**¿Qué es Java?**

**Java** es un lenguaje de programación pensado para desarrollo de aplicaciones de gran envergadura, altamente escalables, de gran integración con otras tecnologías y muy robustas.

Java es uno de los lenguajes más utilizados a nivel mundial, sobre todo, para la programación backend y como base para la programación mobile.

Características:

* Es un lenguaje orientado a objetos
* Es multiplataforma.
* Maneja la memoria de manera automática
* Evolución permanente
* Gran demanda en el mercado

**Versiones de Java**

**J2SE (Java2 Standard Edition)**: Tiene como objetivo el desarrollo de aplicaciones de escritorio. Incluye la funcionalidad básica del lenguaje como manejo de clases, colecciones, entrada/salida, acceso a base de datos, manejos de sockets, hilos de ejecución, etc.

**J2EE (Java2 Enterprise Edition)**: Tiene como objetivo el desarrollo de aplicaciones empresariales, de gran envergadura. Contempla ambientes Web, como las ambientales manejados por servidores de aplicación. Las tecnologías principales incluidas en esta área son Servlets, JSP y EJB, frameworks, entre otras.

**¿Qué necesitamos para programar en Java?**

* Un **IDE** (En este caso utilizaremos el ***IDE Netbeans 19***).
* **JRE (Java Runtime Environment)**: Es un conjunto de utilidades que permite la ejecución de programas Java.
* **JDK (Java Development Kit)**: Es un software que provee herramientas de desarrollo para la creación de programas en Java.

## Tipos de datos y Variables en Java

**Java** se destaca por ser un lenguaje **fuertemente tipado**, es decir, que es **muy exigente** tanto con su **sintaxis** como con sus **tipos de datos**.

Entre estas “exigencias” se destacan:

* Es **Case sensitive** (distingue mayúsculas de minúsculas).
* Cada línea de código debe finalizar con **;** (punto y coma).
* El principio y fin de cada estructura, así como los bloques de código se delimitan con **{ }** (llaves)

Los **tipos de datos**, por su parte, definen qué puede ser almacenado dentro de una **variable** y los límites de lo que allí se guarda.

**Variables y constantes**

Una variable es un nombre que se asocia con una porción de la memoria de la computadora, en la que se guarda un valor determinado. Un claro ejemplo del uso de variables es en matemática, donde un valor “x” puede tener infinita cantidad de valores asignados, por ejemplo, x = 5.

Las constantes son valores que se mantienen siempre de igual manera y que no dependen de una asignación para obtener el mismo. Por ejemplo, el valor del IVA (21%).

**Tipos de datos**

Los tipos de datos definen que puede ser almacenado dentro de una variable y los límites de lo que allí se almacena. Por ejemplo, si quiero que una variable solo almacene números enteros.

Algunos de los tipos de datos más usados son:

Enteros > **int** > Ejemplo: 0, 1, 35, 120, 44, etc.

Decimales > **double** > Ejemplo: 1,5 ; 5,8 ; 35,64 ; etc.

Booleanos > **boolean** > Ejemplo: true o false

Caracteres > **char** > Ejemplo: ‘a’, ‘b’, ‘l’, ‘d’, ‘e’, ‘p’

Cadena de Caracteres/Texto > **string** > Ejemplo: “Hola mundo, probando 1, 2, 3”

Entero Largo > **long** > Números entre: (-9.223.372.036.854.775.808 y 9.223.372.036.854.775.807)

**int numero = 5;**

**String nombre = "Juan";**

**char letra = 'A';**

**double temp = 32.33;**

**boolean bandera = true;**

**long largo = 123456789;**

El lenguaje Java da de base una serie de tipos de datos primitivos: byte, short, int, long, float, double, boolean, char. Es importante saber que estos son tipos de datos del lenguaje y que no representan objetos. Cosa que sí sucede con el resto de elementos del lenguaje Java.

**Variables: Operadores aritméticos**

**Un operador** es un **símbolo especial** que indica al compilador que se debe efectuar una determinada operación. Estas operaciones pueden ser de asignación, aritméticas, condicionales, relacionales, entre otras. Algunas de las más conocidas son:



## Condicional Simple: IF

La estructura de control **IF** permite decidir entre dos opciones resultantes de la evaluación de una condición. Si ésta se cumple, se ejecuta una parte de código, caso contrario, se sigue con las sentencias que siguen luego de la estructura IF o, en su defecto se ejecuta el código que se encuentra dentro de una sentencia **else**.

Cuando se procesa una declaración “IF” se evalúa la expresión de condición y el resultado es interpretado como un valor booleano. Si el resultado es verdadero, se ejecutan las sentencias contenidas dentro del “IF”. Caso contrario, el programa “saltea” lo que se encuentra dentro de este bloque.

**if (num2 > num1){**

**System.out.println("Num2 es mayor que Num1");**

**}**

## Condicional Simple: IF | ELSE + IF | ELSE

Así como el IF y el ELSE se combinan entre sí, también se pueden combinar con otros IF y ELSE dentro de ellos, generando así **condicionales múltiples**.

Los condicionales múltiples no son nada más que estructuras **IF** y **ELSE** **anidadas** entre sí que nos permiten tener mayor control del flujo de un programa dependiendo de lo que queremos que se haga (o no) según diferentes condiciones y situaciones.

**Condicional Múltiple**

Se utiliza cuando se necesita anidar varias condiciones IF si no se cumple una primera. Consiste en realizar una primera evaluación de una condición, si esta no se cumple, por el lado del ELSE se plantea una nueva condición y así sucesivamente la cantidad de veces que sean necesarias.

**if (num2 > num1){**

**System.out.println("Num2 es mayor que Num1");**

**}else{**

**if (num1 == num2){**

**System.out.println("Num1 es igual que Num2");**

**}else{**

**System.out.println("Num1 es mayor que Num2");**

**}**

**}**

## Condicional SWITCH

La **estructura “Switch”** permite múltiples caminos a partir de la evaluación de una sola expresión/condición. La construcción de esta estructura se ejecuta mediante la evaluación de la condición y un conjunto de casos llamados cases.

Cada case es una posible respuesta a la evaluación de esa condición, si el valor que se busca coincide con algún case, se ejecuta el mismo hasta la sentencia break o hasta el final del Switch (dependiendo del caso).

La expresión puede ser una variable o cualquier otro tipo, siempre y cuando se evalúe un valor simple. La construcción se ejecuta mediante la evaluación de la condición y un conjunto de sentencias CASE. Cada CASE es una posible respuesta a la evaluación de esa condición, si el valor que se busca coincide con algún CASE, se ejecuta el mismo hasta la sentencia break o hasta el final del SWITCH (dependiendo del caso).

**switch(dia){**

**case 1: nombreDia = "Lunes";**

**break;**

**case 2: nombreDia = "Martes";**

**break;**

**case 3: nombreDia = "Miércoles";**

**break;**

**case 4: nombreDia = "Jueves";**

**break;**

**case 5: nombreDia = "Viernes";**

**break;**

**case 6: nombreDia = "Sábado";**

**break;**

**case 7: nombreDia = "Domingo";**

**break;**

**default: nombreDia = "Número de día Invalido";**

**break;**

**}**

## Operador Ternario

**El operador ternario**, es un operador en programación que permite tomar decisiones basadas en una condición y asignar un valor a una variable o expresión en función de si la condición es verdadera o falsa.

Este operador se caracteriza por su sintaxis compacta y su capacidad para simplificar la escritura de condicionales simples en una sola línea de código.

Posee tres principales partes en su estructura:

* **Condición:** Una expresión que se evalúa como **verdadera** o **falsa**.
* **Valor\_si\_verdadero:** El valor que se asignará a la variable si la condición es verdadera.
* **Valor\_si\_falso:** El valor que se asignará a la variable si la condición es falsa.

**variable = (condicion) ? valor\_si\_verdadero : valor\_si\_falso;**

## Estructura Repetitiva WHILE

Las **estructuras repetitivas** nos permiten ejecutar una cierta porción de código por una determinada cantidad de veces. Esta ejecución repetitiva se conoce como «bucle» o iteración, existiendo dos tipos de ellas: Controladas por un centinela o controladas por un contador

**Bucle WHILE**

Permite ejecutar un bloque de código en forma repetitiva “mientras” (WHILE en inglés) se cumpla una determinada condición. La condición se evalúa ***siempre*** al inicio del ciclo/bucle. Si la condición es válida, ingresa al bucle y ejecuta el código. Cuando deja de cumplirse la condición, el programa sale del bucle y continúa con el resto del programa.

Controlados por contador

**int contador = 0;**

**while (contador <= 10){**

**System.out.println(“Estoy en la vuelta N°: ” + contador);**

**contador = contador + 1;**

**}**

## Controlados por centinela

**boolean centinela = true;**

**while (centinela == true){**

**System.out.println(“El valor de la bandera es ” + centinela);**

**centinela = false;**

**}**

## Bucle Infinito

Un bucle infinito se produce cuando por algún motivo el programa entra en un bucle, pero el mismo no tiene una condición de salida. Esto puede suceder tanto en los bucles controlados por contador como en los controlados por centinela. Se llama ***bucle infinito*** porque, al no haber una condición de salida, el bucle continúa ejecutándose sin fin.

Controlados por contador

**int contador = 0;**

**while (contador <= 10){**

**System.out.println(“Estoy en la vuelta N°: ” + contador);**

**}**

## Controlados por centinela

**boolean centinela = true;**

**while (centinela == true){**

**System.out.println(“El valor de la bandera es ” + centinela);**

**}**

**Estructuras Repetitivas: FOR**

Así como tenemos a la estructura While, que puede ser controlada tanto por centinelas como por contadores, también contamos con otra estructura que es considerada como la preferida a la hora de ejecutar bucles controlados por contador. Esta estructura es el conocido FOR.

La estructura FOR, está compuesta por tres partes:

1. **Inicialización de la variable** que utilizaremos en la condición (se ejecuta solo una vez al principio del ciclo).
2. **Condición de fin del ciclo** (se evalúa esta expresión al comienzo de cada iteración).
3. **Modificación (incremento o decremento) de la variable** (se ejecuta al final de cada iteración).

El For también es la estructura que se utiliza por excelencia para recorrer arrays (vectores y matrices).

**Bucle FOR**

El bucle FOR realiza un bucle una determinada cantidad de veces. Se lo puede considerar un bucle controlado por contador, sin embargo, posee su variable controladora propia.

**for (int i = 0; i <= 10; i++){**

**System.out.println(“Iteración i número: ” + i);**

**}**

## VECTORES: Declaración y Asignación

Los **arreglos (o arrays)** son un conjunto de datos que se almacenan en memoria de manera contigua con el mismo nombre, pero con diferentes índices para diferenciar cada uno de ellos. Son estructuras fijas que se declaran y que mantienen su tamaño durante toda la ejecución del programa. Si declaro un arreglo de 5 posiciones, mantendrá esas 5 posiciones **siempre**. Existen arreglos **unidimensionales y bidimensionales**.

# **Estructuras Estáticas o Arrays (Vectores y Matrices)**

Los arreglos (o arrays en inglés), son un conjunto de datos que se caracterizan por almacenarse en memoria de manera contigua, bajo un mismo nombre, pero con diferentes “índices” o “identificadores” para diferenciar la ubicación de cada uno de ellos. Son conocidos, en muchas ocasiones por sus dos tipos más comunes: los vectores y las matrices.

Los arreglos son estructuras fijas, es decir, que una vez declarados e inicializados, mantienen su tamaño durante toda la ejecución del programa. ¿Qué quiere decir esto? Por ejemplo, si se declara e inicializa un arreglo con 5 posiciones, estas 5 se mantendrán de principio a fin de la ejecución del programa que se esté desarrollando sin la posibilidad de cambiar su tamaño.

Los arreglos, al igual que las variables comunes, deben poseer **un solo tipo de dato** determinado. Este tipo de dato debe ser **único para todos los elementos** que conforman el array con el que se esté trabajando. Un ejemplo de un arreglo de tipo numérico puede verse representado en la ilustración 1.

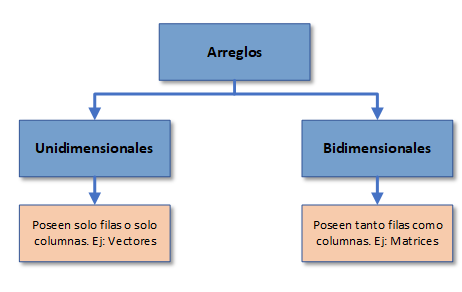


**Ilustración 1:** Ejemplo de arreglo

## Tipos de Arreglos

Existen dos tipos principales de arreglos, los unidimensionales y los bidimensionales. Los arreglos unidimensionales son aquellos que poseen una sola dimensión sin importar su disposición, es decir, posee solo una fila o una columna con un conjunto de valores. Un ejemplo por excelencia de este tipo de arreglos por excelencia son los vectores.

Por otro lado, los arreglos bidimensionales son aquellos que poseen dos dimensiones, es decir, tanto filas como columnas al mismo tiempo. Un ejemplo de esto son las reconocidas matrices.

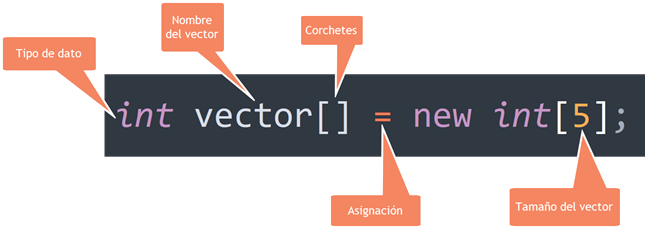


**Ilustración 2:** Clasificación de Arreglos

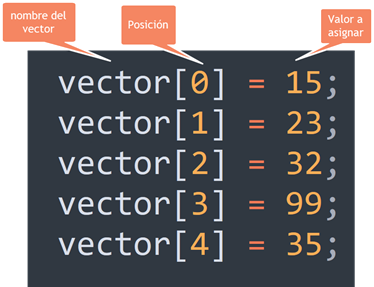
# **Declaración, inicialización y asignación de vectores en Java**

Como vimos anteriormente en los vectores en java o cualquier lenguaje de programación son [arreglos **unidimensionales**](https://todocodeacademy.com/estructuras-estaticas-arrays-vectores-matrices/) que permiten almacenamiento de datos de manera contigua. En Java se declaran especificando **el tipo de dato** que almacenarán, **el nombre** y **la identificación []** la cual determina que se trata de un vector. Al mismo tiempo, para **inicializar** las posiciones de un vector, es necesario asignar al vector declarado **la palabra new** más **el tipo de dato** y **nuevamente []**, donde esta vez se especifica **la longitud** que tendrá el arreglo.

Un ejemplo práctico de esto puede visualizarse en la Ilustración a continuación.

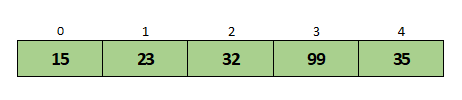
Declaración e inicialización de un vector

Una vez declarado e inicializado un vector, es posible asignarle diferentes valores en cada una de sus posiciones a partir de dar a conocer el índice donde estos datos deberán ir. A nivel código, si suponemos un vector de 5 posiciones, esta asignación de valores se puede realizar, por ejemplo, mediante las siguientes líneas de código:



Ejemplo de asignación de vectores en Java

Esta misma asignación que acabamos de hacer, si la imaginamos a nivel gráfico y considerando las posiciones contiguas de memoria que ocupa un vector, se podría asemejar bastante a la siguiente imagen:



Ejemplo gráfico de asignación de valores a un vector

Un detalle **muy importante** a tener en cuenta es que, por convención mundial, los vectores comienzan su índice en el **valor 0**. ¿Qué quiere decir esto? Que sí tenemos un vector de 5 posiciones, sus índices irán del 0 al 4, por lo que sí hacemos referencia al índice 5, no estaríamos posicionados en la 5ta posición, sino en la sexta; esto, al tratarse de un vector de únicamente 5 posiciones provocaría **un error por desbordamiento**.

**Recorrido y Carga de vectores en Java**

## Recorrido

En **Java** los vectores se inicializan y luego pueden empezar a contener valores a partir de asignaciones. Sin embargo, a partir de esto, en algún momento, vamos a necesitar conocer estos valores o incluso cargar valores nuevos. Para ello existen procesos de tanto de **recorrido como de carga** para los mismos.

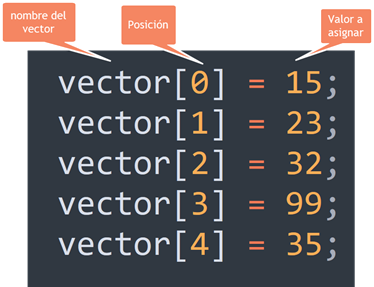
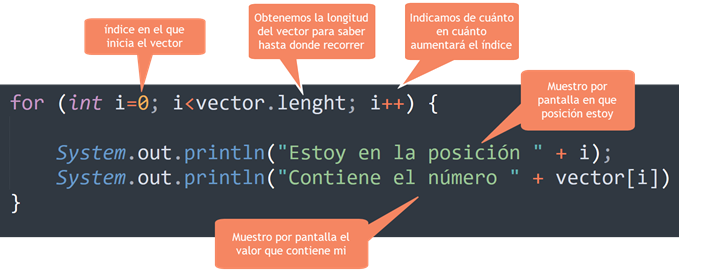


Imagen 1: Vector cargado

Un vector que ya se encuentra cargado o con valores asignados (Ej: imagen 1), puede ser recorrido. Este recorrido se lleva a cabo, tanto para mostrar los valores que contiene el vector como para utilizarlos en caso de que sean necesarios. Para realizar este recorrido la mejor opción es utilizar la **estructura repetitiva for**.

Los vectores se recorren SIEMPRE de **manera secuencial**, es decir, posición a posición según un determinado orden que se establezca. Suponiendo el mismo vector de la imagen 1, podríamos recorrerlo mediante la porción de código que se encuentra a continuación en la imagen 2:

Imagen 2: Recorrido de un vector

El ciclo for siempre tendrá **tres parámetros**, el primero corresponde a la **inicialización de una variable** “i” que representará, en este caso, el índice del vector.

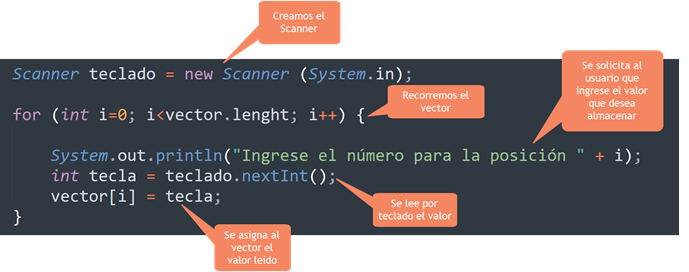
Como **segundo parámetro**, tenemos la **condición de parada** en la cual, mediante la función length podemos obtener la longitud exacta de nuestro vector, para asegurarnos de que no haya un error por desbordamiento y que el recorrido pare cuando llegue a la última posición.

Por último, como **tercer parámetro**, tenemos a la modificación (también conocida como incremento o decremento), es decir, la expresión que nos va a indicar de cuanto en cuanto queremos que crezca o disminuya nuestro índice para así poder hacer el recorrido secuencial. En este caso particular, iremos posición por posición (1 a 1), es por ello que se especifica i++ (equivalente a colocar i=i+1).

## Carga

**Recorrer un vector es bastante sencillo** y la estructura repetitiva for nos brinda una gran ayuda. **Supongamos que el vector se encuentra vacío** y lo que se desea realizar es permitirle al usuario la posibilidad de cargar los vectores por teclado. Para ello, será necesario utilizar una clase especial llamada **Scanner**.

El **Scanner** es una clase (concepto de la programación orientada a objetos) que permite el ingreso de información o datos mediante algún periférico de entrada. Para hacer uso de él y poder cargar nuestro vector, es necesario recorrerlo e ir “Scanneando” los valores para cada posición. Un ejemplo de esto puede verse en la Imagen 3.

Imagen 3: Scanneo de elementos

**Modelo de capas: IGU, Lógica y Persistencia**

A la hora de programar existen ciertos patrones de diseño o buenas prácticas que podemos llevar a cabo para que nuestro código sea más legible, funcione mejor, tenga mayor facilidad de mantenimiento, entre otras cuestiones. Entre los patrones de diseño más conocidos para lograr un código eficiente encontramos al patrón MVC (modelo, vista controlador).

Este patrón es bastante «complicado» de aplicar cuando recién lo conocemos, es por ello que hoy te mostramos una forma de «aproximarnos» a él mediante otro modelo similar conocido como «modelo de capas». En él, existen tres capas que interactúan entre sí:

* La interfaz gráfica de usuario (IGU)
* La lógica (de negocio)
* La persistencia (o controladora de base de datos)

