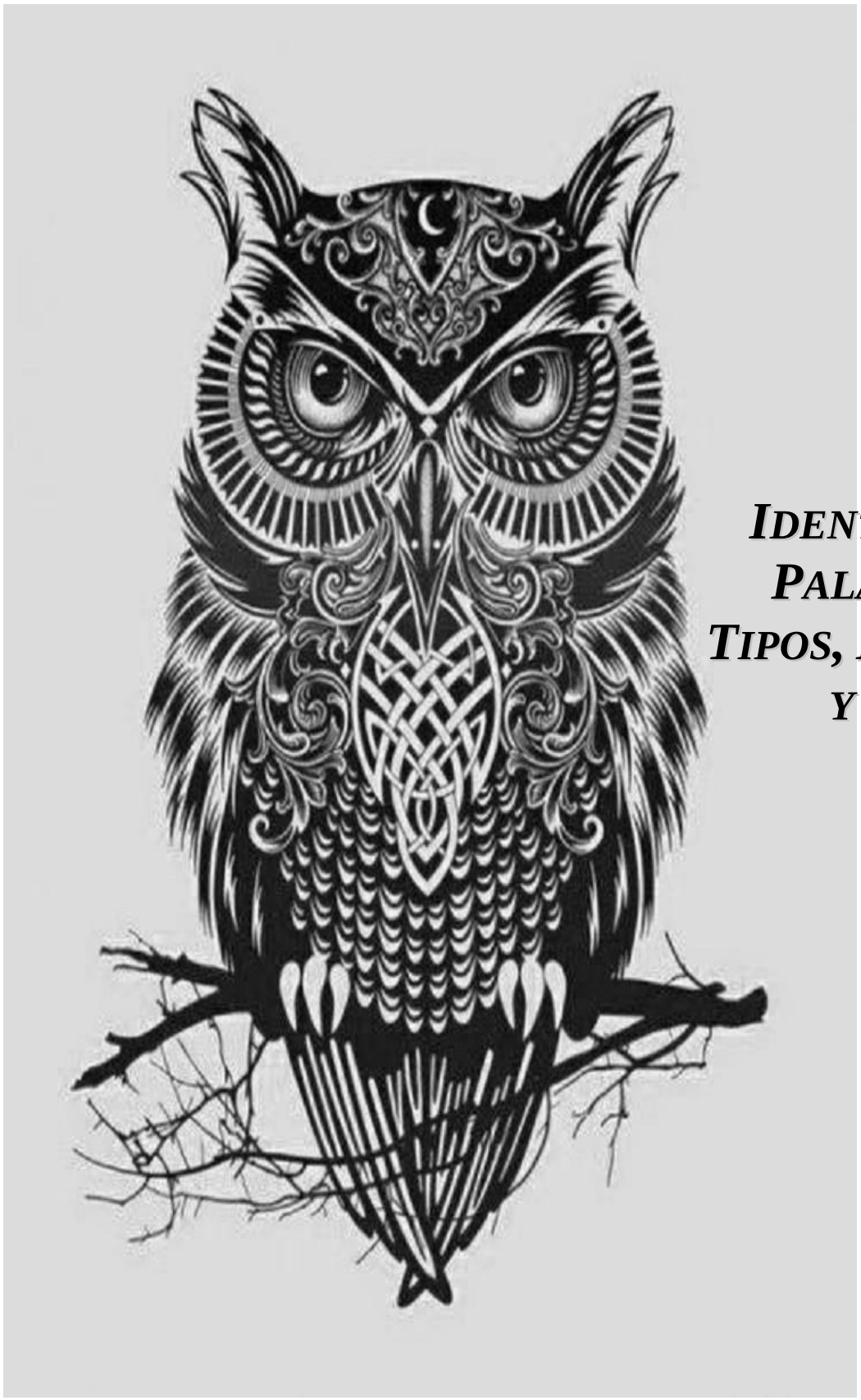
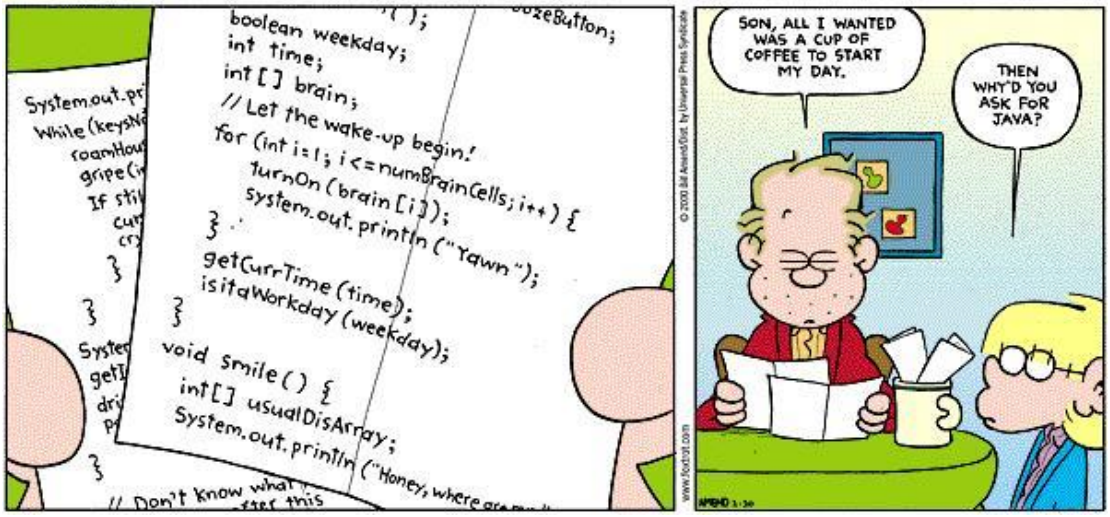


IDENTIFICADORES, PALABRAS CLAVE, TIPOS, EXPRESIONES Y OPERADORES

LUIS ERNESTO
RUBIO TORRES

NOVIEMBRE 2018

Identificadores, Palabras clave, Tipos, Expresiones y Operadores



Juego de Caracteres

Java utiliza un juego de caracteres de 16 bits (2^{16}=65536 posibles caracteres). Lo que significa que puede representar muchos más caracteres que los 256 posibles del código ASCII, usual en la mayoría de los lenguajes de programación utilizados.

Esto significa que se pueden utilizar en los programas Java códigos pertenecientes a diferentes lenguajes. Para facilitar la correspondencia entre los códigos ASCII y los Unicode, los entornos Java actuales leen programas escritos en ASCII estándar o ISO Latin 1.

Como los editores no soportan caracteres Unicode, Java reconoce la *sentencia de escape* \uddd para codificar caracteres Unicode, en la que cada d es un dígito hexadecimal (por ejemplo, el carácter \u2603 es un muñeco de nieve).

Comentarios

Es fundamental ir acompañando de comentarios el código que se escribe, lo que facilita su comprensión y la detección de errores, aunque haya pasado tiempo desde que fue escrito, o fuese otra persona la que lo escribió.

Evidentemente, los comentarios no han de ser considerados por el compilador Java, y están dirigidos exclusivamente para que los programadores entendamos el código.

Un programa sin comentarios funciona perfectamente, pero será más difícil, si no imposible de modificar que otro que sí los tenga. Además, conviene adoptar el buen hábito de comentar el código desde el principio, para que comentar y programar sea una misma cosa.

Los tres tipos de comentarios de Java son:

- // comentario No se tienen en cuenta los caracteresque hay desde // hasta el final de la línea.
- /* comentario */ No se tienen en cuenta los caracteres comprendidos entre /* y */.
- /** comentario */ No se tienen en cuenta los caracteres comprendidos entre /** y */, pero las líneas comentadas de este modo son tratadas por el generador automático de documentación, javadoc. Este tipo de comentarios de documentación deben ir inmediatamente antes de una declaración de clase o constructor.

Java no permite comentarios anidados, por lo que el siguiente trozo de código no compilará:

```
/* esto es un comentario
/* a=3; */
fin del comentario */
```

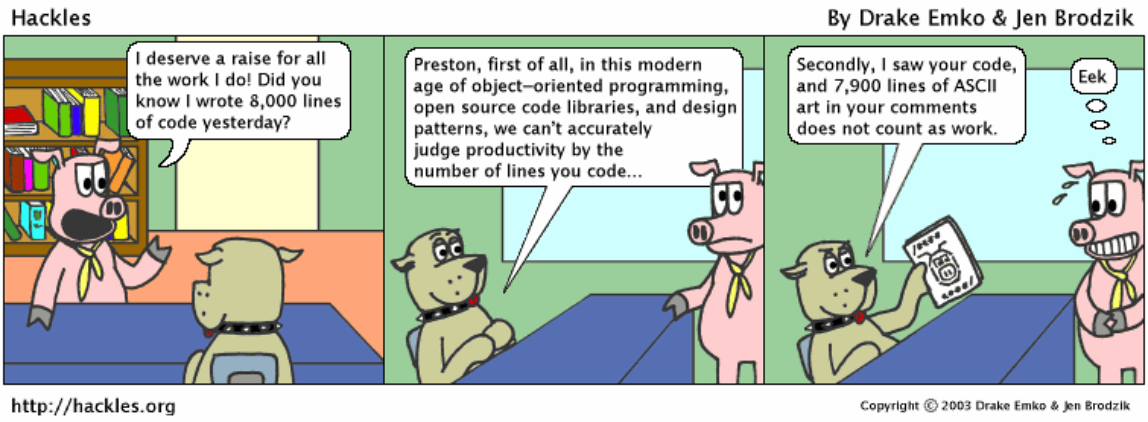
Ya que el primer grupo /* se empareja con el primer grupo */. Para evitar eso es mejor utilizar la doble barra cuando se quiere comentar una línea dentro de un comentario.

```
/* esto es un comentario
// a=3;
fin del comentario */
```

Resulta interesante analizar el siguiente ejemplo de código:

```
/** una de nuestas clases de ejemplo */
class HolaMundo2{
  /** el metodo main de la clase HolaMundo */
  public static void main(String[] args) {
    //String mensaje="Hola otra vez mundo";
    String mensaje="Saludos";
    System.out.println(mensaje);
  }
} // fin de la clase HolaMundo2
```

Sentencias



Una *sentencia* es una línea de código Java. No existe correspondencia entre uno a uno entre las líneas de código y las líneas de texto en un archivo de código fuente. Java utiliza el punto y coma para indicar el final de una sentencia. La línea

a = b + c + d + e + f;

es equivalente a

a = b + c +
d + e + f;

Bloques de Código

Las sentencias se pueden agrupar en bloques para que una sola pueda controlar de manera sencilla la ejecución de muchas otras. Los bloques de código Java están delimitados por llaves '{','}'. Los bloques de código se pueden anidar dentro de otros bloques. Estructura de los Archivos Fuente: Paquetes

Los archivos de código fuente en Java pueden contener sólo tres tipos de sentencias fuera de bloques de código:

- Sentencias package: definen un paquete al que pertenecerán las clases del archivo.
- Sentencias import: establecen un método para referirse a clases que ya han sido creadas, mediante el nombre de clase, sin especificar el nombre del paquete.
- Sentencias class: sirven para definir clases.

Estos tres tipos de sentencias reseñadas no son obligatorias en todos los programas, pero si aparecen deben hacerlo en el orden especificado:package, import y class.

Tokens

Los *tokens* de un lenguaje son sus palabras básicas. Un analizador sintáctico descompone el código fuente en tokens y a continuación trata de averiguar qué sentencias, identificadores y otros elementos componen el código.
En Java, los espacios en blanco (espacios, tabuladores, saltos de línea y saltos de página) no tienen significado; sirven sólo para separar tokens o como contenido de literales de carácter o de cadena de caracteres.

El espacio en blanco es necesario para separar tokens que de otro modo constituirían un token único. Así, por ejemplo, en la sentencia
return 0;

no se puede eliminar el espacio en blanco pues se crearía la sentencia no válida
return0;

El programador debe usar espacios en blanco extra cuando esto facilite la legibilidad del código, tal y como ocurre en la identación:

```
{
  sentencia 1
  sentencia 2 {
    sentencia 2.1
    sentencia 2.2 {
      sentencia 2.2.1
      sentencia 2.2.2
    }
  }
  sentencia 3
}
```

Identificadores

Un identificador es un nombre que se le da a una variable, clase o constante. Los identificadores deben comenzar por carácter alfabético (incluidos el carácter subrayado '_' y el dólar '$'), seguido por cualquier carácter alfabético o numérico (esto considerándolos como caracteres unicode).

Cualquier diferencia en los caracteres de dos identificadores ya les hace diferentes y únicos. Uno de los recursos para diferenciar dos indentificadores es el uso de las mayúsculas y las minúsculas; y aunque no existe ninguna regla, existe una forma de utilizarlas dependiendo del tipo de identificador que se esté construyendo:

Tipo de Identificador	**Convenio**	**Ejemplo**
Identificadores de clases	Cada palabra dentro del identificador comienza por mayúscula	HolaMundo, Automóviles
Identificadores de métodos	Cada palabra dentro del identificador comienza por mayúscula, excepto la primera	insertarLibro, main
Identificadores de variables	Cada palabra dentro del identificador comienza por mayúscula, excepto la primera	numeroVeces, mensaje
Identificadores de constantes	Cada palabra dentro del identificador se escribe en mayúsculas, y se separan por el símbolo subrayado	MAXIMO_TOTAL

Palabras Reservadas (clave) Java

Las palabras clave Java no se pueden usar como identificadores y son las que aparecen en la siguiente lista:

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

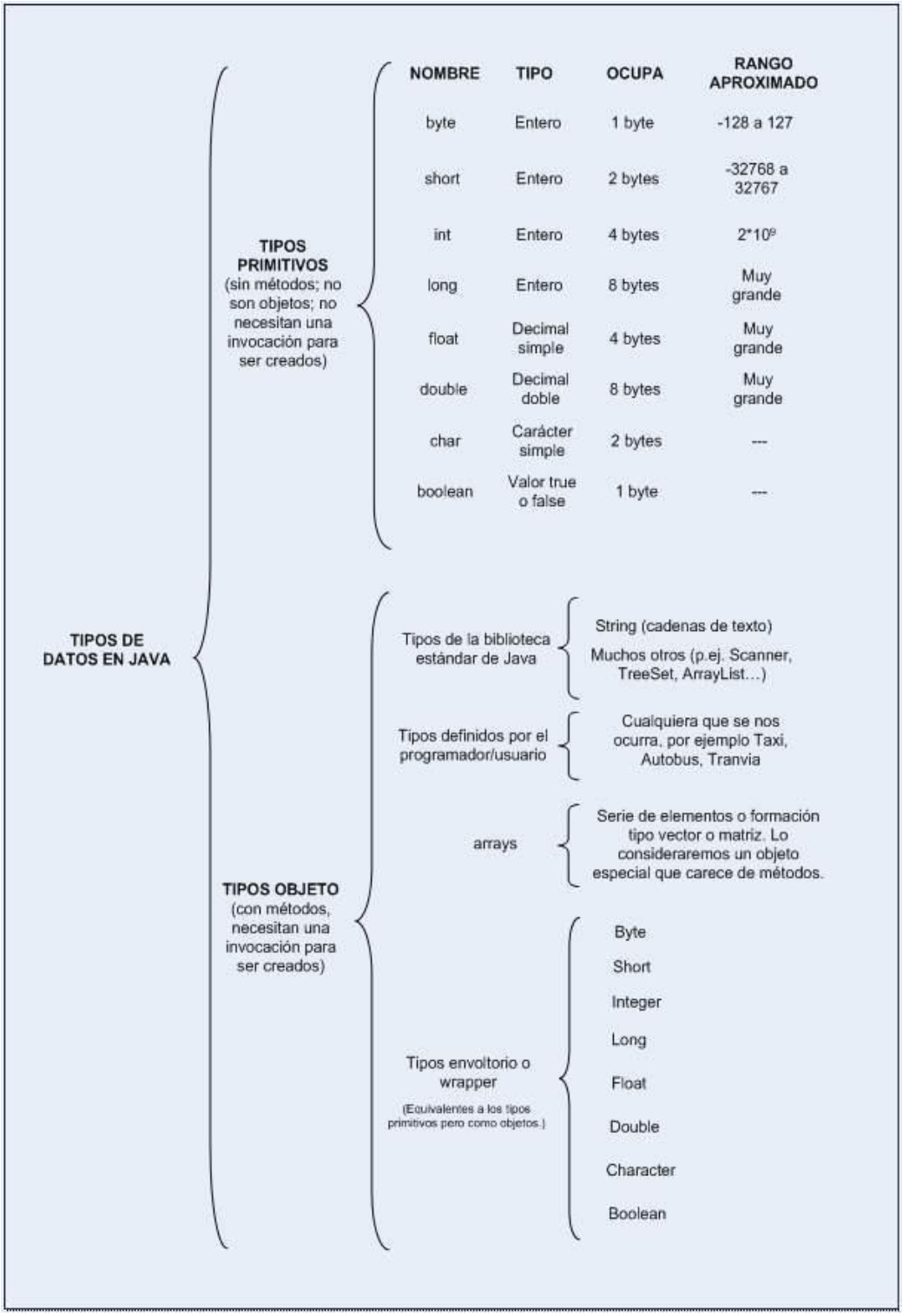
Aunque null, true y false no son palabras clave, se les aplican las mismas restricciones.

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

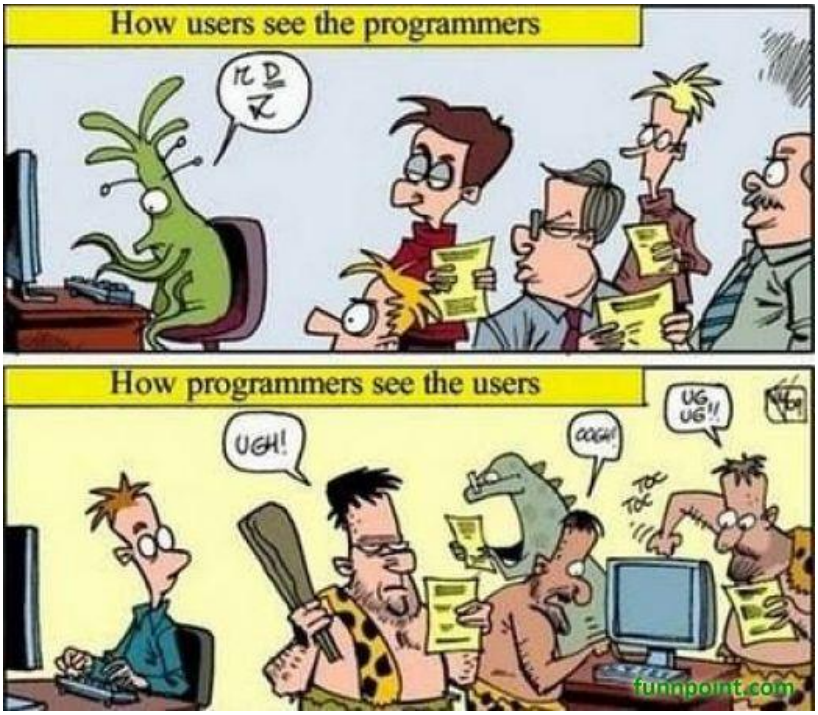
When programmers play Scrabble.

Variables y Tipos de Datos

Las variables son los sustantivos de los lenguajes de programación. En una primera aproximación se puede decir que son almacenes de información. Esta información puede variar a lo largo del tiempo, por lo que se justifica el nombre de variable para estos almacenes. Cada una de las variables tiene un nombre o identificador que la diferencia del resto y sirve para invocar a su contenido, (invariable en el tiempo), y sirve para almacenar un tipo de dato concreto.



Tipos Primitivos



Algunas de las palabras reservadas son nombres de tipos de datos Java:

Tipo	Descripción
boolean	Valor lógico true o false
char	Carácter Unicode 1.1.5, 16 bits.
byte	Entero en complemento a dos con signo, 8 bits.
short	Entero en complemento a dos con signo, 16 bits.
int	Entero en complemento a dos con signo, 32 bits.
long	Entero en complemento a dos con signo, 64 bits.
float	Número en punto flotante IEEE 754-1985, 32 bits.
double	Número en punto flotante IEEE 754-1985, 64 bits.

Literales (constantes)

En Java, cada uno de los tipos de datos tiene *literales*, que es como se escriben los valores constantes de cada tipo. En los apartados siguientes se describe cómo se especifican constantes literales para cada tipo.

Literales Booleanos

Los literales booleanos son true y false.

Literales Enteros

Las constantes enteras son cadenas de dígitos octales, decimales o hexadecimales. El comienzo de la constante declara la base del número:

- un 0 inicial denota un número octal, base 8 (035).
- un 0x o un 0X inicial denota un número hexadecimal, base 16(0x1D, 0X1D)).
- cualquier otra posibilidad denota un número decimal, base 10 (29).

Las constantes enteras con long si terminan en L o en l, (29L). En otro caso se supone que son de tipo int. Si un literal de tipo int se asigna a un short o a un byte y su valor está dentro del rango válido para ese tipo, el literal entero se trata como si fuera un short o un byte.

Literales en Punto Flotante

Los números en punto flotante se expresan como números decimales con punto decimal opcional, seguidos opcionalmente por un exponente. Deben contener como mínimo un dígito: 18., 1.8e1, .18E2

El numero puede ir seguido por f o F para denotar una constante de simple precisión, o por d o D para las constantes de doble precisión: 1.8F , 1.8D. Las constantes de tipo punto flotante son de doble precisión a menos que se diga lo contrario.

El cero puede ser positivo, +0.0, o negativo, -0.0. Ambos son iguales a efectos de comparación, aunque producen resultados distintos cuando se utilizan en expresiones (1.0d/0.0d es positivo, mientras que 1.0d/-0.0d es negativo).

Una constante double no se puede asignar diretamente a una variable float aunque esté en el rango permitido. Se debe usar una constante float o convertir la double a float.

Literales Carácter

"Ye gods, man, you've just launched all the missiles! Quick, press 'control, alt, delete' ..."

Los literales de carácter aparecen entre comillas simples, como 'a'. Ciertos caracteres especiales se pueden representar utilizando secuencias de escape:

- \n salto de línea (\u000A).
- \t tabulador (\u0009).
- \b retroceso (\u0008).
- \r retorno de carro (\u000D).
- \f salto de página (\u000C).
- \\ barra invertida (\u005C).
- \' apóstrofe ' (\u0027).
- \" dobles comillas " (\u0022).
- \ddd un carácter dado en código octal (sólo ASCII).
- \uhhhh un carácter dado en código hexadecimal (Unicode).

Literales Cadenas de Caracteres

Los literales de cadenas de caracteres son secuencias de caracteres entre comillas dobles: "Hola mundo". Todas las secuencias de escape para los literales tipo caracter son válidas para las cadenas de caracteres. Un literal de cadena de caracteres referencia un objeto de tipo String, aunque esto ya lo explicaremos más adelante.

Literales Referencia

La única referencia literal de objeto es null. Se puede usar en cualquier lugar donde se espere una referencia. null representa un objeto no válido o no creado.

Variables

Antes de usar una variable debe ser declarada. La declaración de una variable especifica el tipo de datos, el nombre de la variable y, opcionalmente, el valor por defecto de la variable. Más adelante se explicará cómo limitar el acceso a una variable en su declaración. Una declaración general de variable es del tipo:

tipo_de_dato identificador [= valor_por_defecto] {, identificador [= valor_por_defecto] }*
La parte del tipo de una declaración dice qué valores va a admitir esa variable y el comportamiento de la misma. Los identificadores son tan sólo símbolos, los nombres de las variables que se están definiendo.

No hay diferencia entre declarar las variables en una sola declaración o declararlas en múltiples declaraciones del mismo tipo. Por ejemplo

```
int i=1;
int j=2;
int k;
k=3;
```

es equivalente a

```
int i=1, j=2, k=3;
```

Las declaraciones de variables se pueden poner en cualquier lugar del código, siempre y cuando precedan al primer uso de la variable. Sin embargo, es práctica común situar las declaraciones al principio de cada bloque de código.

Variables de Tipo Booleano

Las variables tipo booleano contienen los valores verdad o falso. Las variables de tipo booleano sin inicializar se inicializan a false.

Variables de Tipo Carácter

El tipo de datos carácter, char, contiene un solo carácter. Cada carácter es un número o código de ccarácter que se refiere al juego de caracteres Unicode. Y esto es así porque los diseñadores de Java tuvieron en cuenta la internacionalización y la escalabilidad. Las variables de tipo carácter se inicializan a '\u0000'.

Variables de Tipo Numérico

Java posee seis tipos de datos numéricos que difieren en el tamaño y en la precisión de los números que puede contener. Generalmente, cuando se escoge un tipo de datos para una variable, se deberá usar el tipo de dato más pequeño que contenga el mayor número con el que se va a trabajar, ahora o en un futuro previsible. Esto ahorra memoria, al tiempo que proporciona espacio para futuras expansiones que necesite el código.

La VM de Java inicializa cada variable numérica a cero antes de su uso (0, 0.0f o 0.0d, dependiendo si son enteras o de punto flotante de simple o doble precisión). Esto constrasta con la mayoría de los lenguajes de programación en que las variables no inicializadas contienen valores aleatorios.

Variables de Tipo String

Un String (cadena) es una secuencia de caracteres. El tipo de datos String es en realidad una clase del núcleo API (java.lang.String), más que un tipo integrado, pero se utiliza tan frecuentemente que es apropiado introducirlo aquí.

Los Strings pueden ser tan largos como se desee; no se ha definido una longitud máxima en la especificación de Java. Sin embargo, la mayoría de las implementaciones la limitarán probablemente en torno a los dos mil millones de caracteres, (2GB), lo que supone capacidad más que suficiente para casi cualquier aplicación.

Los Strings son de sólo lectura, es decir, no se puede cambiar el contenido de un String, aunque se puede redefinir una variable de tipo String (Java proporciona la clase StringBuffer para las cadenas de caracteres modificables).

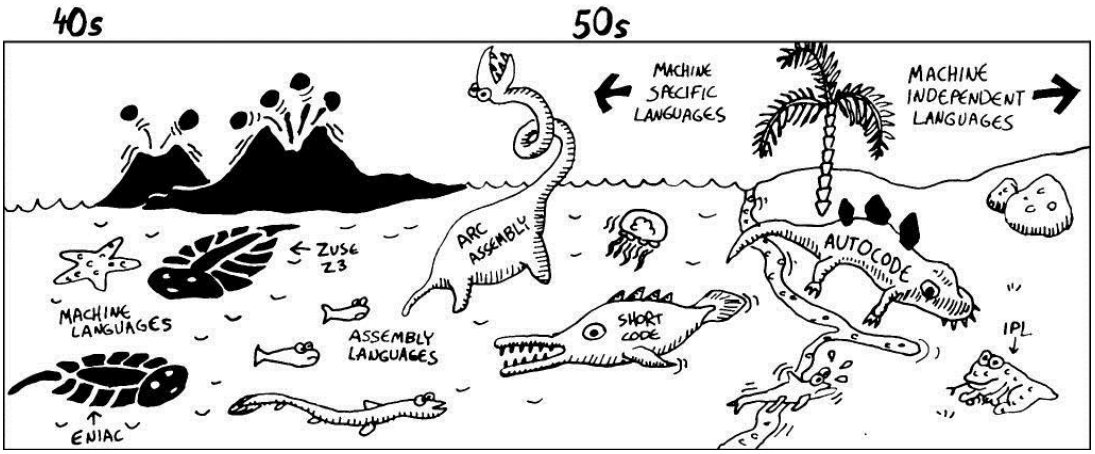
Así, la variable String mensaje definida inicialmente como:

String mensaje = "Hola mundo";

no se puede variar su contenido. Aunque si que se puede conseguir que la variable mensaje apunte a otro objeto String:

mensaje = mensaje + "!!";

Expresiones y Operadores



Las *expresiones* son combinaciones de variables, palabras reservadas, literales, llamadas a métodos y campos, y operadores que son evaluadas y dan como resultado un valor de un determinado tipo de datos. Este valor puede ser un número, un caracter o cualquier otro tipo de datos.

El tipo de una expresión está determinado por los tipos de las partes que la componen y la semántica de los operadores.

Un *operador* es un símbolo que transforma una variable o la combina de alguna otra manera con otra variable o literal.

Por ejemplo, la siguiente expresión tiene dos operadores, '=' o asignación y '*' o producto:

```
a = b * c;
```

Las expresiones con que actúa un operador se llaman *operandos*. Existen operadores unarios y binarios, según actuen sobre uno o dos operandos. Por ejemplo, el operador asignación es binario mientras que el '-' es unario en este caso:

```
a = -d;
```

Precedencia y Asociatividad de los operadores

Prior.	Operador	Tipo de operador	Operación
1	++ -- +, - ~ ¡	Aritmético Aritmético Aritmético Integral Booleano	Incremento previo o posterior (unario) Incremento previo o posterior (unario) Suma unaria, Resta unaria Cambio de bits (unario) Negación (unario)
2	(tipo)	Cualquiera	
3	*, /, %	Aritmético	Multiplicación, división, resto
4	+, - +	Aritmético Cadena	Suma, resta Concatenación de cadenas
5	<< >> >>>	Integral Integral Integral	Desplazamiento de bits a izquierda Desplazamiento de bits a derecha con inclusión de signo Desplazamiento de bits a derecha con inclusión de cero
6	<, <= >, >= instanceof	Aritmético Aritmético Objeto, tipo	Menor que, Menor o igual que Mayor que, Mayor o igual que Comparación de tipos
7	== ¡= == ¡=	Primitivo Primitivo Objeto Objeto	Igual (valores idénticos) Desigual (valores diferentes) Igual (referencia al mismo objeto) Desigual (referencia a distintos objetos)
8	& &	Integral Booleano	Cambio de bits AND Producto booleano
9	^ ^	Integral Booleano	Cambio de bits XOR Suma exclusiva booleana
10	| |	Integral Booleano	Cambio de bits OR Suma booleana
11	&&	Booleano	AND condicional
12	||	Booleano	OR condicional
13	? :	Booleano, cualquiera, cualquiera	Operador condicional (ternario)
14	= *=, /=, %= +=, -= <<=, >>= >>>= &=, ^=, |=	Variable, cualquiera	Asignación Asignación con operación

La *precedencia* de los operadores es su 'adherencia' con respecto a los demás. Los operadores tienen diferentes precedencias, lo que establece el orden de evaluación o de aplicación de cada uno de ellos cuando hay más de uno en la misma expresión. La precedencia se puede modificar utilizando los paréntesis '(', ')'.

Cuando aparecen juntos dos operadores con la misma precedencia, la *asociatividad* determina cuál se evaluará primero.

La siguiente tabla recoge todos los operadores en orden de mayor a menor precedencia. Todos los operadores son binarios excepto los que actuan sobre expr, los operadores de creación y de conversión de tipo, mientras que el operador condicional es ternario. Los operadores con idéntica precedencia aparecen en la misma línea:

postfijos	[] . (parámetros) expr++ expr--
unarios	++expr --expr +expr -expr ~ !
creación o conversión	new (tipo)expr
multiplicación	* / %
suma	+ -
desplazamiento	<< >> >>>
relacionales	< > <= >= instanceof
igualdad	== !=
AND a nivel de bits	&
XOR a nivel de bits	^
OR a nivel de bits	|
AND lógico	&&
OR lógico	||
condicional	?:
Asignación	= += -= *= /= %= >>= <<= >>>= &= ^= |=

Todos los operadores binarios que no son de asignación asocian por la izquierda. Los de asignación asocian por la derecha. Lo que hace que

a=b=c;

sea equivalente a

a=(b=c);

Orden de Evaluación

Java garantiza que los operandos en una expresión se evalúan de izquierda a derecha. Así, por ejemplo, en la expresion x + y + z, el compilador evalúa x, evalúa y, suma los dos valores, evalúa z y lo suma al resultado anterior.

El orden de evaluación tiene importancia cuando x, y o z ocasionan efectos colaterales de cualquier tipo. Por ejemplo, si son evocaciones a métodos que afectan al estado de un objeto, o imprimen algo, el resultado variaría dependiendo del orden de evaluación.

Excepto en el caso de los operadores &&, || y ?:, todos los operandos de un operador son evaluados antes de que se realice la operación.

Tipo de la Expresión

Todas las expresiones tienen un tipo de dato. El tipo de una expresión está determinado por los tipos de las partes que la componen y la semántica de los operadores.

Si se aplica un operador aritmético a un valor entero, el resultado es de tipo int, a menos que uno de los dos operandos sea long, en cuyo caso el resultado será long. Todas las operaciones con enteros se realizan con precisión int o long, por lo que los tipos byte y short son promocionados a int antes de ser evaluados.

Si cualquiera de los operandos es un número en punto flotante, la operación se realiza en aritmética de punto flotante. Tales operaciones se hacen en float a menos que uno de los operandos sea double, en cuyo caso se usa double tanto para el cálculo como para el resultado.

Cuando se utiliza un char en una operación aritmética se convierte a int asignando ceros a los 16 bits superiores.

Conversiones de Tipo
Java es un lenguaje de tipado fuerte, lo que significa que comprueba la compatibilidad del tipo en tiempo de compilación en casi todos los casos. Java impide las asignaciones incompatibles y proporciona conversiones cuando la compatibilidad es posible.

Conversiones Implícitas

Algunas conversiones se realizan de manera automática, sin tener que preocuparnos por ellas. Estas se producen cuando utilizamos un tipo de dato más pequeño en lugar de uno más grande, en términos de almacenamiento.

Así, podríamos utilizar un int en lugar de un long (pero no al contrario). También se produce una conversión implícita de enteros a punto flotante, pero no a la inversa. Además, un valor en punto flotante se puede asignar a cualquier variable en punto flotante de precisión igual o mayor.

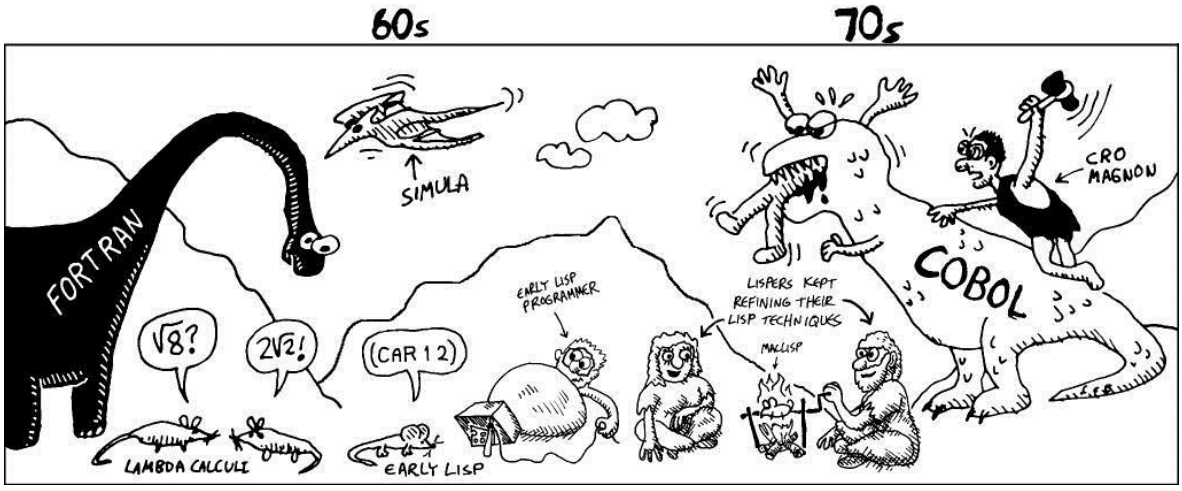
Conversiones Explícitas

Cuando un tipo no se puede asignar a otro con conversión implícita a menudo se puede convertir explícitamente al otro tipo. Una conversión explícita solicita un valor nuevo de un tipo nuevo que sea la mejor representación disponible del valor antiguo en el tipo antiguo. Algunas conversiones no están permitidas, como un boolean a int. Este tipo de conversiones puede llevar asociado la pérdida de precisión y el redondeo.

Las conversiones explícitas se pueden realizar colocando delante del elemento cuyo tipo se quiere cambiar el nuevo tipo entre paréntesis, como se puede apreciar en el siguiente ejemplo.

```
double d=7.99;
int i=(int)d;
```

Conversiones de cadenas



La clase String es especial, pues se usa implícitamente en el operador + de concatenación de cadenas. Cuando en la concatenación está un objeto no String Java lo convierte a cadena de caracteres invocando su método toString.

Cuando se convierte a cadena de caracteres una referencia nula resulta la cadena "null". Si no hay definido un método toString para una clase, se utiliza uno por defecto, que devuelve una representación del objeto en forma de cadena de caracteres.

Operadores Aritméticos

Operan sobre cualquier tipo de dato numérico, ya sea entero o punto flotante, y producen resultados enteros o de punto flotante. Son:

- - menos unario,
- + más unario,
- * producto,
- / división,
- % resto de la división,
- + suma y
- - resta.

La aritmética de enteros es aritmética modular en complemento a dos, es decir, si un valor excede el rango de su tipo es recucido a su módulo. Por tanto, la aritmética de enteros nunca produce desbordamientos ni subdesbordamientos. La división entre enteros trunca el resto con la siguiente regla: (x/y)*y + x%y == x. La división por cero o el resto por cero no son válidas.

La aritmética de caracteres es aritmética de enteros ya que un char es convertido implícitamente a int.

Java usa el estándar IEEE 754-1985 tanto para la representación como para la aritmética de números en punto flotante. De acuerdo con él, podemos producir desbordamientos hacia el infinito o subdesbordamientos a cero. También se puede representar los resultados de expresiones no válidas, NaN, como la división por cero.

Concatenación de Cadenas de Caracteres

Se puede usar + para concatenar dos cadenas de caracteres:

String ho="ho";

String la= ho + "la";

la += "!";

System.out.println(la);

cuya salida será:

hola!

Operadores de Incremento y Decremento

Los operadores ++ y -- son los operadores de incremento y decremento, respectivamente. La expresión i++ es equivalente a i=i+1, salvo que i se evalúa sólo una vez.

Los operadores de incremento y decremento pueden ser prefijos o postfijos, dependiendo de si aparecen delante o detrás del operando. Si el operador está delante, la operación se realiza antes de que sea devuelto el valor de la expresión. Si está detrás la operación se realiza después de que se haya usado el valor original. Por ejemplo:

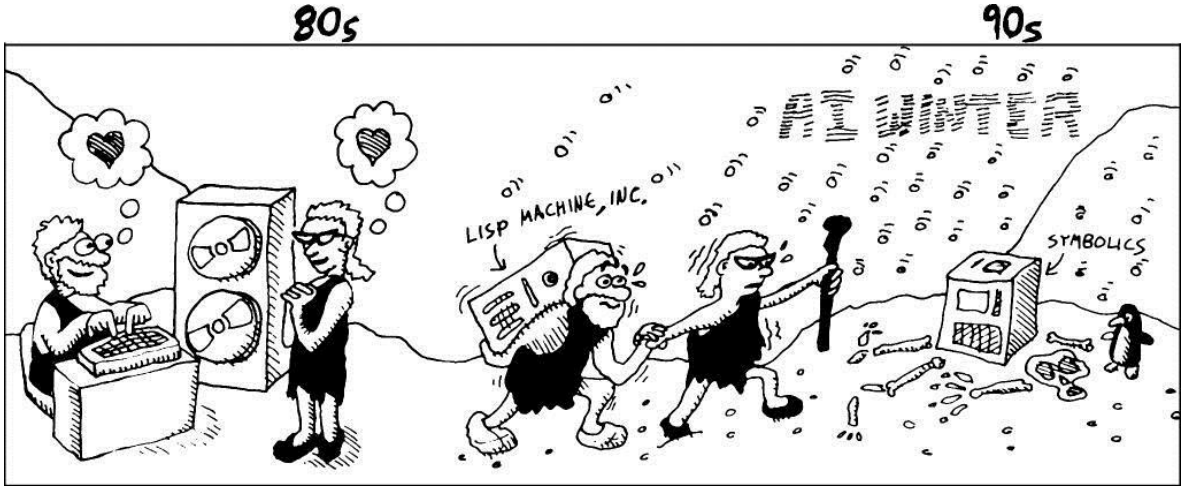
```
class Prepost{
  public static void main(String [] args){
  int i=0;
  System.out.println(++i + " " + i++ + " " + i);
  }
}
```

produce la salida

1 1 2

Los operadores de incremento y decremento se pueden aplicar también a variables char para obtener el carácter Unicode siguiente o anterior, respectivamente.

Operadores Lógicos, Relacionales y Condicionales



Java soporta un juego estándar de operadores relacionales y de igualdad:

- > mayor que
- >= mayor o igual que
- < menor que
- <= menor o igual que
- == igual
- != distinto
-

Todos estos operadores dan lugar a valores boolean. El operador unario ! convierte un valor lógico en el complementario, !true es igual a false. Los operadores == y != son los únicos que pueden comparar operandos booleanos. Los demás tipos se comparan de forma natural.

Dos referencias se pueden comparar siempre a efectos de igualdad. ref1 == ref2 será true cuando se refieran al mismo objeto o cuando ambas sean null. En caso contrario serán false.

Para comparar la igualdad de dos objetos de tipo String hay que utilizar el método String.equals ya que la comparación con el operador == compararía la referencia, no el contenido de los objetos.

Los resultados de expresiones booleanas se pueden combinar con otros resultados para componer expresiones más complejas utilizando las siguientes conectivas:

- && Y lógico
- || O lógico

Estos operadores intentan evitar evaluar el segundo operando si es posible, de modo que en la expresión x && y la variable y será evaluada sólo si x es true, ya que en otro caso la expresión será siempre false.

Java ha mantenido el operador condicional de asignación del C: ?:. Este operador es ternario:

variable = (condición ? valor1 : valor2);

El primer operando es de tipo boolean, y los otros dos pueden ser de cualquier tipo compatible con la variable a la que se asignará uno de ellos dependiendo del valor

de la condición. Si la condición es true el valor de la expresión será valor1, si el false será valor2.

```
int a=1,b=2,menor;
menor=(a<b?a:b);  // menor contendrá el menor de a y b
```

Operadores de Campos de Bits
Los operadores binarios a nivel de bits se pueden usar sólo con datos enteros o booleanos:

- & Y
- | O inclusivo
- ^ O exclusiva (XOR)
- ~ Complemento
- << Desplaza los bits a la izquierda, rellenando por la derecha con ceros.
- >> Desplaza los bits a la derecha, rellenando por la izquierda con el bit más alto (signo).
- >>> Desplaza los bits a la derecha, rellenando por la izquierda con ceros.

Ejemplos de sentencias que utilizan estos operandos son:

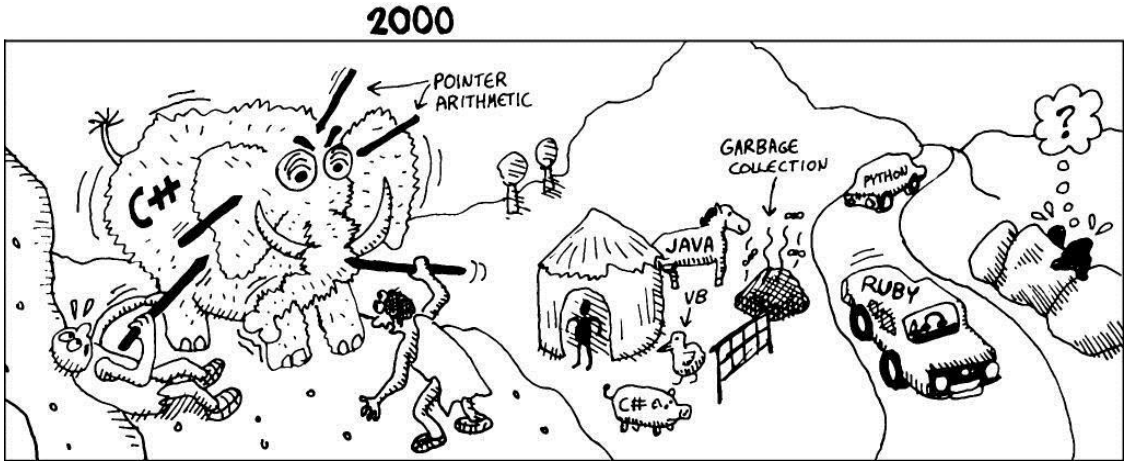
```
int a=1;
int b;

b=a<<1;
System.out.println("a << 1 = " + b);
b=a<<2;
System.out.println("a << 2 = " + b);
b=a & 0x00000000;
System.out.println("a & 0x00000000 = " + b);
b=a | 0x00000000;
System.out.println("a | 0x00000000 = " + b);
```

cuyos resultados serán;

```
a << 1 = 2
a << 2 = 4
a & 0x00000000 = 0
a | 0x00000000 = 1
```

Los operadores booleanos a nivel de bits, & y | se pueden usar también sobre valores boolean, devolviendo el mismo valor que sus homólogos lógicos && y ||, aunque con una diferencia: los operadores a nivel de bit evalúan siempre los dos operandos.

Operadores de Asignación



El operador asignación más elemental es =, y su cometido es asignar el operando derecho al izquierdo, y devolver el derecho como resultado. Cualquier otro operando aritmético o de nivel de bits binario puede combinarse con = para formar otro operador de asignación.

Así, la expresión b += a es equivalente b = b + a, sólo que en la primera versión b se evalúa sólo una vez.

Fuente:

La Biblia de Java. Vanhelsuwé et. al. Anaya Multimedia. 1996.

El Lenguaje de Programación Java. Ken Arnold y James Gosling. Addison-Wesley/DOMO. 1997.

http://www.infor.uva.es/~jvegas/cursos/prog/tema3.html