientes *if*, los *else* se emparejan con el *if* más cercano que no esté asociado ya a un *else*.

Veamos algunos ejemplos concretos:

```
/*Ejemplo de programa con if...else
*/
public class EjemploIfElse{
    public static void main (String args [ ]){
        int num1 = 5;
        int num2 = 21;
        if ( num1 > num2 ){
            System.out.println ("El número mayor es " + num1); // podrían ir más
//sentencias
        }else { //Ojo, aquí en el else, ya no va ninguna condición
//Esta ya no se imprime siempre como en los if simples
            System.out.println("El número mayor es " + num2);
        }
    }
}
```

```
public class EjemploIFAnidados
{
    public static void main (String[] args)
    {
        int edad = 5;
        if (edad >= 18){
            System.out.println ("Tienes 18 o más.");
        }
        else if (edad >= 15){
            System.out.println ("Tienes 15 años o más pero seguro que menos de
18.");
        }
        }else if (edad >= 10){
            System.out.println ("Tienes 10 años o más pero menos de 15.");
        } else {
            System.out.println ("Eres un crío.");
        }
    }
}
```

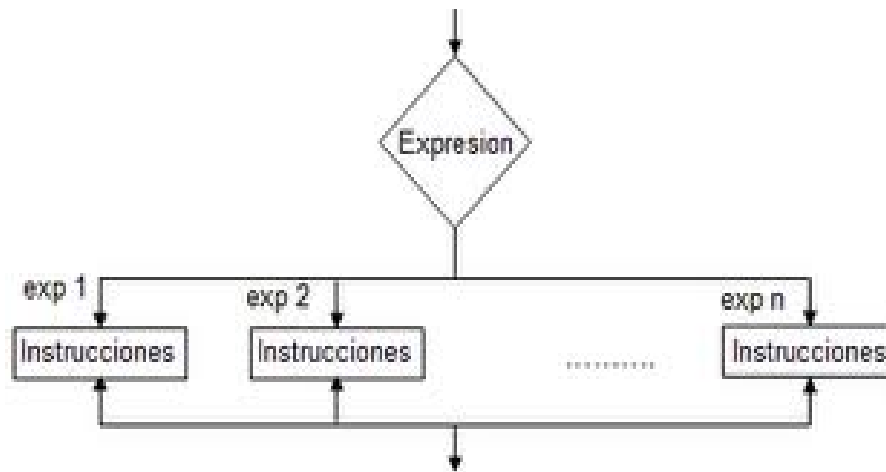

//Ejemplo de programa con if...else anidados

```
public class EjemploIfElseAnidados{  
    public static void main (String args [ ]){  
        int matematicas = 4, lengua =2;  
        if ( matematicas>= 5) {  
            if (lengua>= 5){  
                System.out.println ("Enhorabuena, todo aprobado");  
            }else{  
                System.out.println ("Has aprobado mates pero no  
lengua");  
            }  
        }else {  
            System.out.println ("Has suspendido matemáticas pero, no sé nada  
sobre lengua");  
        }  
    }  
}
```

2.3. Estructura switch

Se utiliza cuando una expresión puede tener varios valores y dependiendo del valor que tome la condición, hay que ejecutar una serie de sentencias. Hay dos posibilidades para programar esto. La primera es usar la estructura switch, la cual deja el código más limpio y fácil de interpretar. La otra opción es usar if else anidados pero el código puede llegar a ser de difícil comprensión. Veamos el uso de la estructura switch:





En lenguaje Java se escribe de la siguiente manera:

```
Switch (expresión) {
    case valor_1:
        sentencia;
        break;
    case valor_2:
        sentencia;
        break;
    case valor_3:
        sentencia;
        break;
    default: [Opcional, pero se suele poner siempre]
        sentencia;
}
```

- La sentencia *switch* compara la expresión encerrada entre paréntesis con cada uno de los casos disponibles.
- La expresión que evalúa *switch* puede ser *int*, *char* o *String*. Ojo con usar *String* porque el valor debe ser exactamente igual, tildes y mayúsculas incluidas.
- Cada caso empieza con la palabra reservada *case* y una etiqueta, seguido del carácter ':' y la o las sentencias que se ejecutarán si el valor de la expresión y la etiqueta coinciden.
- El flujo continuará desde el caso cuya etiqueta coincide con la expresión hasta el final de la sentencia *switch*, a menos que aparezca la sentencia **opcional** "*break*" que termina la ejecución. Ojo, si no se pone el *break*, el programa sigue ejecutando los siguientes *case*.
- Si ninguna de las etiquetas coincide con el valor de la expresión evaluada, no se realiza nada a menos que aparezca la etiqueta opcional "*default*". Esta se ejecuta si ninguno de los casos coincide con el valor de la expresión. Es el adecuado para mostrar un mensaje de error en la introducción del valor del caso que queremos realizar.



Veamos un ejemplo con cadenas, aunque recomiendo que uséis enteros para no meter la pata con las tildes:

Nota: Como se puede ver en el case lunes, podemos escribir varias sentencias en una misma línea, pero no es aconsejable porque hace el código menos claro.

```
public class Ppal {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        String typeOfDay="Sábado";

        switch (typeOfDay) {
            case "Lunes":
                System.out.println("Lunes"); break;

            case "Martes":
                System.out.println("Segundo día de curro");
                break;
            case "Miércoles":
                System.out.println("Media semana ya!!");
                break;
            case "Jueves":
                System.out.println("Bueno, jueves y punto");
                break;
            case "Viernes":
                System.out.println("Ánimo que llega el finde!");
                break;
            case "Sábado":
                System.out.println("Sábado sabadete...");
                break;
            case "Domingo":
                System.out.println("Se acabó loo bueno");
                break;
            default:
                System.out.println("¿No sabes ni en qué día vives?");
        }
    }
}
```

Otro ejemplo usando char como variable:

```

1 public class EjemploSwitch2{
2     public static void main(String args[]){
3         char operador='+';
4         switch(operador){
5             case '-':
6                 System.out.println("El operador es -");
7                 break;
8             case '+':
9                 System.out.println("El operador es +");
10                break;
11             case '*':
12                System.out.println("El operador es *");
13                break;
14             case '/':
15                System.out.println("El operador es /");
16                break;
17             default:
18                System.out.println("Operador desconocido");
19            }
20            System.out.println("FIN DE PROGRAMA");
21        }
22    }

```

Y otro más usando enteros, en este caso, choice se debe haber definido arriba como int:

```

switch(choice) {
case 1:
    printf("Creating...\n");
    break;
case 2:
    printf("Editing...\n");
    break;
case 3:
    printf("Deleting...\n");
    break;
case 4:
    printf("Merging...\n");
    break;
case 5:
    printf("Thank you, Bye.\n");
    break;
default:
    printf("Invalid input!\n");
    break;
}

```




Ojo, el menú de lo que hace cada opción debe ir en sysos antes de que empiece el switch para que el usuario sepa qué opción debe pulsar.



Haz tú un ejemplo de algún menú, por ejemplo, de un móvil, usando int como tipo de variable para los case.

3. Estructuras de repetición.

Las sentencias iterativas son repeticiones de un bloque de sentencias un número determinado o indeterminado de veces. En JAVA las sentencias iterativas son las siguientes:

 while
 do...while
 for



Debemos ser muy cuidadosos a la hora de usar los bucles puesto que es bastante común cometer errores a la hora de diseñar las condiciones de salida. Un caso muy común es el de los bucles infinitos. Veamos un ejemplo muy claro:

<http://www.youtube.com/watch?v=cMyqv5VeHZ8>

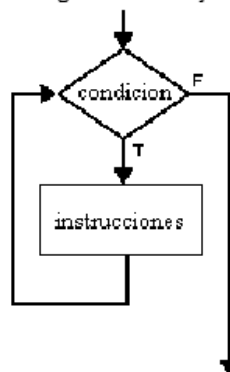
3.1. Estructura while

La sentencia *while* nos permite repetir una sentencia o más, un número determinado o indeterminado de veces mientras una expresión sea cierta.

Pseudocódigo:

```
Mientras (condición)
{
    Instrucciones;
    ...
}
```

Diagrama de flujo:



while (condición) {

Instrucción 1;
Instrucción 2;

...

}

- La condición, como ya sabemos, debe ser una expresión cuyo resultado sea de tipo booleano.
- **Una de las instrucciones que forman parte del bucle debe modificar algunas de las variables de la condición con la finalidad de que la expresión se haga falsa y poder salir del bucle.** Si esto no ocurre el programa entrará en un bucle infinito.
- Si la condición resulta falsa desde el principio, el programa no ejecutará el bucle y seguirá la ejecución con la línea después del bucle, es decir, se ejecutará la siguiente línea que haya después de la llave de cierre.

//Ejemplo con while

```
public class EjemploWhile{
    public static void main(String[ ] args) {
        int numero=1;
        while (numero<=10){
            System.out.println (numero);
            numero ++;
        }
    }
}
```

Este ejemplo, muestra en pantalla los números del 1 al 10. ¿Qué pasaría si la sentencia `numero ++;` estuviera escrita antes del `println ()`? ¿Y si ni siquiera estuviera?

3.2. Estructura `do...while`

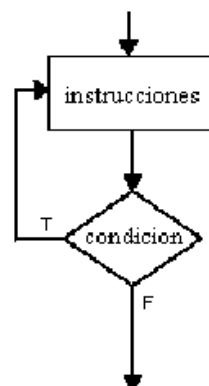
Esta sentencia, a diferencias del *for* (que veremos justo después) y del *while*, evalúa la expresión al final del bucle, por lo tanto el cuerpo del bucle se ejecuta, como mínimo una vez antes de evaluar la expresión. De nuevo, esta expresión debe tener un resultado booleano.

Pseudocódigo:

Hacer

```
{
    instrucciones;
    ...
```

```
}mientras (condición); //Ojo al punto y coma
```



Este bucle es recomendable a la hora de realizar menús para el usuario porque al menos el menú se mostrará una vez al de comienzo del programa y aunque al usuario no le interese ninguna opción habrá visto el menú.

//Ejemplo con do...while

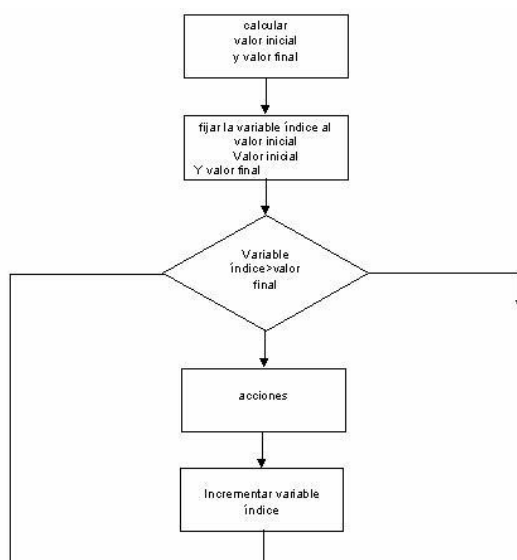
```
public class EjemploDoWhile{
    public static void main(String[] args) {
        int numero=1;
        do {
            System.out.println (numero);
            numero ++;
        } while (numero < =10);
    }
}
```

3.3 Estructura for

Esta estructura de iteración se utiliza cuando ya conocemos el número de veces que se debe repetir el cuerpo del bucle. Se puede conseguir realizar un *for* mediante un bucle *while*, pero el *for* es el bucle más usado de los tres que hemos visto.

Pseudocódigo:

```
Para (Variable de control=Valor de inicio; hasta  
CondiciónDeContinuación;Expresión Incremento)  
{  
    Instrucciones;  
    ...  
}
```



En Java queda:

```
for (inicialización; expbooleana; incremento) {  
    instrucción1;  
    instrucción2;  
}
```

donde:

- **Inicialización** es una sentencia de asignación de una o más variables.
- **expbooleana** es una expresión que debe dar como resultado verdadero o falso. Es la condición que se debe cumplir para continuar ejecutando el bucle.
- **Incremento** es una asignación que se encarga de modificar la variable que se utiliza para “contar” el número de veces que se ha ejecutado el bucle. Normalmente se usa la letra **i** como contador o nombre de dicha variable.
- Cualquiera de las tres expresiones se puede omitir, o incluso las tres, pero no se pueden omitir los separadores (“;”). *¿Qué hará el bucle si se omite una, dos o todas las expresiones del paréntesis? Lo dejo para vosotros.*
- Son muy utilizados para la gestión de arrays.
- La variable que más se suele usar para los bucles for como “contador” es la **i** o si el bucle es doble la **j** (uno de los pocos casos en los que poner solo una letra como nombre de una variable es aceptado a nivel interplanetario).

Veamos un ejemplo:

//Ejemplo con for

```
public class EjemploPara{  
    public static void main(String[ ] args) {  
        //No se suele declarar el contador fuera  
        int tope=10;  
        for (int numero =0; numero < tope; numero ++){  
            System.out.println (numero);  
            //Ya no es necesaria la sentencia numero ++  
        }  
    }  
}
```


4. Arrays.

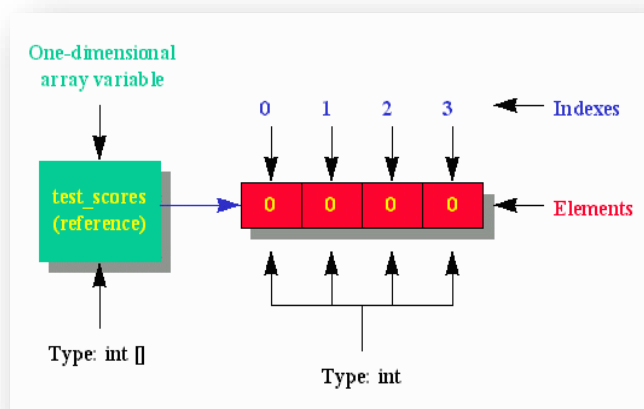
Hasta ahora, todos los ejercicios que hemos hecho no tenían la necesidad de almacenar un gran número de valores o datos, por lo que utilizábamos variables como edad, cantidad, número, etc. Vamos a dar un paso más y vamos a suponer que se nos pide realizar un programa que almacene la temperatura en 100 ciudades y nos muestre la media mundial. ¿Definirías 100 variables? ¿Qué pasaría si a la semana nos dicen que sean 200 ciudades en lugar de 100?



Para solucionar este problema, comenzaremos a utilizar los arrays, también llamados vectores (o arreglos en Sudamérica).

Un array es una estructura homogénea, compuesta por varios elementos **TODOS DEL MISMO TIPO** que están almacenados en la memoria de forma consecutiva. Más adelante veremos que los arrays no solo pueden almacenar tipos básicos, como int, double, etc. sino que también pueden almacenar objetos.

El array nos permite obtener un valor, dando una posición en una tabla. Es muy veloz ya que esta estructura tiene un fuerte paralelo con cómo se organiza internamente la memoria de un ordenador. Los valores en un array están en orden, y en ese mismo orden están dispuestos en la memoria.



Un array se puede declarar, inicializar, cargar, etc.

4.1. Declaración, instanciación e inicialización.

¿Cómo se declara un array?

Hay dos formas diferentes igual de válidas:

tipo [] identificador;
tipo identificador [];

donde **tipo** representa el tipo de dato primitivo o el tipo de objeto correspondiente a los valores almacenados en el array e identificador es el nombre que le damos.

Ejemplo:

```
int [ ] cantidad;  
double sueldo [ ];
```

NOTA: Cuando declaramos un array, el compilador y la máquina Virtual de Java (JVM) no saben qué longitud tendrá porque ha declarado variables de referencia que en ese momento NO señalan a ningún objeto. *Mala idea no asignar tamaño.*

¿Cómo se instancia un array unidimensional?

Aunque todavía no hemos hablado de objetos, para poder utilizar y dar valores (inicializar) un array, es preciso instanciar un objeto Array lo suficientemente grande como para dar cabida a todos los valores del array. Un Array se instancia definiendo el número de elementos que contiene. La sintaxis es la siguiente:

```
identificador = new tipo [longitud];
```

donde **longitud** representa el tamaño (número de elementos) del array, **new** es un operador que se estudiará más adelante y **tipo**, el tipo de datos que vamos a guardar y que especificamos en la declaración.

Lo más común es hacer las dos operaciones anteriores (declarar e instanciar) en la misma línea de código. Veamos:

```
int [ ] edad= new int [5];
```

NOTA: Cuando se crea Array, cada elemento primitivo se inicializa con el valor cero correspondiente al tipo especificado (cero para enteros, \0 para char, carácter vacío según Unicode) y **null** para Arrays de objetos.

¿Cómo se inicializa un array unidimensional?

Es posible escribir el contenido de un array después de crearlo. Existen varias posibilidades, aunque de momento solo veremos una:

- Asignando directamente el valor:
-

```
identificador [índice] = valor;
```

Por ejemplo:

edad [0] = 19;//Se guarda el valor 19 en la primera posición (posición cero) del array edad.

edad [1] = 22;//Se guarda el valor 22 en la segunda posición (posición 1) del array edad.

edad [2] = 36;//Se guarda el valor 36 en la tercera posición (posición o índice 2) del array edad.

i=0	i=1	i=2	i=3	i=4
19	22	36	0	0

NOTA: Recuerda que el primer elemento de un array se sitúa en el índice cero y el último elemento se sitúa en el lugar correspondiente al valor de **longitud** menos uno.

Declaración, instanciación e inicialización de arrays unidimensionales

Si se conocen los valores que se van a incluir en el array en el momento de declararlo, se puede declarar, inicializar y definir los valores del objeto Array en la misma línea de código.

Un ejemplo sería:

```
int [ ] edad = {19, 42, 92, 33, 46};
```

En este caso, el array tendrá una dimensión de 5 elementos y no es necesario especificarlo en ningún otro sitio.

NOTA: OJO, no es posible declarar e inicializar un array en líneas diferentes utilizando la lista de valores separados por coma. Por ejemplo, el siguiente código daría un error:

```
int [ ] edad;  
edad = {19, 42, 92, 33, 46};
```



4.2. Utilización de arrays.

Acceso a un valor dentro de un array

El acceso a cada elemento de un array se realiza a través de su índice. Para acceder a un valor de array, se indica el nombre del array y el número del índice del elemento (entre corchetes []) a la derecha de un operador de asignación.

Por ejemplo:

Para establecer el valor de un determinado índice:

```
edad [ 1 ] =19;
```

Para recuperar los valores de un determinado índice del array:

```
int años= edad [1];
```

*//Esta sentencia guarda en la variable **años** el número almacenado en la posición 1 del array edad, es decir, el valor 19.*

Establecimiento de valores de arrays utilizando el atributo length y un bucle

- **Atributo length**

Todos los objetos Array tienen una variable de atributo llamada **length** que contiene la longitud del Array. El número de elementos de un array se almacenan como parte del objeto array. El software de la Máquina Virtual de Java utiliza la longitud para asegurarse de que cada acceso al array se corresponda con un elemento actual del array.

Si se produce un intento de acceder a un elemento que no existe, por ejemplo, especificando en el código un valor edad [21] para un array edad que tiene una longitud [10], se producirá un error.

- **Establecimiento de los valores del array utilizando un bucle**

Es posible y conveniente, usar el atributo length de la clase Array en la parte condicional de un bucle para iterar (recorrer) en un array. Por ejemplo:

//En este ejemplo, cada índice de miArray se llena con el número 8

```
int [ ] miArray;  
miArray = new int [100];  
for (int contador=0; contador <miArray.length; contador ++){  
    miArray [contador] =8;  
}
```

Para imprimir los valores por pantalla (ojo, se usa otro nombre para la variable que cuenta para que veamos que se puede usar el nombre que queramos, el contador muere con la llave final del bucle:

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

- **Bucle for mejorado.**

A partir de la versión Java 5, existe otra forma de la sentencia “for” diseñada para iterar en los arrays. Puede usarse para hacer los bucles más compactos y fáciles de leer (son más usados con colecciones, que veremos más adelante). Se recorre el array completo hasta el final.

```
for(declaracion : expresion){  
  
}
```

Las dos partes de la declaración son:

- **declaracion:** es una declaración de una variable, de tipo compatible con los elementos del array al que se está accediendo. Esta variable estará disponible dentro del bloque, y su valor será el mismo que el elemento actual del array.
- **expresion:** debe evaluar el array que quieres recorrer. Podría ser el nombre del array, o un método que devuelve un array. El array puede ser de cualquier tipo, objetos, primitivos o arrays de arrays.

Veamos un ejemplo:

```
classEjemploForMejorado {
    public static void main (String [ ] args) {
        int [ ] numeros = {1,3,5,7,9,11,13,15,17,19};
        int suma = 0;
        for (int valor : numeros ) {
            suma =suma + valor;
            System.out.println (valor); //Para ver el valor del elemento
        }
        System.out.println ("La suma es: " + suma);
    }
}
```

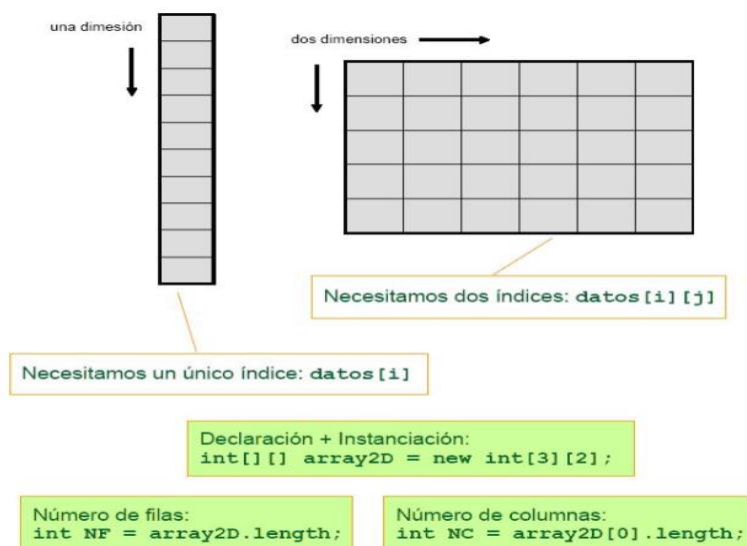
//valor se refiere al valor guardado en el array y no al índice como en los bucles normales.

//numeros es el nombre del array

//Además se asume que el array se recorrerá entero, salvo salida forzosa del bucle.

4.3. Arrays multidimensionales o matrices.

Tratar con arrays de varias dimensiones es bastante parecido a tratar con vectores. Aunque, en principio, se pueden definir arrays de todas las dimensiones que queramos, a partir de 3 o más no se usan. Por lo tanto, no veremos arrays de más de dos dimensiones. A estos últimos se les llama arrays bidimensionales.



Un ejemplo concreto:

```
int [ ] [ ] mitabla = new int [5][8];
```

Esta matriz tiene 5 filas y 8 columnas.

La inicialización en el momento de declarar un array bidimensional sería:

```
int [ ] [ ] mitabla = {{1,4,5},{6,2,5}}; //Tendrá 2 filas (cada llave interior es una fila y 3 columnas, pues cada llave interior tiene 3 elementos.
```

El acceso a la matriz se hace igual que cuando trabajamos con unidimensionales pero usando dos bucles for. Ejemplo, para rellenar o cargar:

```
int [ ] [ ] mitabla = new int [5][8];

for (int i=0; i<mitabla.length; i++){

    for (int j=0; j<mitabla[i].length; j++){

        mitabla [i][j]= i+j;

    }

}
```

¿Qué hace este trozo de código?

Hay multitud de ejemplos e información en internet sobre los arrays. Además, el API de Java proporciona clases para trabajar con ellos. Investigad un poco.

4.4. Cadenas de caracteres.

En Java, las cadenas de caracteres son un array cuyos elementos son del tipo *char*. Estas se tratan como objetos de la clase *String*.

Por ejemplo:

```
char [ ] nombre1= {'p', 'e', 'p', 'e'};
char [ ] nombre2= {112, 101, 112, 101};
char [ ] nombre3= new char [4];
String nombre4= "Pepe";
```

En el código anterior, las variables *nombre1* y *nombre2* contienen exactamente lo mismo dado que internamente Java almacena los caracteres con sus símbolos ASCII correspondientes (a la 'p' le corresponde el 112 y a la 'e' el 101). La variable *nombre3* se ha creado como una cadena pero todavía no se ha inicializado, por lo tanto, sus 4 posiciones contendrán el valor '\0'. La variable *nombre4* es un objeto del tipo *String*.

NOTA: La clase String y otras, como StringBuffer, son clases que ayudan y ofrecen herramientas para trabajar de forma más ágil con cadenas. Las estudiaremos con profundidad en el primer tema dedicado a objetos, cuando hablemos del API de Java.

