Java, El Lenguaje de Programación Orientado a Objetos (Básico).

Logo[1].

Java es un lenguaje de programación y una plataforma informática que fue comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán, probablemente, a menos que tengan Java instalado, y cada día se crean más.

Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios en el mundo.

Java es rápido, seguro y confiable, para ordenadores portátiles, centros de datos, consolas para juegos, computadoras avanzadas, teléfonos móviles, hasta Internet, Java está en todas partes. Si es ejecutado en una plataforma no tiene que ser recompilado para correr en otra[2].

¿QUÉ ES EL LENGUAJE DE PROGRAMACIÓN JAVA?

El lenguaje de programación Java fue desarrollado originalmente por James Gosling, de Sun Microsystems (constituida en 1983 y posteriormente adquirida el 27 de enero de 2010 por la compañía Oracle),4 y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente[3].

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales y librerías de clases en 1991, y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento de las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros han desarrollado también implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Java (lenguaje de programación), está influido por Pascal, C++, Objective-C, y ha influido a, C#, J#, JavaScript, PHP, Python.

Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, enlace dinámico y polimorfismo. Los modelos de objetos son llamados, como en C++, clases y sus copias, instancias.

SINTAXIS JAVA[4].

Una clase siempre debe comenzar con una primera letra mayúscula. Java distingue entre mayúsculas y minúsculas: "MiClase" y "miclase" tienen un significado diferente.

El nombre del archivo java debe coincidir con el nombre de la clase. Al guardar el archivo, guárdelo con el nombre de la clase y agregue ".java" al final del nombre del archivo.

El método main()
El método main() es obligatorio y lo verá en todos los programas de Java:

```
public static void main(String[] args)
```

System.out.println()
Dentro del método main(), podemos usar el método println(), para imprimir una línea de texto en la pantalla:

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hola programador");
  }
}
```

```
Hola programador
```

Nota: Las llaves {} marcan el principio y el final de un bloque de código.
Nota: Cada declaración de código debe terminar con un punto y coma.

Comentarios Java de una sola línea
Los comentarios se utilizan para explicar el código Java y también se puede usar para evitar la ejecución al probar código alternativo.

Los comentarios de una sola línea comienzan con dos barras inclinadas (//), cualquier texto entre // y el final de la línea es ignorado por Java (no se ejecutará). Ejemplo:

```
public class Main {
  public static void main(String[] args) {
    // Este es un comentario
    System.out.println("Hello Estudiante");
  }
```

```
Hello Estudiante
```

VARIABLES JAVA[5].

Las variables son contenedores para almacenar valores de datos, en Java, existen diferentes tipos de variables, por ejemplo:

- ✓ String: almaceno texto: ejemplo: "Hola". Y se escribe entre comillas.
- ✓ Int: almacena números, sin decimales, como -256 o 256.
- ✓ Float: almacena números flotantes, con decimales, como -20.5 o 25.6.
- ✓ Char: almacena caracteres individuales, como 'x' o 'Y'. Los valores se escriben entre comillas simples.
- ✓ Boolean: almacena valores con dos estados: verdadero(true) o falso(false).

Declarar (crear) variables

Para crear una variable, debe especificar el tipo y asignarle un valor:
Sintaxis: *type nombreVariable = valor;*

Donde *type* es uno de los tipos de Java (como int o String), y *nombreVariable* es el nombre de la variable (como **x** o **nombre**). El signo igual se utiliza para asignar valores a la variable. ejemplos:
Cree una variable llamada nombre de tipo String y asígnele el valor "William"

```
public class Main {
  public static void main(String[] args) {
    String nombre = "William";
    System.out.println(nombre);
  }
}
```

```
William
```

Cree una variable llamada miNum de tipo int y asígnele el valor 35:

```
public class Main {
  public static void main(String[] args) {
    int miNum = 35;
    System.out.println(miNum);
  }
}
```

```
35
```

También puede declarar una variable sin asignar el valor y asignar el valor más tarde:

```
int miNum;
myNum = 65;
System.out.println(miNum);
```

Asignar un nuevo valor a una variable existente, sobrescribirá el valor anterior:

```
public class Main {
  public static void main(String[] args) {
    int miNum = 30;
    miNum = 63;  // miNum ahora es 63
    System.out.println(miNum);
  }
}
```

```
63
```

Variables finales

Pero puede agregar la palabra clave final, esto declarará la variable como "final" o "constante", lo que significa que no se puede modificar y es de solo lectura:

```
final int miNum = 65;
miNum = 30; // genera un error
```

Otros tipos

Cómo declarar variables de otros tipos:

```
int miNum = 56;
float miNumFlotante = 3.89f;
char myLetra = 'W';
boolean miBooleana = true;
String miTexto = "Hola Esstudiante";
```

Las reglas generales para nombrar variables son:

- ✓ Los nombres pueden contener letras, dígitos, guiones bajos y signos de dólar.
- ✓ Los nombres deben comenzar con una letra.
- ✓ Los nombres deben comenzar con una letra minúscula y no pueden contener espacios en blanco
- ✓ Los nombres también pueden comenzar con $ y _.
- ✓ Los nombres distinguen entre mayúsculas y minúsculas ("miVar" y "mivar" son variables diferentes)
- ✓ Las palabras reservadas (como las palabras clave de Java, como int o boolean) no se pueden usar como nombres.

TIPOS DE DATOS JAVA[6].

Los tipos de datos se dividen en dos grupos:

Tipos de datos primitivos: incluye byte, short, int, long, float, double, boolean y char. Un tipo de datos primitivo especifica el tamaño y el tipo de los valores de las variables y no tiene métodos adicionales, hay ocho tipos de datos primitivos en Java:

Tipo de Dato	Tamaño	Descripción
byte	1 byte	-128 a 127
short	2 bytes	-32,768 a 32,767
int	4 bytes	-2,147,483,648 a 2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	4 bytes	6 a 7 dígitos decimales
double	8 bytes	15 dígitos decimales
boolean	1 byte	Verdadero(true) o falso(false)
char	2 bytes	Almacena un solo carácter/letra o valores ASCII

Tipos de datos no primitivos

Tipos de datos no primitivos son: String, Arrays, Interface y Classes.
Los tipos de datos no primitivos se denominan **tipos de referencia** porque se refieren a objetos, la principal diferencia entre los tipos de datos **primitivos** y **no primitivos** son:

- ✓ Los tipos primitivos están predefinidos en Java. Los tipos no primitivos son creados por el programador y no están definidos por Java (excepto para String).
- ✓ Los tipos no primitivos se pueden usar para llamar a métodos para realizar ciertas operaciones, mientras que los tipos primitivos no pueden.
- ✓ Un tipo primitivo siempre tiene un valor, los tipos no primitivos pueden ser null.
- ✓ Un tipo primitivo comienza con una letra minúscula, mientras que los tipos no primitivos comienzan con una letra mayúscula.

- ✓ El tamaño de un tipo primitivo depende del tipo de datos, mientras que los tipos no primitivos tienen el mismo tamaño.

OPERADORES JAVA[7].

Los operadores se utilizan para realizar operaciones en variables y valores.
Java divide los operadores en los siguientes grupos:

Operadores aritméticos.

Operador	Nombre	Descripción	Ejemplo
+	Adición	Suma dos valores	a + b
-	Sustracción	Resta un valor de otro.	a - b
*	Multiplicación	Multiplica dos valores.	a * b
/	División	Divide un valor por otro.	a / b
%	Módulo	Devuelve el resto de la división.	a % b
++	Incremento	Aumenta el valor de una variable en 1.	++ a
--	Decremento	Disminuye el valor de una variable en 1.	-- a

Operadores de Asignación.
Los operadores de asignación se utilizan para asignar valores a las variables.

Operador	Ejemplo	Igual que
=	a=20	a = 20
+=	a+=30	a = a + 30
-=	a-=25	a = a - 25
*=	a*=14	a = a * 14
/=	a/=3	a = a / 3
%=	a%=9	a = a % 9
&=	a&=4	a = a & 4
|=	a|=8	a = a | 8
^=	a^=20	a = a ^ 20
>>=	a>>=7	a = a >> 7
<<=	a<<=6	a = a << 6

Operadores de comparación.
Los operadores de comparación se utilizan para comparar dos valores:

Operador	Nombre	Ejemplo
= =	Igual que	a = = b
! =	Distinto que	a! = b
>	Mayor que	a > b
<	Menor que	a < b
> =	Mayor o igual que	a > = b
< =	Menor o igual que	a < = b

Operadores lógicos.
Los operadores lógicos se utilizan para determinar la lógica entre variables o valores:

Operador	Nombre	Descripción	Ejemplo
& &	Lógico I	Suma dos valores	a + b
| |	Lógico O	Resta un valor de otro.	a - b
!	Lógico negación	Multiplica dos valores.	a * b

CONDICIONES JAVA IF ... ELSE[8].

Java tiene las siguientes declaraciones condicionales:
if
Úse if para especificar un bloque de código que se ejecutará, si la condición es verdadera.
Sintaxis:

```
if (condicion) {
// bloque de código que va a ejecutar si la condición es verdadera.}
```

Tenga en cuenta que if está en minúsculas, las letras mayúsculas (If o IF) generarán un error. Ejemplo, probamos dos valores para averiguar si 30 es mayor que 5. Si la condición es true, imprime un texto:

```
public class Main {
  public static void main(String[] args) {
    if (30 > 5) {
      System.out.println("30 es mayor que 5"); // obvio
    }
  }
}
```

```
30 es mayor que 5
```

else
Se usa else para especificar un bloque de código al ejecutar, si la misma condición es falsa. Sintaxis:

```
if (condición) {
// bloque de código que va a ejecutar si la condición es verdadera
} else {
 // bloque de código que va a ejecutar si la condición es falsa
}
```

```java
public class Main {
  public static void main(String[] args) {
    int hora = 23;
    if (hora < 18) {
      System.out.println("Buen día.");
    } else {
      System.out.println("Buenas Tardes.");
    }
  }
}
```

```
Buenas Tardes.
```

else if

Úse else if para especificar una nueva condición, para probar si la primera condición es falsa. Sintaxis:

```java
if (condicion1) {
 // bloque de código que va a ejecutar si la condición1 es verdadera
} else if (condicion2) {
// bloque de código que se ejecutara si la condición1 es falsa y la
//condición2 es verdadera
} else {
// bloque de código que va a ejecutar si la condición1 y la condición2
//son falsas
}
```

```java
public class Main {
  public static void main(String[] args) {
    int num = 2;
    if (num < 0) {
      System.out.println("Es un numero negativo");
    } else if (num == 0) {
      System.out.println("Es un 0");
    }  else {
      System.out.println("Es un numero positivo");
    }
  }
}
```

```
Es un numero positivo
```

switch

Úse switch para especificar muchos bloques alternativos de código para ejecutar.

```java
switch(expresión) {
case 1:
// bloque de código
break;
case 2:
// bloque de código
break;
default:
// bloque de código
}
```

Así es como funciona: La switch expresión se evalúa una vez, el valor de la expresión se compara con los valores de cada uno case, si hay una coincidencia, se ejecuta el bloque de código asociado, las palabras clave break y default son opcionales.

El siguiente ejemplo utiliza el número para averiguar el color:

```
public class Main {
  public static void main(String[] args) {
    int color = 2;
    switch (color) {
      case 1:
        System.out.println("Rojo");
        break;
      case 2:
        System.out.println("Verde");
        break;
      case 3:
        System.out.println("Azul");
        break;
       default:
    }
  }
}
```

```
Verde
```

CICLO WHILE DE JAVA[9].

Los bucles pueden ejecutar un bloque de código siempre que se alcance una condición específica, los bucles son útiles porque ahorran tiempo, reducen los errores y hacen que el código sea más legible.
El bucle while recorre un bloque de código siempre que una condición específica sea true:
Sintaxis:

```
while (condición) {
// bloque de código en ejecución
}
```

En el siguiente ejemplo, el código en el ciclo se ejecutará una y otra vez, siempre que la variable (cont) sea menor que 3:

```
public class Main {
  public static void main(String[] args) {
    int cont = 0;
    while (cont < 3) {
      System.out.println(cont);
      cont++;
    }
  }
}
```

```
0
1
2
```

El bucle Do/While
El bucle do/while es una variante del bucle while. Este ciclo ejecutará el bloque de código una vez, antes de verificar si la condición es verdadera, luego repetirá el ciclo mientras la condición sea verdadera. Ejemplo:

```
public class Main {
  public static void main(String[] args) {
    int cont = 1;
    do {
      System.out.println("Contador "+ cont + "°");
      cont++;
    }
    while (cont < 4);
  }
}
```

```
Contador 1°
Contador 2°
Contador 3°
```

BUCLE FOR[10].

Cuando sepa exactamente cuántas veces desea recorrer un bloque de código, use el bucle for lugar del bucle while:
Sintaxis:

```
for (declaración 1; declaración 2; declaración 3) {
// bloque de código a ejecutar
}
```

La declaración 1 se ejecuta una vez antes de la ejecución del bloque de código.
La declaración 2 define la condición para ejecutar el bloque de código.
La declaración 3 se ejecuta (todas las veces) después de que se haya ejecutado el bloque de código.
El siguiente ejemplo imprimirá los números del 1 al 6:

```
public class Main {
  public static void main(String[] args) {
    for (int cont = 1; cont < 7; cont++) {
      System.out.println(cont);
    }
  }
}
```

```
1
2
3
4
5
6
```

JAVA MSTRICES (ARRAYS)[11].

Las matrices se utilizan para almacenar múltiples valores en una sola variable, en lugar de declarar variables separadas para cada valor, Para declarar una matriz, defina el tipo de variable con corchetes:

```
String [] colores;
```

Ahora hemos declarado una variable que contiene una matriz de cadenas. Para insertarle valores, podemos usar una matriz literal: coloque los valores en una lista separada por comas, dentro de llaves:

```
String [] colores = {"Rojo", "Azul", "Verde", "Negro", "Blanco"};
```

Para crear una matriz de enteros, escribimos:

```
int [] miNum = {5, 10, 15, 20, 25, 30};
```

Acceder a los Elementos de un Array

Accede a un elemento de matriz haciendo referencia al número de índice.
Ejemplo: esta sentencia accede al valor del primer elemento en colores:

```
public class Main {
  public static void main(String[] args) {
    String[] colores = {"Rojo", "Azul", "Verde", "Negro", "Blanco"};
    System.out.println(colores[0]);
  }
}
```

```
Rojo
```

Cambiar un elemento de matriz

Para cambiar el valor de un elemento específico, consulte el número de índice:

```
public class Main {
  public static void main(String[] args) {
    String[] colores = {"Rojo", "Azul", "Verde", "Negro", "Blanco"};
    colores[0] = "amarillo";
    System.out.println(colores[0]);
  }
}
```

```
amarillo
```

Longitud de la matriz

Para averiguar cuántos elementos tiene una matriz, use la length propiedad:

```
public class Main {
  public static void main(String[] args) {
    String[] transporte = {"avión", "autobus", "taxi", "barco"};
    System.out.println(transporte.length);
  }
}
```

```
4
```

Bucle a través de una matriz

Puede recorrer los elementos de la matriz con el bucle for y usar la propiedad length para especificar cuántas veces debe ejecutarse el bucle. Ejemplo genera todos los elementos en la matriz de **árbol:**

```
public class Main {
  public static void main(String[] args) {
    String[] arbol = {"ramas", "raices", "hojas", "tallo"};
    for (int cont = 0; cont < arbol.length; cont++) {
      System.out.println(arbol[cont]);
    }
  }
}
```

```
ramas
raices
hojas
tallo
```

Bucle a través de una matriz con For-Each

También hay un bucle " **for-each ", que se usa exclusivamente para recorrer elementos en matrices:**

Sintaxis:

```
for (tipo variable: nombrearray) {
  ...
}
```

El siguiente ejemplo genera todos los elementos en la matriz de una **casa**, usando un bucle " **para cada uno ":**

```
public class Main {
  public static void main(String[] args) {
    String[] casa = {"techo", "puertas", "paredes", "habitaciones", "baño",
                     "sala", "cocina"};
    for (String x : casa) {
      System.out.println(x);
    }
  }
}
```

```
techo
puertas
paredes
habitaciones
baño
sala
cocina
```

El ejemplo anterior se puede leer así: **para cada** String elemento (llamado **x** - como en **x** ndex) en **casa**, imprime el valor de **x**.

Si compara el ciclo for y el ciclo **for-each** , verá que el método **for-each** es más fácil de escribir, no requiere un contador (usando la propiedad de longitud) y es más legible.

CONCLUSIÓN

1. Con esta guía tendrá la agilidad de iniciarse en el mundo de la programación orientada a objetos, con el lenguaje Java.

2. Podrá elaborar programas cortos y sencillos que empiecen a desarrollar su lógica de programación.

BIBLIOGRAFÍA

[1] «Java (lenguaje de programación)», *Wikipedia, la enciclopedia libre*. 7 de marzo de 2022. Accedido: 10 de marzo de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Java_(lenguaje_de_programaci%C3%B3n)&oldid=142127450

[2] T. Lindholm y Y. Frank, *Java Virtual Machine Specification Good*, 2nd ed. Better World Books, 1999. Accedido: 10 de marzo de 2022. [En línea]. Disponible en: https://www.iberlibro.com/9780201432947/Java-Virtual-Machine-Specification-Lindholm-0201432943/plp

[3] «A Brief History of the Green Project», 27 de enero de 2007. https://web.archive.org/web/20070127143602/http://today.java.net/jag/old/green/ (accedido 10 de marzo de 2022).

[4] «Sintaxis Java». https://www.w3schools.com/java/java_syntax.asp (accedido 10 de marzo de 2022).

[5] «Java Variables». https://www.w3schools.com/java/java_variables.asp (accedido 10 de marzo de 2022).

[6] «Tipos de datos Java». https://www.w3schools.com/java/java_data_types.asp (accedido 10 de marzo de 2022).

[7] «Operadores Java». https://www.w3schools.com/java/java_operators.asp (accedido 10 de marzo de 2022).

[8] «Java si... si no». https://www.w3schools.com/java/java_conditions.asp (accedido 10 de marzo de 2022).

[9] «Ciclo while de Java». https://www.w3schools.com/java/java_while_loop.asp (accedido 10 de marzo de 2022).

[10] «Java For Loop». https://www.w3schools.com/java/java_for_loop.asp (accedido 10 de marzo de 2022).

[11] «Java Arrays». https://www.w3schools.com/java/java_arrays.asp (accedido 10 de marzo de 2022).