# Sintaxis y Semántica de los Lenguajes

## Trabajo Practico Teorico

**Alumnos:** Matias Cornalo, Facundo Bove Hernandez, Santiago Invernizzi, Santiago Perez

**Comisión:** K2055

**Año:** 2023

**Profesora:** Roxana Leituz

# Lenguajes

## Java

### *Historia*

El lenguaje Java fue desarrollado en sus inicios por James Gosling, en el año 1991. Inicialmente Java era conocido como Oak, que luego pasó a llamarse Red, y finalmente Java.
Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box (decodificador) en una pequeña operación denominada *The Green Project* en Sun Microsystems en 1991. El lenguaje fue desarrollado por un equipo de 13 personas, encabezado por James Gosling, y el objetivo era implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Posteriormente, el equipo reorientó la plataforma hacia la Web, y Patrick Naughton creó un prototipo de navegador, que luego sería conocido como HotJava. Finalmente, se publica la primera versión de Java en 1996.

### *Descripción*

Java es un lenguaje de programación orientado a objetos. Este lenguaje tiene la filosofía de “write once, run anywhere”, es decir que el mismo código puede ser utilizado en múltiples sistemas operativos, y para esto se compila el código escrito en Java, generando un código conocido como “bytecode”, el cual es ejecutado en la JVM (Java Virtual Machine). La JVM es un programa escrito en el código nativo de la plataforma destino, y esta interpreta y ejecuta el código. Debido a la forma en que funciona Java, es un lenguaje híbrido entre compilado e interpretado, ya que tiene la primera etapa de compilación (cuando se genera el bytecode), y luego este código es interpretado.

Usos que se le suele dar a Java:
-Desarrollo de videojuegos: se usa para crear videojuegos de varias plataformas, incluyendo juegos modernos que integran tecnologías como el machine learning o la realidad virtual.
-Cloud Computing: para aplicaciones descentralizadas basadas en la nube.
-Macrodatos: para motores de procesamiento de datos que trabajan con datos complejos y grandes cantidades de datos en tiempo real.
-Inteligencia artificial: para proporcionar gran cantidad de bibliotecas de machine learning.
-Internet de las cosas: para programar sensores y hardware en dispositivos que se conectan independientemente a internet.

BNF Java
https://cs.au.dk/~amoeller/RegAut/JavaBNF.html

JavaScript se introdujo en 1995, creado por brendan Eich, como una forma de agregar programas a páginas web en el navegador Netscape Navigator

Características

- Débilmente tipado
- Interpretado

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una **velocidad máxima de 28.8 kbps**. Esa velocidad era **más que suficiente** para la época salvo que quisieras descargar imágenes de cierto tamaño. Lo cierto era que la web en aquel entonces no ofrecía gran cosa más que servir como una inmensa biblioteca donde los usuarios consultaban mayormente contenido basado en texto.

En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos. Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario.

Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar las limitaciones de la web de ese entonces, adaptando otras tecnologías existentes (como ScriptEase) al navegador Netscape Navigator 2.0, que iba a lanzarse en aquel año. Inicialmente, Eich denominó a su lenguaje LiveScript y fue un éxito.

Fue entonces cuando, justo antes del lanzamiento, Netscape decidió cambiar el nombre por el de JavaScript y firmó una alianza con Sun Microsystems para continuar el desarrollo del nuevo lenguaje de programación.

El motivo principal del cambio de  nombre al lenguaje,  fue marketing .  Cuando se introdujo JavaScript, el lenguaje Java se estaba comercializando en gran medida y estaba ganando popularidad.

Actualmente es uno de los lenguajes de programación más demandados y se encuentra presente en prácticamente cualquier aplicación web. Empresas como Google o Facebook han desarrollado incluso sus propias implementaciones basadas en este lenguaje para lograr por ejemplo una experiencia visual del usuario más atractiva al momento de actualizar tus estados.

Es principalmente implementado el lenguaje en:

- Creación de páginas web
- Desarrollo de todo el backend de una aplicación, programando con Node.js, que sigue siendo JavaScript.
- Sistemas operativos
- Servidores de Internet
- Bases de datos
- Plataformas de juego
- Desarrollo móvil

**BNF JS**
https://www.ecma-international.org/wp-content/uploads/ECMA-262_14th_edition_june_2023.pdf

# *ALGORITMOS*

¿cómo comparamos algoritmos? , en base a qué nos basamos para decir que uno es mejor que otro, bueno… Para eso hay que ver un concepto importante, la notación Big O.

Pero primero introduzcamos los 2 algoritmos a comparar : **Selection and bubble sort.**

**Bubble sort :**
cómo funciona :

```java
public static void bubbleSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

Este Algoritmo es más fácil de entender que el de selección, lo que hace es comparar el primer con el segundo, segundo con el tercero… y así. Si es menor, lo cambia.

**Selection Sort :**

```java
public static void selectionSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}
```

El algoritmo de selección es un poco más complejo que el bubble sort, pero más eficiente, principalmente porque compara menos veces, sin embargo, no nos vamos a centrar en la comparación de los algoritmos en sí, sino en la comparación de cada uno según el lenguaje. ¿Cómo funciona? Básicamente itera el primer for y toma el primer elemento como mínimo (minIndex), para después iterar el segundo buscando si hay alguno menor comparado con nuestro minIndex. Si lo hay, lo asigna al minIndex y hace después el swap (intercambio).

*Los códigos están en Java.*

Ahora si, la notación **BigO.**

Esta es fundamental y es útil para evaluar cómo es la curva de crecimiento de su tiempo de ejecución conforme aumentan el tamaño del input de entrada, en este caso, se analiza el tiempo entre más grande sea el tamaño del array.

Hay diferentes clasificaciones :
O(1) -> Tiempo constante: es el mejor resultado, básicamente el tiempo de ejecución no varía.
O(n) -> Tiempo lineal : el crecimiento es lineal a la cantidad de elementos, si tenemos el doble de elementos, tardará el doble.
O(log n) -> en esta forma, crece mucho más rápido al inicio, pero tiende a estabilizarse cuando la muestra se hace más grande.
O(n logn) -> en este caso se trata de funciones similares a las anteriores, pero que rompen el problema en varios trozos por cada elemento, volviendo a recomponer información tras la ejecución de cada "trozo".
O(n^2) -> en tiempo cuadrático, crece de forma exponencial, por lo que frente a muchos datos, va a empezar a tardar demasiado, dejando inoperativo al procesador. Es útil para pruebas más pequeñas.

*Hay diferentes casos :*
**Worst case :** En este caso ,se busca llegar al límite del algoritmo, hay una cantidad muy grande de elementos, y ninguno está ordenado, por lo que por cada comparación deberá ordenarlo, evidentemente, esto lleva mucho más tiempo. Es el cálculo del límite superior.

**Average Case :** En el análisis de caso promedio, tomamos todas las entradas posibles(elementos) y calculamos el tiempo de cálculo para todos. Luego se suman y se dividen por la cantidad para calcular el promedio. Esto, al igual que el best case, no se suele usar tanto, ya que el que da más información es el Worst case.

**Best Case** : Lo contrario al anterior, en este caso, el array ya está ordenado ,por lo tanto no necesita hacer el intercambio. Es el cálculo del límite inferior. El mejor caso sería O(1), es decir, es constante, no varía.

**Bubble Sort** -> En el mejor de los casos, tiene una notación O(n) ideal, cuando ya está ordenado,

En el peor de los casos, tiene una notación O(n^2), es decir, que entre más elementos no ordenados haya, más tiempo va a tardar, hasta llegar al punto en el que exceda el tiempo de cálculo (time exceed), y en el promedio también tiene una notación O(n^2)

**Selection Sort** -> En el mejor, peor y average de los casos, la notación es O(n^2).

**Space Complexity :** El “space complexity” determina la cantidad de memoria necesaria que utiliza el algoritmo.
Al igual que para el time (time complexity), se pueden clasificar de la siguiente manera :

O(1) -> Constant space : El uso de memoria es constante, no varía. Obviamente este sería el caso deseado, pero no es lo más común cuando trabajamos con arreglos grandes. En 1 indica que no hay relación con n, no que necesita una cantidad 1 de memoria.

O(n) -> Linear Space : Esto indica que requiere uso de memoria adicional, la cantidad de memoria adicional aumenta en proporción lineal al input (cantidad de elementos del array).

O(n^2) - Quadratic Space: el uso de memoria crece de forma cuadrática. Está asociada a algoritmos que  utilizan estructuras de datos bidimensionales, como una matriz.

Esta notación Big O( que utilizamos para describir el time and space complexity) nos es de mucha utilidad principalmente cuando trabajamos con arrays de un cantidad de elementos grande.

***Diferencia segun lenguaje :***

Bueno, ya vimos lo que implica comparar algoritmos según espacio y tiempo, pero también hay que considerar los lenguajes en los que lo estamos testeando, en este caso, javascript y Java.

Una pequeña comparación entre los lenguajes, según las características que vimos más arriba.

En términos de velocidad, Java tiende a ser más rápido que JavaScript debido a las diferencias en la forma en que se ejecutan estos dos lenguajes y las tecnologías que los respaldan.

Compilación y Optimización: Java es un lenguaje compilado en el que el código fuente se traduce en bytecode que luego se ejecuta en la Máquina Virtual Java (JVM). La JVM realiza una serie de optimizaciones en tiempo de ejecución, como la compilación Just-In-Time (JIT), que convierte el bytecode en código de máquina nativo. Esto puede permitir que el código Java se ejecute más rápido que JavaScript, que generalmente se interpreta línea por línea (interpretado).

En otras palabras, Java, al ser compilado,  es más rápido, y eso genera una pequeña diferencia de tiempo en la ejecución de los algoritmos (Algo que generalmente es más notorio cuando hay más elementos).

Además, JVM es capaz de aplicar optimizaciones avanzadas como la eliminación de código muerto, la inlining de métodos y la reorganización de bucles para mejorar el rendimiento. Estas optimizaciones pueden llevar a un rendimiento más rápido en comparación con JavaScript, que generalmente se ejecuta en un entorno de navegador y no tiene un nivel de optimización comparable.

Java es un lenguaje estáticamente tipado, lo que permite a la JVM realizar optimizaciones de tipo en tiempo de compilación, previo a ejecutar el código. Esto puede llevar a una ejecución más rápida y eficiente. En contraste, JavaScript es dinámicamente tipado, lo que puede dificultar algunas optimizaciones en tiempo de ejecución.

La JVM está diseñada también para aprovechar las características del hardware moderno, como la ejecución en paralelo y la gestión eficiente de memoria. Estas optimizaciones pueden influir en el rendimiento general del código Java en comparación con el código JavaScript.

Otra aclaración muy importante, al correr el código, no sólo entran en juegos los lenguajes, sino también el procesador, la memoria, etc de la pc en la que se está evaluando.

En resumen, tanto en Java como en JavaScript, los algoritmos de ordenamiento Selection Sort y Bubble Sort tienen complejidades de tiempo y memoria similares. Sin embargo, en términos de tiempo de ejecución, Java puede ser más rápido debido a su naturaleza compilada y a las optimizaciones de la JVM. El rendimiento en JavaScript dependerá de factores como el motor JavaScript utilizado. No dependerá tanto del lenguaje sino de los algoritmos a ejecutar.

**Ejemplos prácticos de runtime.**

Estas son las especificaciones de la pc en la que se hicieron las pruebas :

AMD Ryzen 5 3600 6-Core Processor  3.59 GHz 16 GB

**Código Tiempo de ejecución Java Bubble Sort.**

```
import java.util.Random; // El import es para poder usar Random

public class SortingAlgorithms {

	public static void bubbleSort(int[] arr) {
		int n = arr.length;
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n - i - 1; j++) {
				if (arr[j] > arr[j + 1]) {
					int temp = arr[j];
					arr[j] = arr[j + 1];
					arr[j + 1] = temp;
				}
			}
		}
	}
  // Este pedazo de código genera el array con n cantidad para después ser testeado.
	public static void main(String[] args) {
		int[] arr = new int[10000];
		Random rand = new Random();
		for (int i = 0; i < 10000; i++) {
			arr[i] = rand.nextInt(10000) + 1;
		}
// Esto lo usamos para calcular el tiempo.
		long startTime = System.nanoTime();
		bubbleSort(arr.clone());
		long bubbleSortTime = System.nanoTime() - startTime;

		System.out.println("Bubble Sort time: " + bubbleSortTime);
	}
}
```

Output -> 12.2817s con un n = 100.000 elementos. También probamos el caso en el que sea menor, con 10.000, pero la diferencia no era lo suficientemente notoria.

**Código Tiempo de ejecución JavaScript Bubble Sort.**

```
function bblSort(arr) {

    for (var i = 0; i < arr.length; i++) {

        for (var j = 0; j < (arr.length - i - 1); j++) {


            if (arr[j] > arr[j + 1]) {

                var temp = arr[j]
                arr[j] = arr[j + 1]
                arr[j + 1] = temp
            }
        }
    }

  //  console.log(arr);
}

function generarArrayAleatorio(n) {
  const array = [];

  for (let i = 0; i < n; i++) {
    const numeroAleatorio = Math.floor(Math.random() * 100); // Genera un número
aleatorio entre 0 y 99
    array.push(numeroAleatorio);
  }

  return array;
}

const n = 100000; // Cambia este valor a la cantidad de elementos que desees
const miArrayAleatorio = generarArrayAleatorio(n);
// console.log(miArrayAleatorio);

console.time('Execution Time');
bblSort(miArrayAleatorio);
console.timeEnd('Execution Time'); // Tiempo de ejecución
```

Output -> 15.114s con un n = 100.000 elementos.

En Bubble sort, Java es más rápido que javascript (+ 2.83).

**Código Tiempo de ejecución Java Selection Sort.**

```
import java.util.Random;

public class SortingAlgorithms {

    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr = new int[100000];
        Random rand = new Random();
        for (int i = 0; i < 100000; i++) {
            arr[i] = rand.nextInt(100000) + 1;
        }

        long startTime = System.nanoTime();
        selectionSort(arr.clone());
        long selectionSortTime = System.nanoTime() - startTime;

        System.out.println("Selection Sort time: " + selectionSortTime);

    }
}
```

Output -> 15.114s con un n = 100.000 elementos.

**Código Tiempo de ejecución Java Selection Sort.**

```
function selectionSort(arr) {
    const n = arr.length;
    for (let i = 0; i < n; i++) {
        let minIndex = i;
        for (let j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        const temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

const arr = new Array(100000);
for (let i = 0; i < 100000; i++) {
    arr[i] = Math.floor(Math.random() * 100000) + 1;
}

const startTime = process.hrtime.bigint();
selectionSort([...arr]);
const selectionSortTime = process.hrtime.bigint() - startTime;

console.log("Selection Sort time: " + selectionSortTime + " nanoseconds");
```

Output -> 6.1671 con un n = 100.000 elementos.

Nuevamente Java supera a Javascript en velocidad.

**Información Adicional :**

https://gist.github.com/aprilmintacpineda/f7368e7578afb42b86ae466a2fd87bce
En este código se hace una comparación muy parecida a la que hacemos nosotros, aunque la diferencia da un poco menos. Los códigos son diferentes y la computadora donde se evaluó también, pero nos pareció interesante ponerlo para tener otra fuente de comparación (Java sigue ganándole a Javascript).