INTRODUCCIÓN

Java es un lenguaje relativamente moderno creado por la empresa **Sun**. La primera versión se distribuyó públicamente en el año 1996. Su primer uso fue un año antes para la construcción del navegador **HotJava**.

Algunas de sus características:

Su sintaxis se basa en la de **C/C++,** intentando eliminar de estos todo lo que resultase complicado o fuente de errores en estos lenguajes. Por ejemplo, los punteros. Además el compilador Java detecta muchos errores que en **C/C++** solo se detectarían en tiempo de ejecución (o incluso nunca). Un ejemplo típico de error en C: **if(a=b) ...** el compilador Java no nos dejaría compilar este código.

Es un lenguaje orientado a objetos. En **C++** es posible crear un programa si utilizar en absoluto el paradigma de orientación a objetos. En Java todo es un objeto a excepción de los tipos primitivos de datos.

En Java no hay aspectos dependientes de la implementación, todas las implementaciones de Java siguen los mismos estándares en cuanto a tamaño y almacenamiento de los datos. Esto no ocurre así en **C++**, por ejemplo. En éste un entero, por ejemplo, puede tener un tamaño de 16, 32 o más bits, siendo lo única limitación que el entero sea mayor que un **short** y menor que un **long int**.

El código generado por el compilador Java es independiente de la arquitectura: podría ejecutarse en un entorno **UNIX**, **Mac OS** o **Windows**. El motivo de esto es que el que realmente ejecuta el código generado por el compilador no es el procesador del ordenador directamente, sino que este se ejecuta mediante una **máquina virtual**.

En principio la velocidad de procesado de Java es bastante inferior a la de **C**, incluso a **C++**. Esto es debido, entre otras cosas, a que un programa Java no se compila (por defecto) al código máquina de la arquitectura en la que estemos trabajando, sino que se compila a un lenguaje intermedio (**btyecode**) el cual es posteriormente interpretado por la máquina virtual Java.

Es posible, sin embargo, el uso de compiladores **just in time** que traducen los bytecodes de Java en código para una determinada CPU, que no precisa de la máquina virtual para ser ejecutado, y guardan el resultado de dicha conversión, volviéndolo a llamar en caso de volverlo a necesitar, con lo que se evita la sobrecarga de trabajo asociada a la interpretación del bytecode.

No obstante por norma general el programa Java consume bastante más memoria que el programa **C++**, ya que no sólo ha de cargar en memoria los recursos necesario para la ejecución del programa, sino que además debe simular un sistema operativo y hardware virtuales (la máquina virtual).

Por otro lado la programación gráfica empleando las librerías **Swing** es más lenta que el uso de componentes nativos en las interfaces de usuario.

En general en Java se ha sacrificado el rendimiento para facilitar la programación y sobre todo para conseguir la característica de neutralidad arquitectural, si bien es cierto que los avances en las máquinas virtuales remedian cada vez más estas decisiones de diseño.

ELEMENTOS DEL LENGUAJE

PALABRAS RESERVADAS

Son palabras que tienen un significado concreto dentro de Java, y no pueden ser utilizadas para otras funciones.

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

BLOQUES E INSTRUCCIONES

Una instrucción es una línea de código finalizada con el carácter ‘;’

Un bloque de código es un conjunto de instrucciones enmarcadas entre los símbolos ‘{‘ y ‘}’

IDENTIFICADORES

Un identificador es un nombre que se asocia a alguno de los diferentes elementos del lenguaje (variables, métodos, clases, objetos).

Para que un identificador sea válido ha de ajustarse a las siguientes reglas:

1. No coincidir con una palabra reservada.
2. Comenzar por una letra (mayúscula o minúscula), un carácter de subrayado ‘_’, o un símbolo ‘$’.
3. A continuación pueden ir cualquier combinación de dígitos, letras (mayúsculas o minúsculas), símbolos de subrayado ‘_’ y símbolos ‘$’.

Java es un lenguaje "**case-sensitive**", esto es, distingue entre mayúsculas y minúsculas. Por tanto los siguientes identificadores serían distintos para Java:

Hola hola hOla

TIPOS PRIMITIVOS DE DATOS

Java es un lenguaje orientado a objetos "puro", a diferencia de otros como **C++** denominados "híbridos". Estos últimos permiten utilizar el paradigma de orientación a objetos, pero al mismo tiempo permiten desarrollar programas que no hagan uso del mismo. En Java en cambio, siempre se ha de utilizar la orientación a objetos. En principio trabajaremos sólo con objetos, pero hay un caso especial: los tipos primitivos. Estos tipos no son objetos y se utilizan por motivos de eficiencia (al crear un objeto éste se coloca en el montículo, mientras que los tipos primitivos se colocan en la pila). A diferencia de **C/C++**, el tamaño de estos datos es fijo y no depende de la arquitectura sobre la que estemos desarrollando.

Para cada tipo primitivo se dispone de un tipo "**envoltura**", esto es, una clase de objetos que se corresponde con el tipo primitivo.

	tipo	**Tamaño (bits)**	**Tipo envoltura**
Enteros	byte	8	Byte
	short	16	Short
	int	32	Integer
	long	64	Long
Reales	float	32	Float
	double	64	Double
Booleanos	boolean	-	Boolean
Caracter	char	16	Character

Los tipos de datos carácter utilizan la codificación Unicode de 16 bits.

LITERALES

Enteros.

Para representar valores enteros se escribe el número precedido por el signo '+' o '–' (positivo o negativo) En caso de ser positivo el signo es opcional.

Positivo	45
Negativo	-45

Si el número el **long** debe de ir seguido de la letra 'L' (mayúscula o minúscula).

long	45L

Se pueden expresar los números en octal o hexadecimal precediéndolos de '0' ó '0x' respectivamente.

Octal	034	
Hexadecimal	0x45	(la x puede escribirse en mayúsculas o minúsculas)

Reales.

Los números reales pueden escribirse en formato punto flotante o en formato exponencial, en cuyo caso el exponente se indica mediante la letra 'e' (mayúscula o minúscula) y el número correspondiente.

2.3 1.5e3

Si el tipo es **float**, el número debe ir seguido de la letra 'f' (mayúscula o minúscula).

32.64F

Caracteres

El tipo **char** representa un único carácter. Este debe ir incluido entre comillas simples.

´j´

Hay una serie de caracteres especiales, llamados caracteres de escape, cuyo significado es el siguiente:

'\b'	Backspace
'\f'	Form feed
'\n'	New line

'\r'	Carriage return
'\t'	Tabulator
\\	\
\'	'
\"	"

Booleanos

Tienen 2 valores posibles:

```
true
false
```

VARIABLES

Permiten asociar un tipo de dato, con un identificador. Todas las variables han de ser declaradas antes de poder ser utilizadas. Para declararlas se indica el tipo de dato seguido del identificador que deseemos utilizar. Para finalizar la declaración se utiliza el símbolo ';'

```
int   contador;
```

Se pueden declarar varias variables de un mismo tipo separando los identificadores con comas.

```
int  a, b , c;
```

Sería equivalente a:

```
int a;
int b;
int c;
```

Para asignar un valor a una variable se utiliza el operador de asignación (el símbolo '='). Las variables no son inicializadas por defecto (salvo las que son atributos de una clase, que se inicializan a los valores nulos de los tipos primitivos correspondientes).

```
int a;
[…]
a = 15;
```

También pueden ser inicializadas al mismo tiempo que se declaran:

```
int a = 15;
```

Las variables se pueden declarar y/o inicializar en cualquier lugar de un bloque.

Son visibles dentro del bloque en que hayan sido declaradas, desde el punto de su declaración hasta el final del bloque.

CONVERSIÓN DE TIPOS DE DATOS NUMÉRICOS

Si en una operación se involucran varios datos numéricos de distintos tipos, todos ellos se convierten al tipo de dato que permite una mayor precisión y rango de representación numérica; así, por ejemplo:

- Si cualquier operando es **double** todos se convertirán en **double**.
- Si cualquier operando es **float** y no hay ningún **double** todos se convertirán a **float**.
- Si cualquier operando es **long** y no hay datos reales todos se convertirán en **long**.

La "jerarquía" en las conversiones de mayor a menor es:

double ← float ← long ← int ← short ← byte

Estas conversiones sólo nos preocuparán a la hora de mirar en qué tipo de variable guardamos el resultado de la operación; esta ha de ser, al menos, de una jerarquía mayor o igual a la jerarquía de la máxima variable involucrada en la operación. Si es de rango superior no habrá problemas.

Hay ciertas excepciones a la norma aquí descrita: Java solo tiene dos tipos de operadores enteros: uno que aplica para operar datos de tipo **long**, y otro que emplea para operar datos de tipo **int**. De este modo cuando operemos un **byte** con un **byte**, un **short** con un **short** o un **short** con un **byte** Java empleará para dicha operación el operador de los datos tipo **int**, por lo que el resultado de dicha operación será un **int** siempre.

CASTING

Es posible convertir un dato de jerarquía "superior" a uno con jerarquía "inferior", arriesgándonos a perder información en el cambio. Este tipo de operación (almacenar

el contenido de una variable de jerarquía superior en una de jerarquía inferior) se denomina cast o casting.

Para ellos se pone entre paréntesis el tipo destino precediendo al elemento a convertir.

```
int a = (int) 3.0F;
```

EXPRESIÓN (sentencia)

Una expresión o sentencia es una combinación de variables, operadores y literales, que devuelve un valor.

TIOS DE OPERADORES

Asignación

Asigna un valor a una variable:

=

Aritméticos

Realizan operaciones aritméticas devolviendo valores numéricos.

	operador	Significado
unarios	-	Signo negativo
	++	Incremento en una unidad
	--	Decremento en una unidad
binarios	+	Suma
	-	Resta
	*	Multiplicación
	/	División
	%	Módulo (resto de la división)

Booleanos o relacionales

Realizan comparaciones y devuelven un valor booleano.

	operador	Significado
binarios	==	Igual
	!=	Distinto
	<	Menor
	>	Mayor
	<=	Menor o igual
	>=	Mayor o igual

Lógicos

Realizan operaciones lógicas y devuelven un valor booleano.

	operador	Significado
unarios	!	Negación
binarios	&&	Y lógico
	&	Y lógico a nivel de bit
	||	O lógico
	|	O lógico a nivel de bit
	^	O exclusivo

De Desplazamiento

Realizan operaciones de desplazamiento de bits.

	operador	Significado
binarios	<<	Desplazamiento a la izquierda. Rellena con ceros.
	>>	Desplazamiento a la derecha. Rellena con el bit de signo.
	>>>	Desplazamiento a la derecha. Rellena con ceros.

Para utilizarlos se escribe el valor a desplazar seguido del operador, y este seguido del número de bits a desplazar.

16 >> 2 desplaza el 16 dos bit a la izquierda (daría como resultado 4)

Otros operadores de asignación.

El operador de asignación puede combinarse con otros, de modo que se efectúa la operación indicada por el otro operados y a continuación ese resultado es asignado.

Por ejemplo:

```
a += 5;
```

Sería equivalente a:

```
a = a + 5;
```

PRECEDENCIA DE OPERADORES

En caso de que una expresión aparezcan varios operadores, el orden de evaluación de estos será de mayor a menor precedencia. En caso de igual precedencia, se evaluarán de izquierda a derecha. Para cambiar el orden de precedencia se usan los paréntesis.

a + b * c	multiplica b por c, y al resultado le suma a
(a + b) * c	suma a más b, y el resultado lo multiplica por b

Prioridad	Tipo operador	operadores
+	Unarios	+ - ++ --
	Aritméticos, Desplazamiento	* / % + - << >>
	Relacionales	> < >= <= == !=
	Lógicos	&& || & | ^
	Condicional	?:
-	Asignación	= (y compuestas como +=)

(dentro de una misma fila, los de mayor precedencia están situados a la izquierda)

COMENTARIOS

Los comentarios son texto que el programador introduce en el código con el fin de documentar el mismo, pero que no son tenidos en cuenta a la hora de compilar el código del programa. En Java contamos con tres tipos de comentarios.

Comentarios de una línea.

Comienzan por '//' y se considera comentario desde su posición hasta el final de la línea en la que están.

```
// comentario de una línea
```

Comentarios multilínea

Comienzan con '/*' y terminan con '*/' Todo lo que haya en medio será considerado comentario.

```
/* comentario
  Multilínea */
```

Comentarios JavaDoc

Son comentarios multilínea utilizados por la herramienta JavaDoc de documentación. Comienzan por una línea '/**', acaban con una línea '*/' y el comentario se introduce en las líneas intermedias, debiendo comenzar todas ellas por '*'.

Los comentarios JavaDoc pueden incluir unos indicadores especiales, que comienzan por '@' y se suelen colocar al comienzo de línea.

```
/**
*  Comentario
*  JavaDoc
*  @author  Amador
*/
```

OBJETOS

La forma de declarar objetos es similar a la de los tipos primitivos. Se indica la clase de objeto seguida del identificador que deseemos utilizar como referencia al objeto. Todo objeto necesita ser creado antes de usarse, para ello se utiliza el operador 'new'seguido de un constructor de la clase.

Con tipos primitivos	`int a = 15;`
Con objetos	`Integer a = new Integer(15);`

Para las clases envoltorio y la clase String, se puede utilizar una versión simplificada, sin necesidad de utilizar explícitamente el operador 'new':

```
Integer a = 15;
```

STRING

La clase String define objetos cuyo contenido es una cadena de caracteres. Adelantamos algo sobre ella, pues su uso es muy común en cualquier programa.

Los literales String se escriben delimitados entre símbolos de comillas dobles.

```
String  cadena = "un texto";
```

Un operador muy utilizado es el operador de concatenación: '+' Este operador delvuelve una cadena que es concatenación de sus operandos. Si el operando izquierdo es un String, el derecho se convierte a String en caso de no serlo.

```
"1" + "2"        →   "12"
"1" + 2          →   "12"
"1" + 2 + 3      →   "123"
"1" + (2 +3)     →   "15"
```

OPERADORES DE IGUALDAD Y DESIGUALDAD

Un aspecto al que hay que prestar especial atención es el comportamiento de los operadores de igualdad '==' y desigualdad '!=' al operar sobre objetos. Su forma de operar, en general, es diferente a la de los tipos primitivos. En principio lo que hacen es comparar direcciones de memoria, no contenidos. Para comparar contenidos se debe utilizar el método 'equals()'.

```
Clase1 a = new Clase1(23);
Clase1 b = new Clase1(23);
```

a == b devolvería 'false', pues no comparten la misma dirección de memoria.

a.equals(b) devolvería 'true' (depende de la implementación del método).

Para las clases envoltorio, se pueden utilizar los operadores '==' y '!='. Con la clase String también podrían funcionar, pero no es recomendable su utilización.