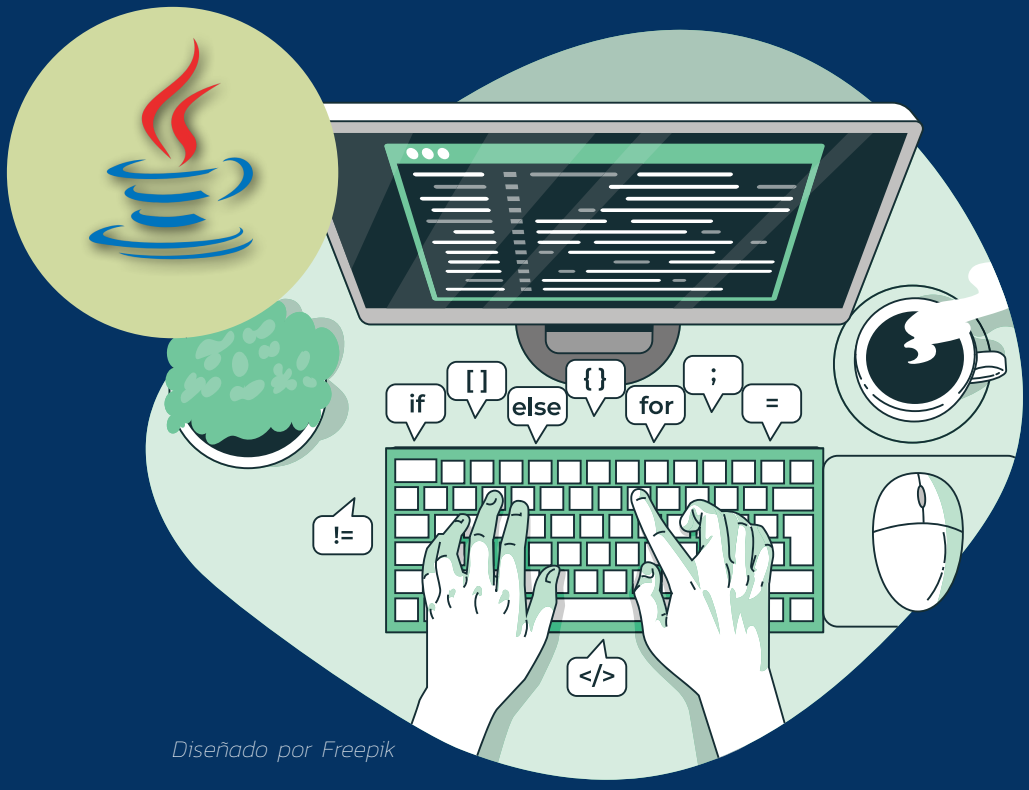
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

ETAPA 3: ESPECIALIZACIONES

JAVA



Diseñado por Freepik

Apunte N° 2

PROGRAMACIÓN ESTRUCTURADA

2

Año 2025

Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

¿Qué es la programación estructurada?

La programación estructurada es un paradigma de programación que se basa en el uso de subrutinas o funciones y tres estructuras de control para crear programas de computadora: Estructura secuencial, Estructura condicional, Estructura iterativa con condición.

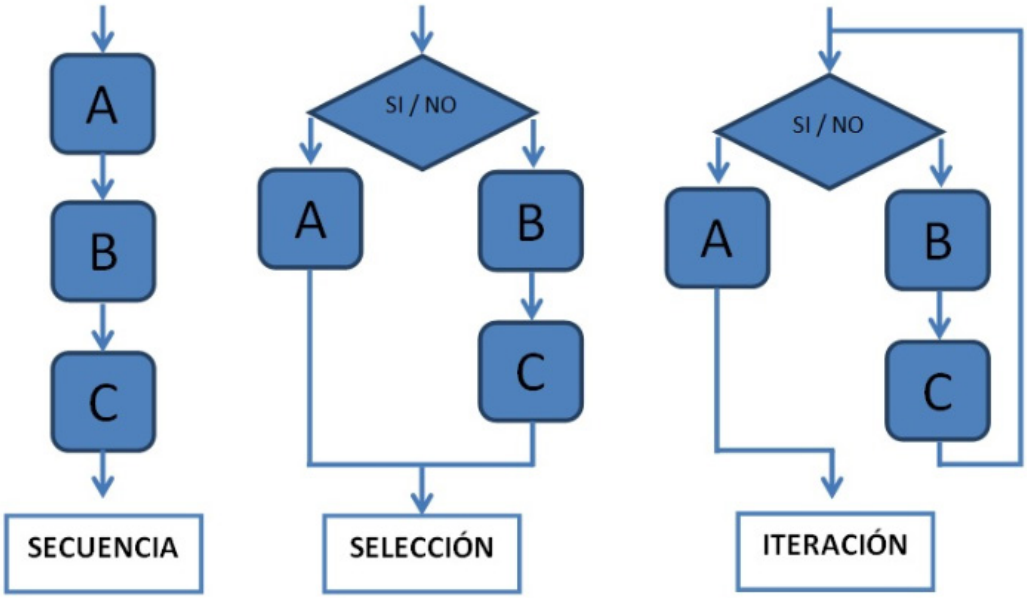
En la programación estructurada se considera innecesario y contraproducente el uso de la transferencia incondicional (GOTO). Esta instrucción suele generar código espagueti, que es más difícil de seguir y de mantener, además de originar numerosos errores de programación.

El teorema de la programación estructurada afirma que cualquier programa puede construirse mediante estas tres estructuras básicas.

El teorema del programa estructurado es un resultado de la teoría de lenguajes de programación. Establece que toda función computa ble puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- **Secuencia:** ejecución de una instrucción tras otra.
- **Selección:** ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- **Iteración:** ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

Este teorema demuestra que la instrucción GOTO no es estrictamente necesaria y que para todo programa que la utilice existe otro equivalente que no hace uso de dicha instrucción.



Ejemplo en código :

```
int contador = 10;

do {
    if (contador != 0 ){
        System.out.println("Hola soy el numero : " + contador);
        contador = contador - 1;
    }
}while (contador > 0 );

if (contador == 0 ){
  System.out.println("Hola esta es una sentencia " + contador);
}else{
  System.out.println("No deberia pasar por aqui… ");
}
```

Estructuras de control

Condicional simple

```
if( condicion )
   {
     accion1;
   }
```

La estructura de control condicional simple, se trata de aquella estructura de control que **analiza un la condición, el cual devuelve como resultado un booleano, es decir, verdadero o falso. Si la condición es verdadera entonces ejecuta las acciones del bloque, sino no ejecuta nada del bloque y sigue el curso del programa**.

Ejemplo en código :

```
int resto = 10 % 2;
if( resto == 0 )
{
System.out.println(10+" es Par");
}
```

Condicional doble

```
if( condicion )
   {
     accion1;
   }
else {
     accion2;
   }
```

La estructura de control condicional doble, nos dice que **en caso de ser verdadera la condición ejecuta el bloque principal y si no ejecuta el segundo bloque de código** (entra por else).

Ejemplo en código :

```
int edad = 18
if( edad >= 18 )
{
System.out.println("Ud. es mayor de edad !");
}
else
{
System.out.println("Ud. es es menor de edad");
}
```

Condicional múltiple if-else if

La estructura de **control múltiple**, es aquella la cual evalúa la primer condición **if,** en caso de ser verdadera ejecuta el primer bloque de código, en caso de ser falso, evaluará la próxima condición **else if** y si es verdadera ejecuta el bloque de código correspondiente, en caso de ser falsa evalúa las demás condiciones (else if) si que existen. Puede darse el caso en que por última instancia aparezca un **else**, en caso de que las anteriores else if no cumplen con la condición entrara por este bloque de código.

SINTETIZANDO: CÓMO FUNCIONA

1. **Se comprueba la condición de la sentencia If.**
2. **Si es falsa, se evalúa la sentencia Elif.**
3. **Si la condición también es falsa, se evalúa la sentencia Else.**

Ejemplo en código :

```
int nota = 10
if( nota == 10 )
{
System.out.println("Su calificación es 10");
}
else if( nota == 9 )
{
System.out.println("Su calificación es 9");
}
else if( nota == 8 )
{
System.out.println("Su calificación es 8");
}
else if( nota == 7 )
{
System.out.println("Su calificación es 7");
}
else if( nota == 6 )
{
System.out.println("Su calificación es 6");
}
else if( nota == 5 )
{
System.out.println("Su calificación es 5");
}
else
{
System.out.println("Su calificación es menor a 5");
}
```

Condicional múltiple switch case

La estructura de control múltiple a partir de **switch case**, es una estructura el cual **evalúa una sola vez la condición, en base a su valor ejecuta un bloque de código (valor1, valor2, etc.)**

```
switch( valorValido )
{
case valor1:
       accionA;
       accionB;
            :
       break;
case valor2:
      accionX;
      accionY;
          :
      break;
   :
default: masAcciones;
}
```

El **break**; es un operador que **rompe con la secuencia de ejecución de acciones**, se ejecutaran las acciones hasta que encuentra un break. El motivo por el cual es necesario esta sentencia, es por el comportamiento que tiene el switch, el cual ejecuta las líneas de acciones (incluso las de otras case) una vez un case se cumpla.

Ejemplo en código :

```
switch( 2 )
{
case 1:
       accionA;
       accionB;
            :
       break;
case 2:
      accionX;
      accionY;
          :
case 3:
      accionX;
      accionY;
       break;
default: masAcciones;
}
```

imagen 9

Notemos que en este caso *(imagen 9)*, entra por el case 2, ejecutara entonces accionX, accionY y demás acciones en caso de tenerlas. Pero a falta del break; la ejecución no se detiene y ejecutará también las acciones del case 3, en donde allí sí tenemos un break; lo que va a hacer parar con la ejecución de sentencias.

El **default** es necesario en switch y siempre se lo deja como última opción en la estructura de control switch. La misma se ejecuta si y sólo si en los N case (condiciones) anteriores no se cumplieron.

Valores válidos en el switch case

TIPOS DE VALORES VÁLIDOS PARA SWITCH
byte, short, int, char
Byte, Short, Integer, Character
String
enum

Condicional in-line u operador ternario

El operador ternario es aquel que tiene la siguiente sintaxis:

```
( condición ) ? acción por si es verdadero : acción por si es falso
```

Se evalúa la condición, en caso de ser verdadero se devuelve el valor indicado después del operador **?** y sino se devuelve el valor indicado después de:

Ejemplo en código :

```
a>b ? "a es Mayor" : "a es Menor"
```

(Como devuelve siempre un valor, nada nos detiene a que dicho valor pueda ser almacenado en una variable).

```
int resultado = ( a > b ) ? 1: 0
```

? Cuestionate lo siguiente: Es el operador ternario lo mismo que el condicional doble?

Estructuras de control repetitivas

Estructura iterativa while

```
while( condicion )
{
accion1;
accion2;
:
}
```

El ciclo itera mientras la condición resulta verdadera. La condición **debe devolver un valor booleano**. Esta estructura también se denomina **pre-test, porque primero se realiza la prueba de la condición y luego (si la misma es verdadera) ejecuta las acciones (mientras se mantenga la condición verdadera)**. El control que debemos de tener en cuenta aquí es que la condición cambie de estado en algún momento, de tal manera que la misma no quede iterando infinitamente.

Ejemplo en código :

```
int i = 1;
while( i <= n )
{
// muestro el valor de i
System.out.println(i);
// incremento el valor de i
i++;
}
```

Estructura iterativa do-while

Al igual que la estructura iterativa, la misma itera mientras la condición sea verdadera. **La condición debe devolver un valor booleano.**

```
do
{
accion1;
accion2;
:
}
while( condicion )
```

Esta estructura también se denomina **post-test, porque ejecuta mínimamente al menos una vez la secuencia de acciones encerrada en el bloque do, posterior a ello recién evalúa la condición.**
Al igual que la estructura pre-test, debemos prestar atención en que la condición tome el valor del falso tal que rompa con el ciclo de ejecuciones en algún momento.

Ejemplo en código :

```
int i = 1;
do
{
System.out.println(i);
i++;
}
while( i <= n )
```

Estructura iterativa for

También denominado bucle for, o estructura repetitiva manejada por contador. Es aquella estructura la cual **ejecuta "N" veces una secuencia de acciones.**

```
for( inicializacion; condicion; incremento/decremento )
{
accion1;
accion2;
:
}
```

Sin embargo el bucle for, **tiene una sección de condición en la cual, si ésta devuelve falso la iteración termina. Lo común es que esta condición la determine el índice inicializado.**

Ejemplo en código :

```
for( int i=1; i<=n; i++ )
{
System.out.println(i);
}
```

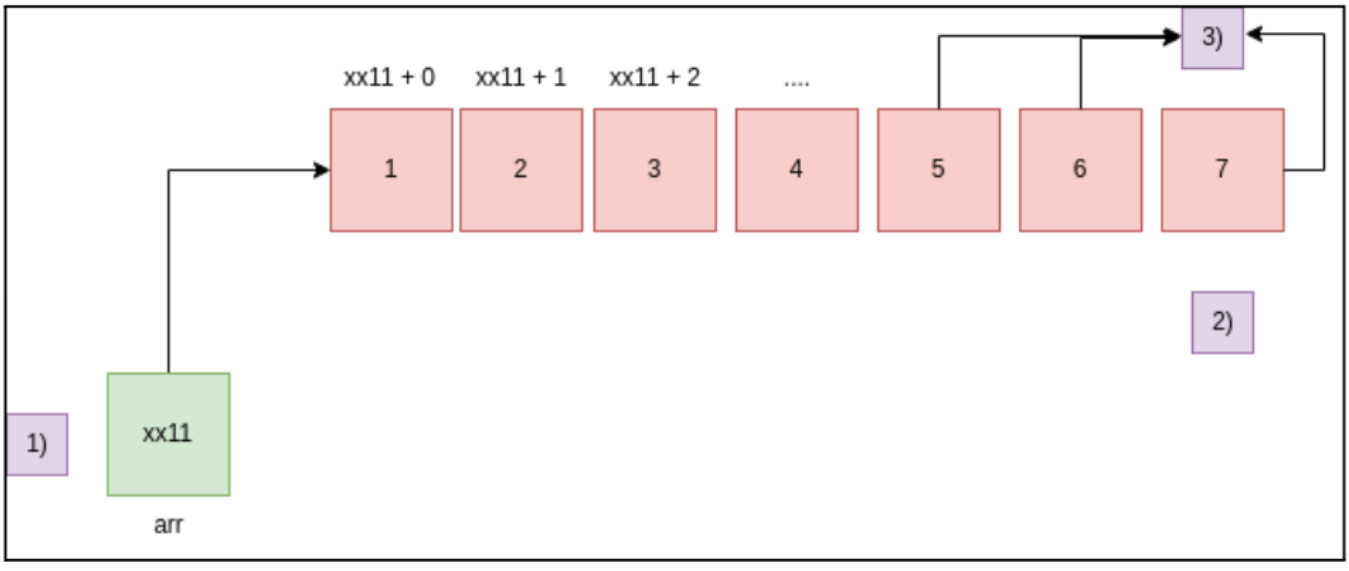
Arreglos

¿Qué es un arreglo?

Es una estructura de datos que permite almacenar un conjunto de elementos del mismo tipo en una misma variable.
Los elementos en un arreglo son almacenados en posiciones contiguas de memoria y se acceden a ellos mediante un índice.
Los arreglos tienen un tamaño fijo, que es determinado en el momento de su creación, y pueden contener cualquier tipo de dato primitivo o una referencia a un objeto.
Los arreglos son muy útiles para almacenar y manipular grandes cantidades de datos de manera eficiente en un programa.

Pasos comunes para instanciar un arreglo

1. **Declaración** : Crear una referencia al array, es decir, es crear el punto de memoria que nos dará acceso al primer encuentro del array.
2. **Instanciación de un array** : Es el paso de crear la estructura de datos como tal.
3. **Inicialización** : Asignar valores a las posiciones de array.



En Java los arrays comienzan siempre desde cero.
Ejemplo en código :

```
// dene un array de 10 elementos enteros numerados de 0 a 9
int arr[] = new int[10];
```

También podemos construir un array de n elementos, siendo n una variable. int n = 10;
Ejemplo en código :

```
int arr[] = new int[n];
```

La declaración no es lo mismo que la instanciación, podemos declarar un arreglo lo cual implica simplemente indicar en una variable el objeto (arreglo) y el tipo que vamos a guardar, sin indicar las dimensiones del mismo.
Ejemplo en código :

```
// declaramos un arreglo de Strings (aun sin dimensionar)
String arr[];
```

La **instanciación** es la aplicación del operador new junto con indicar la dimensión del arreglo.
Ejemplo en código :

```
// instanciamos el array
arr = new String[10];
```

Adicional a esta aclaración, también podemos remarcar la posibilidad de poder **declarar e instanciar al mismo tiempo**.
Ejemplo en código :

```
// declaramos e instanciamos el array de 10 Strings
String arr[] = new String[10]
```

La inicialización no es más que la acción de asignar a cada elemento del arreglo un valor, si no se inicializa un arreglo estos toman valores por defecto:

1. Los tipos primitivos se toma un valor con 0 (int, long, float, double, char, short, byte) o false (que es el equivalente en el tipo boolean).
2. Los objetos (o mejor dicho, los punteros a objetos) apuntan a null, que es el equivalente a no apuntar a nada. Este caso aplica a todo lo que extienda a Object, incluyendo los «wrappers» de los tipos primitivos (Integer, Char, Short...) y a la clase String, por ejemplo.

Una manera de declarar, instanciar e inicializar es utilizando el operador "{ }", el cual define estructuras, define cantidad de elementos y a su vez valores.
Ejemplo en código :

```
int mat[][] = { {3, 2, 1 }
              ,{5, 3, 7 }
              ,{1, 9, 2 }
              ,{4, 6, 5 } };
```

Ejemplo en código :

```
int mat[] = {3, 2, 1 };
```

Operadores utilizados

Operador Instanciación New

Este operador es el que nos devuelve la referencia a memoria del primer espacio reservado para el arreglo, la cual comúnmente la guardamos en una variable.

Operador de acceso [ ]

Este operador es el que nos devuelve la referencia a memoria del primer espacio reservado para el arreglo, la cual comúnmente la guardamos en una variable.

Ejemplo en código :

```
// asigno el numero 123 en la posición 5 del array arr
arr[5] = 123;
```

Ejemplo en código :

```
// Mostramos por pantalla elemento
System.out.println(arr[5]);
```

Ejemplo en código :

```
// producto de dos elementos del arreglo
System.out.println(arr[5] * arr[3]);
```

Ejemplo en código :

```
// Recorrido del arreglo en un bucle for
for( int i=1; i<=arr.length; i++ )
{
System.out.println(arr[i]);
}
```

Lo más común es recorrer siempre el arreglo manejando un bucle for, ya que conocemos exactamente cuántas veces tenemos que iterar, por cada iteración podemos utilizar el índice que forma el bucle para acceder al arreglo.

Atributo length

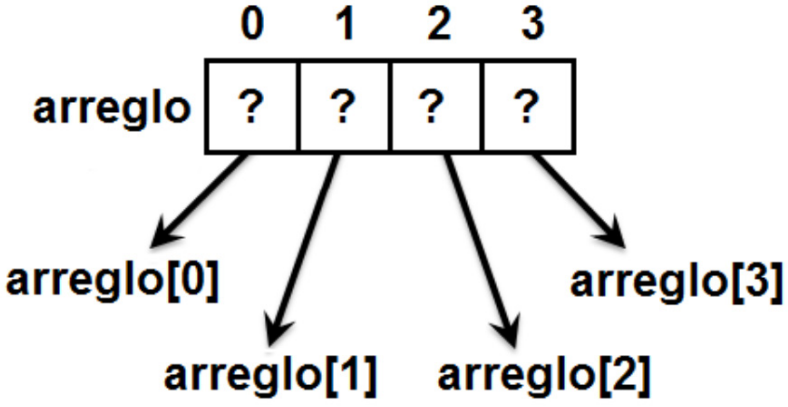
Como el **arreglo** es un **objeto**, el **length** es un **atributo** que nos permite **conocer el tamaño de la estructura de datos**.

Dimensiones de un arreglo

Un arreglo puede tener hasta N cantidad de dimensiones, teniendo en cuenta que al aumentar las dimensiones, es mayor la complejidad de tratamiento de los mismos.

Dimensión 1 o arreglo lineal

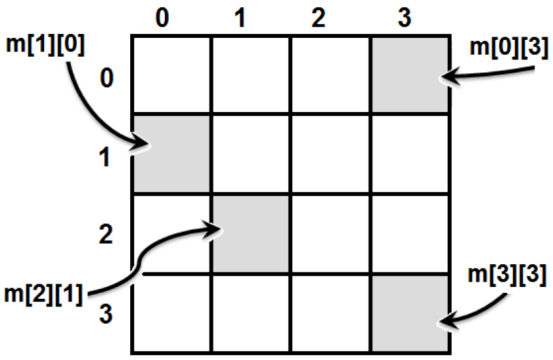
new int[4]



Dimensión 2 o matriz

new String[4][4]

Una matriz es un arreglo de dos dimensiones, pero también podríamos decir que una matriz es un arreglo de arreglos. Viéndolo de esta manera entonces podemos conocer la cantidad de las y columnas de una matriz a través del atributo length.



Ejemplo en código :

```
// creo una matriz de n las x m columnas
int mat[][]=new int[n][m];
int nro;
for(int i=0; i<n; i++ )
{
  for(int j=0; j<m; j++)
    {
       // genero un numero aleatorio entre 0 y 1000
       nro=(int)(Math.random()*1000);
       // asigno el numero en la matriz
       mat[i][j]=nro;
    }
}
for(int i=0; i<n; i++ )
{
  for(int j=0; j<m; j++)
    {
      // imprimo la celda de la matriz
      System.out.print(mat[i][j]+"\t");
    }
   System.out.println();
}
```

Ejemplo en código :

```
int [][]mat = new int[5][3];
int las = mat.length; // cantidad de las
int colums = mat[0].length; // cantidad de columnas
```

Dimensión 3 o planos

new String[4][4][4]