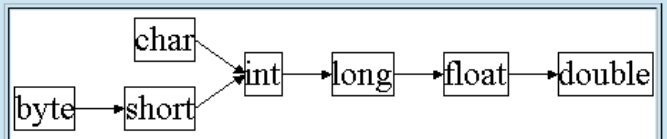
Tipos

Tipos PRIMITIVOS (no son objetos. Java es Hibrido)
Son SIEMPRE IGUALES (no cambian con las plataformas)

Tipos primitivos		
Enteros [complemento a dos con signo]	byte	8 bits
	short	16 bits
	int	32 bits
	long	64 bits
Reales [IEEE 754]	float	32 bits
	double	64 bits
Caracteres [Unicode]	char	16 bits
Booleanos [dpte. implementación]	boolean	
no-tipo		
void		



Conversiones entre tipos primitivos. El sentido de la flecha indica las conversiones "directas". Las "inversas" tambien se permiten con la utilización de "casting".

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

Tienen sus equivalentes como objetos.
(Hay otros tipos sólo como objetos, p.ej. Binario, precisión infinita, etc.)

```
float f;
double g=3.14159;
f=(float)g;

long l=32;          //la constate 32 es int y se convierte automáticamente a long
char c=(char)l;
```

```
public class MaxVariablesDemo {
   public static void main(String args[]) {

      // enteros
      byte maximoByte = Byte.MAX_VALUE;
      short maximoShort = Short.MAX_VALUE;
      int maximoInteger = Integer.MAX_VALUE;
      long maximoLong = Long.MAX_VALUE;

      // reales
float maximoFloat = Float.MAX_VALUE;
      double maximoDouble = Double.MAX_VALUE;

      // otros tipos primitivos
      char unChar = 'S';              //existe Character
      boolean unBoolean = Boolean.TRUE;  //tambien válido: boolean
unBooleano=true

      // (aquí continuará la definición de la clase)
   }
}
```

Identificadores y literales

Identificadores

{letra|_|$}{letra|digito|_|$}*

letra ::= cualquier carácter de escritura en cualquier idioma.
Digito::= [0|1|2|3|4|5|6|7|8|9]
Los identificadores no pueden coincidir con palabras reservadas.

[] ≡ opcional {a|b} ≡ a ó b * ≡ repetible (cero o más veces)

Literales

Entero: {[+]|-} [0[x|X|b|B]] constante_entera [l|L]

"0" indica constante expresada en octal
"0x" o "0X" indica constante expresada en hexadecimal
"0b" o "0B" indica constante expresada en binario
"l" o "L" indica tamaño "long" (por defecto el tamaño es 32 bits)

constante_entera ::= digito digito*

Ejemplos de enteros:

101	+101	
-101		
101l	101L	(101 en 64 bits)
0101		(65 en octal)
0x101	0X101	(257 en hexadecimal)
0b101	0B101	(5 en binario)

Real: {[+]|-} parte_entera . parte_fraccionaria [{e|E}{[+]|-} exponente] [f|F][d|D]

"f" o "F" indica tamaño "float"
"d" o "D" indica tamaño "double" (tamaño por defecto)

parte_entera, parte_fraccionaria, exponente ::= constante_entera

Ejemplos de reales:

37.091	+37.091	37.091D
37.091e12	37.091+e12	+37.091+012
-37.091e12	-37.091+e12	-37.091+012
37.091e-12	37.091-e12	+37.091-012
37.091f	(37.091 en 32 bits)	

Booleano: {true|false}

Caracteres: UNICODE ejemplos... (char c='x') (char c='\n') (char c='\u001C')

(Estos dos elementos son del "mundo de los objetos")

Cadenas: ejemplos... (String s="hola") (String s="\"hola\"") ← Comentario en siguiente página
Objeto nulo: ejemplo... (String s=null)

https://docs.oracle.com/javase/specs/jls/se7/html/jls-3.html

Java es sensible a la capitalización, y no pone límites a la longitud de los identificadores.

Sobre estas características se "acuerdan" numerosas convenciones (no las exige el JDK ni los IDEs, pero las siguen los desarrolladores), p.ej. «`esto`» es un objeto, «`Esto`» es una clase, «`setElement`» es una rutina que tiene por función dar valor a un objeto o variable « `element` », etc.

Arrays. (y algo de syntactic sugar… o no.)

Como cualquier otro lenguaje algorítmico, Java tiene la posibilidad de manejar Arrays, es decir estructuras que almacenan de forma contigua un determinado número de elementos del mismo tipo o clase. Nótese que en los lenguajes clásicos existe otra estructura capaz de almacenar un grupo de elementos heterogéneos (denominada "struct" en C o "record" en Pascal) y que no existe en Java ya que es sustituida y ampliada con el concepto de clase.

Los arrays en Java son en realidad objetos, pero el lenguaje introduce una determinada sintaxis que permite realizar ciertas acciones de un modo natural y/o eficaz (“syntactic sugar”)

```
// Declaración estilo C vs. estilo Java
int   array1[];       //estilo C (y sólo declarado)
int[] array2=null;    //estilo Java (declarado y definido como "No inicializado")

// Generación con tamaño predeterminado en tiempo de compilación
int   array3[]=new int[10]; //el array ya existe, su contenido está indeterminado
int   array4[]={1,2,3,5,7,11}; //el array existe, y su contenido está determinado

// El tamaño puede determinarse en tiempo de ejecución
array2=new int[2*numeroDeParejas()]; //el array ya existe, su contenido está indeterminado

// Asignacion y lectura
array3[5]=7;
int n=array3[5];

// Arrays multidimensionales
int[][] array2D1;
int[][] array2D2=new int[10][];
int[][] array2D3=new int[10][3];
int[][] array2D4={{1,2,3,4,5,6},{2,4,6},{3,6}};

// Asignaciones en Arrays multidimensionales
array2D2[5]=new int[3]; //generacion de una segunda dimension
array2D2[5][1]=7;       //asignación de un valor concreto

// Arrays anónimos (generación en tiempo de ejecución)
array3=      new int[]{6,28,496,8128}; //p.ej. en asignación
```

El campo “length”

```
class EchoParameters {
  public static void main(String[] args)
  {
    for (int i=0; i<args.length;i++)
      System.out.println(args[i]);
  }
}
```

*de la página anterior

La asignación de literales a Strings es también en cierto modo “syntactic sugar” porque nos facilita la generación de objetos como si no lo fuesen, aunque hay un detalle a tener en cuenta:

```
String s1="hola", s2="hola";
```

No son dos objetos String iguales, sino un sólo objeto String referenciado por dos identificadores

Más “syntactic sugar”:

```
Double d1=5.0; //boxing
double d2=d1;  //unboxing
```