CARACTERÍSTICAS PRINCIPALES DEL LENGUAJE JAVA. SINTÁXIS Y ELEMENTOS PRINCIPALES.

CHARACTER SET: EL CONJUNTO DE CARACTERES DE JAVA

Para escribir un programa con java, como en cualquier otro lenguaje de programación, utilizamos caracteres. Como es lógico, hay una serie de restricciones y no podemos utilizar cualquier carácter que queramos a la hora de programar. Cada lenguaje determina el conjunto de caracteres (character set) que se pueden utilizar para escribir código. Java soporta el sistema de codificación UNICODE DE 16 BITS y esto supone que el conjunto de caracteres del lenguaje sea muy amplio.

Letras mayúsculas y minúsculas de la A(a) a la Z(z) de los alfabetos internacionales. Los caracteres ñ y Ñ son válidos.

Dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Caracteres ' _ ' '$' y cualquier otro carácter considerado como letra en el sistema de codificación Unicode.

Caracteres especiales y signos de puntuación:

+ - * / = % & # ! ? ^ “ ‘ ~ \ | < > ( ) [ ] { } : ; . ,

Separadores: Espacio, tabulador, salto de línea **(ayudan a que el programa sea más legible)**:

Secuencias de escape:

Secuencia	**Descripción**
\n	Salto de línea. Sitúa el cursor al principio de la línea siguiente
\b	Retroceso. Mueve el cursor un carácter atrás en la línea actual.
\t	Tabulador horizontal. Mueve el cursor hacia adelante una distancia determinada por el tabulador.
\r	Ir al principio de la línea. Mueve el cursor al principio de la línea actual.
\f	Nueva página. Mueve el cursor al principio de la siguiente página.
\"	Comillas. Permite mostrar por pantalla el carácter *comillas dobles*.
\'	Comilla simple. Permite mostrar por pantalla el carácter *comilla simple*.
\\	Permite mostrar por pantalla el carácter barra inversa.
\udddd	Unicode. Cada **d** representa uno de los 4 dígitos hexadecimales del carácter Unicode a mostrar

```
public class Ejemplo {
    public static void main( String[] args )
    {
        System.out.println("\n\n\n");          //escribe 3 saltos de línea
        System.out.println("\t\t\t\uac43");  //escribe 3 tabuladores y el carácter UNICODE AC43
        System.out.println("\n\n\n");
    }
}
```

IDENTIFICADORES Y PALABRAS RESERVADAS JAVA

Los identificadores son los nombres que el programador/a asigna a los elementos propios que introduce en un programa JAVA: **variables, constantes, clases, métodos, paquetes, etc.**

- Están formados por letras y dígitos (incluidas ñ y Ñ).
- No pueden empezar por un dígito.
- No pueden contener caracteres especiales.
- No puede ser una palabra reservada de Java.

Java diferencia mayúsculas y minúsculas, por lo tanto, **edad** y **Edad** son identificadores distintos.

Un IDE como NetBeans o Eclipse nos ayudará, y avisará, cuando cometamos errores a la hora de identificar/nombrar incorrectamente nuestros elementos en los programas JAVA. En general los identificadores han de ser lo más descriptivos posible.

Las palabras reservadas son identificadores predefinidos que forman parte del lenguaje JAVA, tienen un significado para el compilador y por tanto no pueden usarse como identificadores creados por el usuario en los programas.

abstract	**continue**	**for**	**new**	**switch**
assert	**default**	**goto**	**package**	**synchronized**
boolean	**do**	**if**	**private**	**this**
break	**double**	**implements**	**protected**	**throw**
byte	**else**	**import**	**public**	**throws**
case	**enum**	**instanceof**	**return**	**transient**
catch	**extends**	**int**	**short**	**try**
char	**final**	**interface**	**static**	**void**
class	**finally**	**long**	**strictfp**	**volatile**
const	**float**	**native**	**super**	**while**

TIPOS DE DATOS. Variables y Constantes.

Un programa informático es un conjunto de instrucciones que realizan operaciones con datos, y los datos vienen del mundo real. Pueden ser **numéricos,** cómo el **sueldo** de una persona o su **saldo** en cuenta bancaria. **Alfanuméricos,** como su **nombre** o su **dni.** De tipo **lógico,** true/false como, ¿es mayor de edad?. También pueden ser compuestos como los **arrays**, u **objetos** como iremos viendo a lo largo del curso.

Para que un sistema informático o dispositivo digital pueda manejar todos esos tipos de datos hay que adaptar su formato al digital (010100101) y disponer de "contenedores" de diferentes tipos para poder procesar información de cualquier naturaleza. **Hay 8 tipos de datos primitivos en JAVA.** En el futuro los reconoceremos de forma sencilla a la hora de hacer declaraciones, pues los tipos primitivos empiezan por **letra minúscula. Sin embargo las clases u objetos compuestos se nombran siempre con la 1ª mayúscula.**

TIPOS DE DATOS PRIMITIVOS EN JAVA

Tipo	Representación	Tamaño (Bytes)	Rango de Valores	Valor por defecto
byte	Numérico Entero con signo	1	-128 a 127	0
short	Numérico Entero con signo	2	-32768 a 32767	0
int	Numérico Entero con signo	4	-2147483648 a 2147483647	0
long	Numérico Entero con signo	8	-9223372036854775808 a 9223372036854775807	0
float	Numérico en Coma flotante precisión simple	4	$3.4x10^{-38}$ a $3.4x10^{38}$	0.0
double	Numérico en Coma flotante de precisión doble	8	$1.8x10^{-308}$ a $1.8x10^{308}$	0.0
char	Carácter Unicode	2	\u0000 a \uFFFF	\u0000
boolean	Dato lógico	-	true ó false	false

Declaración de variables de los tipos primitivos en Java: tipo nombreVariable;

int edad; double saldo; char sexo;

Asignación de valores (COMPATIBLES) a las variables: nombreVariable = valor;

edad=22; saldo=23456.56; sexo='M'

Declaración + Asignación en la misma instrucción: tipo nombreVariable = valor;

int edad=22; float saldo=23456.56; char sexo='M';

Declaraciones y/o Asignaciones múltiples:

int x = 5, y = 6, z = 50;

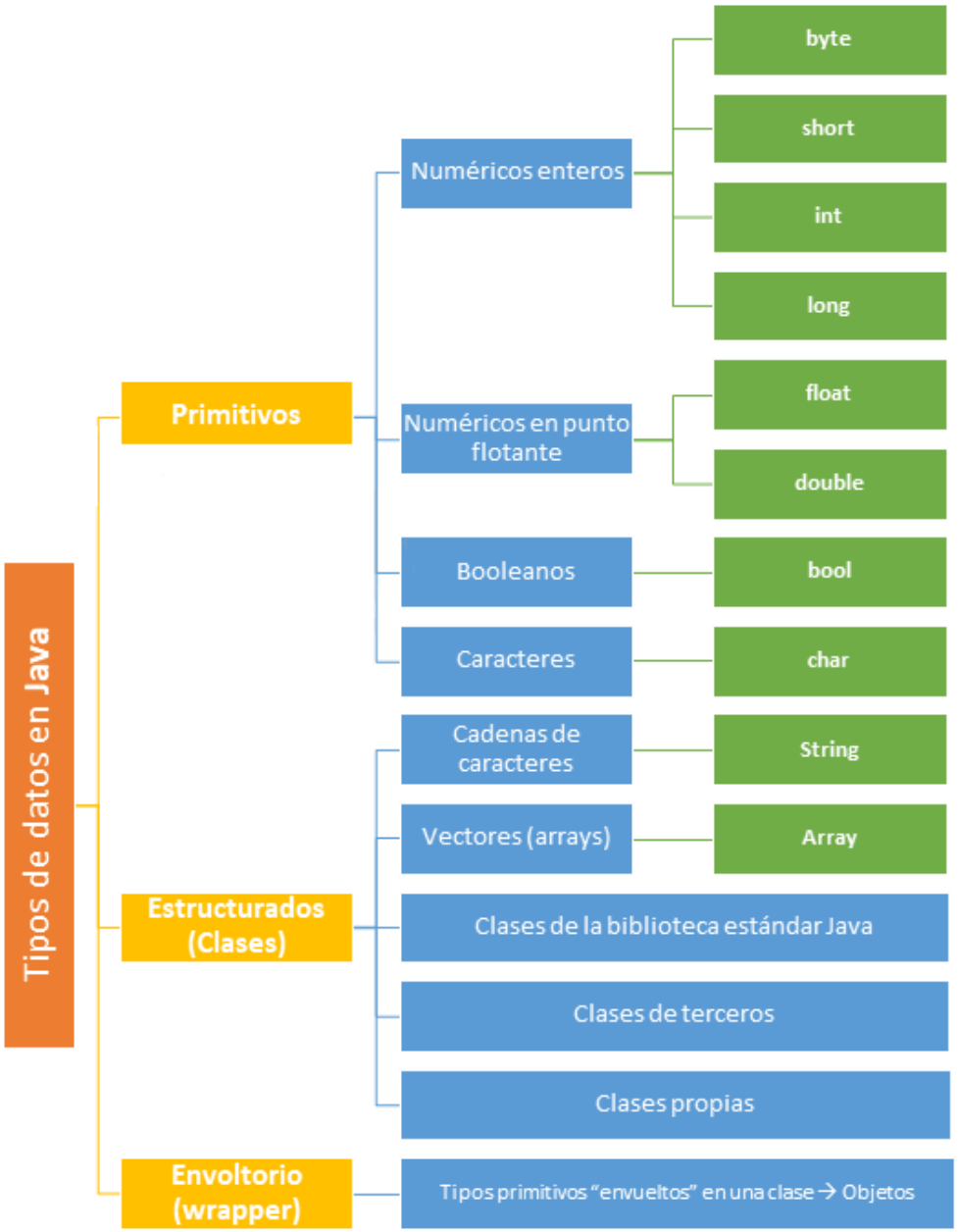
int x,y,z;
x=y=z=50;

Además de los tipos **primitivos**, en JAVA hay una **gran cantidad de tipos Estructurados o complejos** que podremos usar en nuestros programas. Serán tipos que el API de JAVA o librerías de terceros ponen a nuestra disposición, o que nosotros mismos vamos a definir para nuestros proyectos.

También son NO primitivos los Strings, Arrays y Wrappers (envoltorios) de los tipos primitivos

Detalles a destacar:

- Los tipos primitivos empiezan con **minúscula**.
- Los nombres de clases y tipos no primitivos comienzan con **mayúscula**.
- **Todo identificador creado por el usuario comienza con minúscula**, pero si es una palabra compuesta se va iniciando con Mayúscula cada nueva parte de la composición: ***saldoEnCuenta***



Declaración de variables de tipo complejo en Java: TipoComplejo nombreVariable;

String dniPaciente; *Scanner sc;*

Asignación de valores (uso de constructores): nombreVariable = new TipoComplejo(parametro);

dniPaciente = new String("10666666r");
sc = new Scanner(System.in);

Declaración + Asignación: TipoComplejo nombreVariable = new TipoComplejo(parametro);

String dniPaciente=new String("10666666r");

Tipos primitivos Vs tipos envoltorio (wrapper)

Tipo Primitivo	envoltorio
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

¿Para qué tener esa aparente duplicidad entre tipos primitivos y tipos envoltorio? Tiene su explicación. Un tipo primitivo es un dato elemental y carece de **métodos***, mientras que un tipo envoltorio es un objeto, es decir, una entidad compleja que dispone de métodos. Por otro lado, de acuerdo con la especificación de Java, es posible que necesitemos utilizar dentro de un programa un objeto que "porte" como contenido un número entero. Desde el momento en que sea necesario un objeto habremos de pensar en un envoltorio, por ejemplo **Integer**. Inicialmente nos puede costar un poco distinguir cuándo usar un tipo primitivo y cuándo un envoltorio en situaciones en las que ambos sean válidos. Seguiremos esta regla: usaremos por norma general tipos primitivos. Cuando para la estructura de datos o el proceso a realizar sea necesario un objeto, usaremos un envoltorio.

**Los tipos complejos tienen funcionalidad propia (métodos) que definen su comportamiento y que podemos invocar para modificar objetos de esos tipos.*

VARIABLES

Los datos que maneja un programa se almacenan en la memoria del ordenador. Para acceder a ellos se utiliza una dirección de memoria o posición dentro de la memoria, que es donde se encuentra el dato.

Para facilitar la referencia a las posiciones de memoria, los programadores no usamos la dirección de memoria, sino un simple identificador que nosotros mismos creamos para dar un nombre descriptivo a esa variable|constante|método|clase, etc.).

edad | saldoEnCuenta | fecha | añoDeNacimiento

- Una variable tiene **Nombre, Tipo y Valor**

Veamos de nuevo cómo se declaran e inicializan variables (visto antes en el apartado de Tipos de Datos Java)

Declaración de variables: Para poder utilizar una variable en un programa, primero tenemos que declararla (asignarle un nombre y un tipo) | tipo nombreVariable;

int edad;
float saldo;
char sexo;

Inicialización de variables: Asignarles un valor inicial | *nombreVariable = valor;*

edad=22; *saldo=23456.56;* *sexo='M'*

Declaración + Asignación | tipo nombreVariable = valor;

int edad=22; *float saldo=23456.56;* *char sexo='M';*

- **IMPORTANTE:** las variables declaradas PERO no inicializadas en JAVA TOMARÁN EL VALOR POR DEFECTO QUE CORRESPONDE A SU TIPO, EXCEPTO cuando están declaradas dentro de un método, incluido el método main(). En ese caso NO se les asigna un valor por defecto, y es NUESTRA responsabilidad asignarles valores iniciales válidos.
- Las variables que son atributos de una clase sí toman valores iniciales por defecto si no las inicializamos.

CONSTANTES

Un programa puede contener ciertos valores que no deben cambiar durante su ejecución. Estos valores se llaman **constantes.**
Una constante en Java se declara de forma similar a una variable, añadiendo a su declaración la palabra **final** antes del tipo

```
final float impuestoEmisionesClase2 = 14.75;
final int diasEnero = 31;
```

CONVERSION DE TIPOS EN JAVA. CASTING.

Como se ha comentado en repetidas ocasiones, java es un lenguaje muy estricto con los tipos de datos (fuertemente tipado). Por tanto, hay que tener mucho cuidado con los datos que asignamos a cada variable. Si no coinciden exactamente con el tipo declarado para esa variable, **se producirá un error**. A veces esto resulta un poco complicado de manejar, pero hay que acostumbrarse y también entender y manejar la conversión entre tipos **(casting).**

Se produce **casting** (conversión) cuando asignamos a un valor/dato de un tipo concreto a una variable que hemos declarado de otro tipo distinto. Hay 2 tipos de Casting en Java.

- **Widening Casting (automático)** – No nos va a generar ningún problema pues es cuando queremos asignar un dato de un tipo más pequeño en tamaño a una variable de un tipo más grande:

  byte -> short -> char -> int -> long(L) -> float(F) -> double(D)

  De izquierda a derecha. No pasa nada por asignar un valor de un tipo "menor" a una variable de tipo "mayor", pues Java tiene suficiente memoria reservada.

```
public class Wcasting {
  public static void main(String[] args) {
    int numInt = 100;
    double numDouble = numInt; // Se produce casting automático - el entero se convierte en Double

    System.out.println(numInt);
    System.out.println(numDouble);
  }
}
```

  La salida será 100 y 100.0

- **Narrowing Casting** (manual) – Tenemos que aprender a manejarlo o se nos producirán errores. Es cuando queremos asignar un dato de un tipo más grande a una variable de un tipo más pequeño.

  Double(D) -> float(F) -> long(L) -> int -> char -> short -> byte
  Ahora si va a ser un problema asignar un valor de un tipo "mayor" a una variable de tipo "menor", pues Java **NO** tiene suficiente memoria reservada.

```
public class Ncasting {
  public static void main(String[] args) {
    double numDouble = 2500.4356d;
    // int numInt = numDouble; Intentar esa asignación daría error - un double es mayor que un int

    int numInt = (int) numDouble; // con casting funciona, convirtiendo en int el dato de tipo double
    System.out.println(numDouble);
    System.out.println(numInt);
  }
}
```

  La salida será 2500.4356 y 2500

OPERADORES JAVA

ARITMÉTICOS

+	-	*	/	% (RESTO)

- En las operaciones aritméticas donde intervienen variables de tipo char, el valor utilizado para realizar el cálculo es el valor numérico del carácter correspondiente según la tabla ASCII o UNICODE

- En aquellas operaciones en las que intervienen datos de distinto tipo, JAVA convierte los valores al tipo de dato de mayor precisión de todos los datos que intervienen y este será el tipo del resultado. Esta conversión se realiza de forma temporal, solamente para realizar la operación. Los tipos de datos originales permanecen igual después de la operación. **Es muy importante tenerlo en cuenta para poder asignar el resultado de la operación a una variable del tipo correcto.**

  Los tipos short, byte y char se convierten automáticamente a int.

RELACIONALES

<	<=	>	>=	!=	==

Comparan valores, y dan como resultado de la comparación **true** (true) ó **false** (false). **Serán muy importantes en las estructuras de programación condicionales y en los bucles.**

En los **RELACIONALES**, los valores comparados tienen que ser de tipo primitivo. Si intentamos comparar **objetos**, no se comparan los objetos como tal, sino sus referencias en memoria HEAP.

LÓGICOS

&&	||	!

Los operadores **LÓGICOS** se utilizan con valores de tipo **boolean** (true|false). Junto con los relacionales, se utilizan para construir expresiones lógicas cuyo resultado será **true/false**. Son fundamentales para construir condiciones complejas en las estructuras de programación condicionales y bucles.

&& (Operación lógica AND). El resultado es true sólo si los dos valores son true.
|| (Operación lógica OR). El resultado es false sólo si los dos valores son false.
! (NEGACIÓN). Se aplica sobre un solo elemento, y cambia su valor.

A	B	A **&&** B
F	F	F
F	V	F
V	F	F
V	V	V

A	B	A **||** B
F	F	F
F	V	V
V	F	V
V	V	V

A	**!** A
F	V
V	F

&& → obligará al cumplimento de varias condiciones para obtener un resultado **true** en una condición compuesta … " tener > 18 años && estatura > 1,70 "
|| → permitirá varias alternativas para cumplir una condición compuesta y obtener un resultado **true** … "tener > 18 años || permiso paterno "

Ejemplos. Partimos de 3 variables, con sus valores iniciales:

```
int i = 7;
float f = 5.5F;
char c = 'w';
```

i >= 6 && c != 'w'	false
i >= 6 || c != 'w'	true
f < 10 && i > 100	false
!(c != 'p') || i % 2 == 0	false
i + f <= 10	false
i >= 6 && c == 'w' && f == 5	false
c != 'p' || i + f <= 10	true

OPERADORES UNITARIOS.

– +	**Signos Positivo/Negativo**
++ --	**Incremento/Decremento**
~	**Complemento a 1**
!	**NOT (Negación)**

El **operador ++** (incremento), incrementa en 1 el valor de la variable.

```
int i = 1;
i++;  //  incrementa en 1 la variable i. Es lo mismo que hacer i = i + 1 (El nuevo valor de i es 2)
```

El **operador --** (decremento), decrementa en 1 el valor de la variable.

```
int i = 1;
i--; // decrementa en 1 la variable i. Es lo mismo que hacer i = i – 1 (El nuevo valor de i es 0)
```

Los operadores incremento y decremento pueden aparecer antes o después de la variable.

i++ | ++i | i-- | --i En una expresión:

- Si precede al operando (++i | --i), el valor de i se modificará **antes de que se evalúe la expresión** en cuestión.
- Si sigue al operando (i++ | i--), entonces el valor del operando se modificará **después de evaluar la expresión** en cuestión.

```
int x, i = 3;
x = i++;
x vale 3, i vale 4.
```

```
int x, i = 3;
x = ++i;
x contiene 4, i contiene 4.
```

```
int i = 1;
System.out.println(i);
System.out.println (++i);
System.out.println (i++);
System.out.println(i);

1
2
2
3
```

OPERADORES DE ASIGNACIÓN

- Se utilizan para asignar el valor de una expresión a una variable.
- Los valores deben ser de tipo primitivo.

=	Asignación
+=	Suma y asignación
–=	Resta y asignación
***=**	Producto y asignación
/=	División y asignación
%=	Resto de la división entera y asignación
<<=	Desplazamiento a la izquierda y asignación
>>=	Desplazamiento a la derecha y asignación
>>>=	Desplazamiento a la derecha y asignación rellenando con ceros
&=	AND sobre bits y asignación
|=	OR sobre bits y asignación
^=	XOR sobre bits y asignación

Ejemplos de asignaciones:

x += 3; -> equivale a escribir x = x + 3;
y *= 3; -> equivale a escribir y = y * 3;

Dadas las siguientes variables:

```
int i = 5, j = 7, x = 2, y = 2, z = 2;
float f = 5.5F, g = -3.25F;
```

i += 5	i = i + 5	10
f –= g	f = f – g	8.75
j *= (i – 3)	j = j * (i – 3)	14
f /= 3	f = f / 3	1.833333
i %= (j - 2	i = i % (j – 2)	0

En general, en una asignación del tipo A = B, si el rango de valores que puede almacenar A es mayor que el rango de valores que puede almacenar B, entonces la asignación se podrá realizar. Por ejemplo, si A es de tipo double y B de tipo float la asignación A = B se puede hacer **pero no** se podrá realizar al revés sin casting. En Java están permitidas las asignaciones múltiples.

Ejemplo: a = b = c = 3; equivale a: a = 3; b = 3; c = 3;

OPERADOR TERNARIO (CONDICIONAL)

Se puede utilizar en sustitución de la sentencia de control if-else, pero hace las instrucciones menos claras. (Lo veremos más en detalle cuando hablemos de las sentencias de programación condicionales)

?:	expresión1 **?** expresión2 **:** expresión3

```
int i = 10, j;
j = (i < 0) ? 0 : 100;
```

Es equivalente a escribir:

```
if(i < 0)
  j = 0;
else
  j = 100;
```

PRIORIDAD Y ORDEN DE EVALUACIÓN DE LOS OPERADORES EN JAVA

Operadores Java ordenados de mayor a menor prioridad:

Operador	Prioridad
() []	Izquierda a derecha
++ -- ~ !	Derecha a izquierda
new	Derecha a izquierda
* / %	Izquierda a derecha
+ -	Izquierda a derecha
>> >>> <<	Izquierda a derecha
> >= < <=	Izquierda a derecha
== !=	Izquierda a derecha
&	Izquierda a derecha
^	Izquierda a derecha
|	Izquierda a derecha
&&	Izquierda a derecha
||	Izquierda a derecha
?:	Derecha a izquierda
= += -= *= ...	Derecha a izquierda

```
int c = (((25 - 5) * 4) / (2 - 10)) + 4;
int c = ((20 * 4) / (2 - 10)) + 4;
int c = (80 / (2 - 10)) + 4;
int c = (80 / -8) + 4;
int c = -10 + 4;
int c = -6;
```