**Introducción**

El sistema binario utiliza solamente dos dígitos ( 0 y 1 ) llamados **bits**.

La palabra bit procede de la unión de las palabras **bi**nary digi**t**.

Un bit es la unidad mínima de representación de información.

Utilizando 1 bit podremos solamente representar dos valores posibles: 0, 1.

Utilizando 2 bits podemos representar 4 valores: 00, 01, 10, 11.

Utilizando 3 bits podemos representar 8 valores:

000, 001, 010, 011, 100, 101, 110, 111.

Utilizando 4 bits podemos representar 16 valores:

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111.

En general, utilizando **N bits**podremos representar **2N valores**.

Ocho bits forman un **byte**.

El byte es la unidad básica de medida de la información.

Un byte es la cantidad más pequeña de información que el ordenador puede manejar.

Con un byte se pueden representar 28 = 256 caracteres.

En el interior del ordenador los datos se transmiten y almacenan en grupos de bytes llamados **palabras.**

La longitud de palabra depende de cada tipo de ordenador: 8, 16, 32, 64.

**Tipos de datos en java, tamaños y su conversión**

1. **Primitivos**

* **byte**: como su propio nombre denota, emplea un solo byte (8 bits) de almacenamiento. Esto permite almacenar valores en el rango [-128, 127].
* **short**: usa el doble de almacenamiento que el anterior, lo cual hace posible representar cualquier valor en el rango [-32.768, 32.767].
* **int**: emplea 4 bytes de almacenamiento y es el tipo de dato entero más empleado. El rango de valores que puede representar va de -231 a 231-1.
* **long**: es el tipo entero de mayor tamaño, 8 bytes (64 bits), con un rango de valores desde -263 a 263-1.

**Tipos numéricos en punto flotante**

* **float**: conocido como tipo de **precisión simple**, emplea un total de 32 bits. Con este tipo de datos es posible representar números en el rango de 1.4x10-45 a 3.4028235x1038.
* **double**: sigue un esquema de almacenamiento similar al anterior, pero usando 64 bits en lugar de 32. Esto le permite representar valores en el rango de 4.9x10-324 a 1.7976931348623157x1030

**Booleanos y carácter**

* **boolean**: tiene la finalidad de facilitar el trabajo con valores "verdadero/falso" (booleanos), resultantes por regla general de evaluar expresiones. Los dos valores posibles de este tipo son true y false.
* **char**: se utiliza para almacenar caracteres individuales (letras, para entendernos). En realidad está considerado también un tipo numérico, si bien su representación habitual es la del carácter cuyo código almacena. Utiliza 16 bits y se usa la codificación UTF-16 de Unicode.

1. **Tipos de datos estructurados, clases u objetos.**

* **String:** Las cadenas de caracteres se delimitan entre comillas dobles, en lugar de simples como los caracteres individuales. En la declaración, sin embargo, no se indica explícitamente que se quiere crear un nuevo objeto de tipo String, esto es algo que infiere automáticamente el compilador.

Las cadenas, por tanto, son objetos que disponen de métodos que permiten operar sobre la información almacenada en dicha cadena. Así, encontraremos métodos para buscar una subcadena dentro de la cadena, sustituirla por otra, dividirla en varias cadenas atendiendo a un cierto separador, convertir a mayúsculas o minúsculas, etc.

* **Vectores o Arrays:** Los vectores son colecciones de datos de un mismo tipo. También son conocidos popularmente como *arrays* e incluso como "arreglos" (aunque se desaconseja esta última denominación por ser una mala adaptación del inglés).

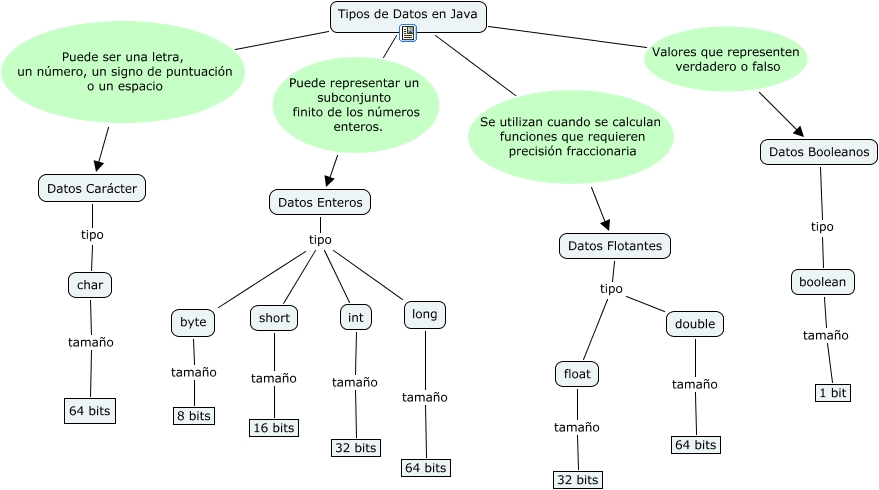
Un vector es una estructura de datos en la que a cada elemento le corresponde una posición identificada por uno o más índices numéricos enteros.

Los elementos de un vector o array se empiezan a numerar en el 0, y permiten gestionar desde una sola variable múltiples datos del mismo tipo.

Por ejemplo, si tenemos que almacenar una lista de 10 números enteros, declararíamos un vector de tamaño 10 y de tipo entero, y no tendríamos que declarar 10 variables separadas de tipo entero, una para cada número.

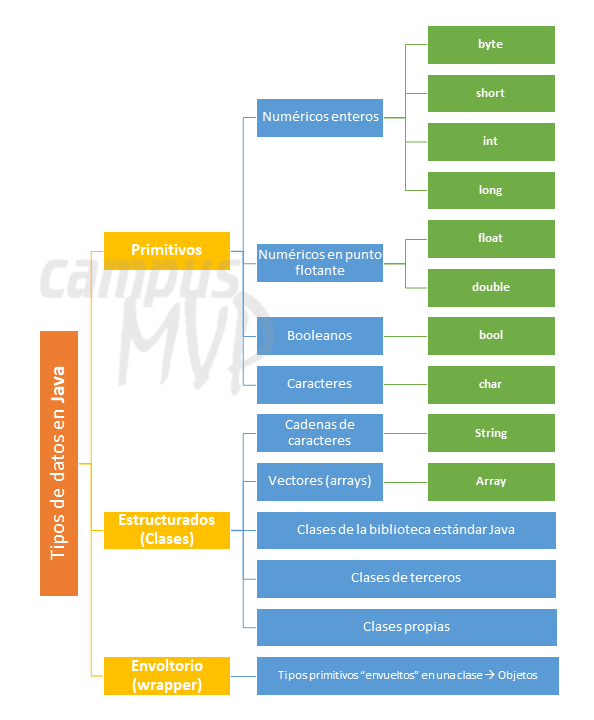
1. **Tipos envoltorios o Wrapper**

**Int 🡪 Integer**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tipo de dato** | | **Representación** | **Tamaño (Bytes)** | | | **Rango de Valores** | | **Valor por defecto** | **Clase Asociada** |
| **byte** | Numérico Entero con signo | | | 1 | -128 a 127 | | | 0 | Byte |
| **short** | Numérico Entero con signo | | | 2 | -32768 a 32767 | | 0 | | Short |
| **int** | Numérico Entero con signo | | | 4 | -2147483648 a 2147483647 | | | 0 | Integer |
| **long** | Numérico Entero con signo | | | 8 | -9223372036854775808 a 9223372036854775807 | | | 0 | Long |
| **float** | Numérico en Coma flotante de precisión simple Norma IEEE 754 | | | 4 | ± 3.4x10-38 a ± 3.4x1038 | | | 0.0 | Float |
| **double** | Numérico en Coma flotante de precisión doble Norma IEEE 754 | | | 8 | ± 1.8x10-308 a ± 1.8x10308 | | | 0.0 | Double |
| **char** | Carácter Unicode | | | 2 | \u0000 a \uFFFF | | | \u0000 | Character |
| **boolean** | Dato lógico | | | - | true ó false | | | false | Boolean |
| **void** | - | | | - | - | | | - | Void |

Fuente: <http://puntocomnoesunlenguaje.blogspot.com/2012/04/tipos-de-datos-java.html>

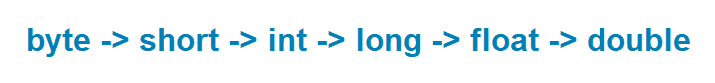


Fuente: <https://www.campusmvp.es/recursos/post/variables-y-tipos-de-datos-en-java-tipos-simples-clases-y-tipos-envoltorio-o-wrapper.aspx>

**CONVERSIÓN DE DATOS o CASTEO**

Cuando asigna valor de un tipo de datos a otro, los dos tipos pueden no ser compatibles entre sí. Si los tipos de datos son compatibles, entonces Java realizará la conversión (automáticamente) que se conoce como **Conversión automática de tipos** y, en caso negativo, deberá hacer un ***casting*** o convertirlo explícitamente. Por ejemplo, asignando un valor a una variable

* **Conversión automática**: funciona cuando pasamos de un tipo de dato de un tamaño menor a uno mayor, es decir, que tiene la capacidad de absorberlo. Por ejemplo, en Java, los tipos de datos numéricos son compatibles entre sí, pero no se admite la conversión automática de tipo numérico a char o boolean. Además, **char y boolean no son compatibles entre sí**.



* **Conversión explicita o “casting”:** Si queremos asignar un valor de tipo de dato más grande a un tipo de dato más pequeño, realizamos un casteo/casting o lo que se conoce como conversión de tipo explícito. Esto es útil para tipos de datos incompatibles donde la conversión automática no se puede realizar. Char y int no son compatibles entre sí.

Ejemplo:

// Programa Java para ilustrar el tipo de conversion explicita

class Test

{

public static void main(String[] args)

{

double d = 100.04;

//casting de tipo

long l = (long)d;

//casting de tipo

int i = (int)l;

System.out.println("Valor Double "+d);

//parte fraccionaria perdida

System.out.println("Valor Long "+l);

//parte fraccionaria perdida

System.out.println("Valor Int "+i);

}

}

Salida:

Valor Double 100.04

Valor Long 100

Valor Int 100

**Conversión de tipo Explícit**

**Fuente:** [**https://lecasabe.com/conversion-de-tipos-en-java/**](https://lecasabe.com/conversion-de-tipos-en-java/)

byte a = 20;  
int x = (int) a;

Al escribir int entre paréntesis se fuerza a cambiar el dato de tipo byte a int.

**String a Integer**

Integer entero = Integer.valueOf(cadena);  
o  
int entero = Integer.parseInt(cadena);

Ejemplo:  
Integer a = Integer.valueOf("900");  
o  
int b = Integer.parseInt("900");

**Integer a String**

String cadena = Integer.toString(entero);  
o  
String cadena = String.valueOf(entero);

Ejemplo:  
String importe = Integer.toString(900);  
o  
int entero = 900;  
String mensaje = String.valueOf(entero);

**char a String**

String cadena = Character.toString(char);

Ejemplo:  
char codigo = 'A';  
String cadena = Character.toString(codigo);

**String a char**

char caracter = cadena.charAt(0); //Solo primer caracter

Ejemplo:  
String codigo= "E";  
char caracter = cadena.charAt(0);

**String a Double**

double doble = Double.parseDouble(cadena;  
Ejemplo:  
double doble = Double.parseDouble("900.1");

**Double a String**

String cadena = String.valueOf(doble);  
Ejemplo:  
double totalDoble = 900.5;  
String totalString = String.valueOf(doble);

**String a Float**

float flotante = Float.parseFloat(cadena);

Ejemplo:  
float importe = Float.parseFloat("900.5");

**Float a String**

String cadena = Float.toString(flotante);

Ejemplo:  
String total = Float.toString(900.1f);

**String a Boolean**

Boolean boolean = Boolean.valueOf(cadena);  
o  
boolean boolean = Boolean.parseBoolean(cadena);

Ejemplo:  
Boolean boolean = Boolean.valueOf("true");  
o  
boolean boolean = Boolean.parseBoolean("false");

**Boolean a String**

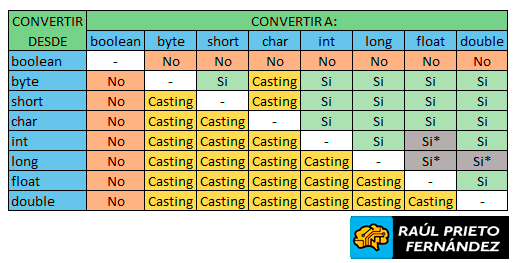
String cadena = String.valueOf(b);  
o  
String cadena = Boolean.toString(b);

Ejemplo:  
boolean b = true;  
String cadena = String.valueOf(b);  
o  
boolean b = false;  
String cadena = Boolean.toString(b);

**MÉTODO VALUEOF PARA CONVERSIÓN DE TIPOS**

El método valueOf es un método sobrecargado aplicable a numerosas clases de Java y que permite realizar conversiones de tipos. Veamos algunos ejemplos de uso.

|  |  |
| --- | --- |
| **EXPRESIÓN** | **INTERPRETACIÓN aprenderaprogramar.com** |
| **miInteger = miInteger.valueOf (i)** | Con i entero primitivo que se transforma en Integer |
| **miInteger = miInteger.valueOf (miString)** | El valor del String se transforma en Integer |
| **miString = miString.valueOf (miBooleano)** | El booleano se transforma en String “true” o “false” |
| **miString = miString.valueOf (miChar)** | El carácter (char) se transforma en String |
| **miString = miString.valueOf (miDouble)** | El double se transforma en String. Igualmente aplicable a float, int, long. |



Fuente: <https://www.raulprietofernandez.net/blog/programacion/conversion-de-tipos-de-datos-en-java>