Caracteres y secuencias de escape de Java

En este tema vamos a estudiar los caracteres que pueden aparecer en un programa Java. Ya sabemos como crear un programa básico, ahora veremos los caracteres que podemos usar para construir programas más complejos.
Los caracteres que pueden aparecer en un programa Java para formar las constantes, variables, expresiones, etc., son:

Las **letras mayúsculas y minúsculas** de la A(a) a la Z(z) de los alfabetos internacionales (la ñ y Ñ son válidas)
Dígitos (0, 1, 2, ...)
Los caracteres ' _ ' '$' y cualquier carácter unicode por encima del 00C0.
Los caracteres especiales y signos de puntuación siguientes:
+ - * / = % & # ! ? ^ " ` ~ \ | < > ( ) [ ] { } : ; . ,
Espacios en blanco: se llaman espacios en blanco a los siguientes caracteres: espacio, tabulador, salto de línea. Su labor es la misma que la de un espacio en blanco: separar entre elementos de un programa.

Por ejemplo, podemos escribir el método main sin utilizar espacios en blanco, de la forma:

```
public static void main(String [] args){System.out.println("Hola Mundo!!!");}
```

Aunque queda mucho más claro si introducimos espacios:

```
public static void main(String [] args){
  System.out.println("Hola Mundo!!!");
}
```

Secuencias de escape: Una secuencia de escape esta formada por una barra inversa seguida de una letra o de una combinación de dígitos.
Una secuencia de escape siempre representa un solo carácter aunque se escriba con dos o más caracteres.
Se utilizan para realizar acciones como salto de línea o para usar caracteres no

imprimibles.

Algunas secuencias de escape definidas en Java son:

Secuencia de escape	Descripción
\n	Ir al principio de la línea siguiente
\b	Retroceso
\t	Tabulador horizontal
\r	Ir al principio de la línea
\"	Comillas
\'	Comilla simple
\\	Barra inversa
\udddd	Carácter Unicode

Java: Identificadores y palabras reservadas

1. IDENTIFICADORES JAVA

Los identificadores son los nombres que el programador asigna a variables, constantes, clases, métodos, paquetes, etc. de un programa.

Características de un identificador Java:

Están formados por letras y dígitos.

No pueden empezar por un dígito.

No pueden contener ninguno de los caracteres especiales vistos en una entrada anterior.

No puede ser una palabra reservada de Java. Las palabras reservadas en Java son todas las que aparecen en el punto siguiente.

Ejemplo de identificadores válidos:
Edad nombre _Precio
$cantidad PrecioVentaPublico

Ejemplo de identificadores no válidos:
4num z# "Edad"

Java diferencia mayúsculas y minúsculas, por lo tanto, nombre y Nombre son identificadores diferentes.

2. PALABRAS RESERVADAS DE JAVA

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

Las palabras reservadas en Java ordenadas alfabéticamente son las siguientes:

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**

Comentarios en Java

Un comentario es un texto que se escribe dentro de un programa con el fin de facilitar la comprensión del mismo.
Los comentarios se utilizan para explicar y documentar el código fuente.
En Java se pueden utilizar tres tipos de comentarios:

Comentario tradicional estilo C/C++.

Empieza con los caracteres /* y acaba con */.
Pueden ocupar más de una línea y pueden aparecer en cualquier lugar donde pueda aparear un espacio en blanco.
No pueden anidarse.

Ejemplos de comentarios estilo C/C++:

```
/* Programa Ecuación segundo grado
   Calcula las soluciones de una ecuación de segundo grado */

/* Lectura de datos por teclado */
```

Comentarios de una sola línea.

Comienzan con una doble barra (//) y se pueden extender hasta el final de la línea.
No tienen carácter de terminación.

Ejemplos de comentarios de una sola línea:

```
// Programa Ecuación segundo grado

// Calcula las soluciones de una ecuación de segundo grado

int p; // precio del producto
```

Comentarios de documentación Javadoc.

Son comentarios especiales para generar documentación del programa.

Comienza con /** y termina con */

Ejemplo de comentario de documentación Javadoc:

```
/**
 *
 * @author Fran
 * @version 1.0
 *
 */
```

Tipos de datos Java

REPRESENTACIÓN INTERNA DE LOS DATOS

En el mundo real los datos que manejamos se representan mediante letras, números, símbolos, imágenes, sonidos, etc.

Esto se conoce como **representación externa** de los datos.

Pero si queremos introducirlos en un ordenador, todos estos elementos se deben transformar ó codificar.

Un ordenador está compuesto fundamentalmente por circuitos electrónicos digitales. Los datos circulan por estos circuitos en forma de impulsos eléctricos.

De forma muy simplificada podemos decir que por un circuito pasa o no pasa corriente y esto lo podemos representar con dos dígitos: 0 y 1.

Todos los datos e información que contiene un ordenador, están representados de forma interna mediante secuencias de ceros y unos.

Un sistema de representación que utiliza solamente dos símbolos (0 , 1) se llama **sistema binario.**

Por tanto, los datos tal y como los expresamos de forma natural se deben codificar de forma interna en binario para que puedan ser tratados por el ordenador.

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 **2^N 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 2^8 = 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

Un dato siempre lleva asociado un tipo de dato, que determina el conjunto de valores que puede tomar.

En Java toda la información que maneja un programa está representada por dos tipos principales de datos:

a) Datos de tipo básico o primitivo.

b) Referencias a objetos.

Los tipos de datos básicos o primitivos no son objetos y se pueden utilizar directamente en un programa sin necesidad de crear objetos de este tipo. La biblioteca Java proporciona clases asociadas a estos tipos que proporcionan métodos que facilitan su manejo.

Los tipos de datos primitivos que soporta Java son:

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	$\pm$ 3.4x10^{-38} a $\pm$ 3.4x10^{38}	0.0	Float

double	Numérico en Coma flotante de precisión doble Norma IEEE 754	8	$\pm$ 1.8x10^{-308} a $\pm$ 1.8x10^{308}	0.0	Double
char	Carácter Unicode	2	\u0000 a \uFFFF	\u0000	Character
boolean	Dato lógico	-	true ó false	false	Boolean
void	-	-	-	-	Void

DATOS NUMÉRICOS ENTEROS

En Java los representan los tipos: byte, short, int, long.

El tipo de dato numérico entero es un subconjunto finito de los números enteros del mundo real. Pueden ser positivos o negativos.

Ejemplo de declaración de variables enteras:

```
int a;
byte n1, n2;
short x;
```

DATOS NUMÉRICOS REALES

En Java los representan los tipos: float, double.

El tipo de dato numérico real es un subconjunto finito de los números reales. Siempre llevan un punto decimal y también pueden ser positivos o negativos. Los números reales tienen una parte entera y una parte decimal.

Por ejemplo: 0.08 -54.0001

Ejemplo de declaración de variables reales:

```
float peso;
double longitud;
float altura = 2.5F;
double area = 1.7E4; // equivale a 1.7 * 10^4
double z = .123; //si la parte entera es 0 se puede omitir
```

DATOS DE TIPO CARÁCTER

En Java se representa con el tipo char.

Un dato de tipo carácter se utiliza para representar un carácter dentro del rango \u0000 a \uFFFF (números desde 0 hasta 65535) en **Unicode**.

En realidad un dato de tipo char **contiene un número entero** dentro del rango anterior que representa un carácter.

En Java se utiliza el código Unicode para la representación de caracteres. Este código actualmente representa los caracteres de la mayoría de idiomas escritos en todo el mundo.

Los 127 primeros caracteres de Unicode corresponden al código ASCII.

El **Código ASCII** (*American Standard Code for Information Interchange* o Código Estándar Americano para el Intercambio de Información) asigna valores numéricos a las letras, números, signos de puntuación y algunos otros caracteres especiales.

ASCII incluye **256 códigos** divididos en dos conjuntos, estándar y extendido, de 128 cada uno. El conjunto **ASCII básico**, o estándar, utiliza **7 bits** para cada código, lo que da como resultado 128 códigos de caracteres desde **0 hasta 127**.

El conjunto **ASCII extendido** utiliza **8 bits** para cada código, dando como resultado 128 códigos adicionales, numerados desde el **128 hasta el 255**.

En el conjunto de caracteres ASCII básico, los primeros 32 valores están asignados a los códigos de control de comunicaciones y de impresora (caracteres no imprimibles) empleados para controlar la forma en que la información es transferida desde una computadora a otra o desde una computadora a una impresora. En este grupo están los códigos correspondientes a la barra espaciadora (SP por space), la tecla ENTER de retorno de carro a un nuevo renglón (CR por carry return), etc. También existen caracteres de control usados en teleprocesamiento, como ser ACK (Acknowledge - aviso de mensaje recibido), BEL (bell - aviso por señal sonora), ETX (end of text – fin de texto), STX (start of text – comienzo de texto), etc.

Los 96 códigos restantes del código básico corresponden a los caracteres imprimibles y se asignan a los signos de puntuación corrientes, a los dígitos del 0 al 9 y a las letras mayúsculas y minúsculas del alfabeto latino.

Los códigos correspondientes al ASCII extendido, del 128 al 255, se asignan a aquellos caracteres que no pertenecen al alfabeto anglosajón, por ejemplo, las vocales con tilde, la ñ, y en general todos los caracteres especiales que utilizan los distintos lenguajes.

Debido a lo limitado de su tamaño, el código ASCII no es suficiente para representar caracteres de alfabetos como el Japonés, Chino o árabe. La solución a este problema ha sido crear un código más grande con el que poder representar cualquier carácter de cualquier idioma: el código Unicode.

El código **UNICODE** proporciona una única representación numérica para cada símbolo, independientemente del ordenador, el programa o el lenguaje de programación que se use.

La codificación Unicode se ha transformado en un estándar adoptado por las principales empresas de hardware y software. **Java utiliza la codificación Unicode**.

La descripción completa del estándar y las tablas de caracteres están disponibles en la página web oficial de Unicode http://www.unicode.org/. La referencia completa se publica, además, en forma de libro impreso cada vez que se libera una nueva versión principal. La versión digital de este libro está disponible de forma gratuita.

Ejemplo de declaración de variables de tipo carácter:
char car;
char letra1 = 'z';
char letra = '\u0061'; //código unicode del carácter 'a'

DATOS DE TIPO LÓGICO

Se representan con el tipo boolean.

Los datos de este tipo sólo pueden contener dos valores: true (verdadero) ó false (falso).

Ejemplo de declaración de variables lógicas:

boolean primero;

boolean par = false;

Los tipos de datos lógicos son también conocidos como **booleanos** en honor del matemático inglés George Bool, que desarrolló la teoría conocida como álgebra de bool que fue la base para la representación de los circuitos lógicos.

Literales en Java. Variables y constantes en Java

1. LITERAL JAVA

Un literal Java es un valor de tipo entero, real, lógico, carácter, cadena de caracteres o un valor nulo (null) que puede aparecer dentro de un programa.

Por ejemplo: 150, 12.4, "ANA", null, 't'.

LITERAL JAVA DE TIPO ENTERO

Puede expresarse en decimal (base 10), octal (base 8) y hexadecimal (base 16).

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

El tipo de un literal entero es **int** a no ser que su valor absoluto sea mayor que el de un int ó se especifique el sufijo l ó L en cuyo caso será de tipo **long**.

Literal Java de tipo entero en Base decimal

Está formado por 1 o más dígitos del 0 al 9.

El primer dígito debe ser distinto de cero.

Por ejemplo:

1234 literal java entero de tipo int

1234L literal java entero de tipo long

123400000000 literal java entero de tipo long

Literal Java de tipo entero en Base octal

Está formado por 1 o más dígitos del 0 al 7.

El primer dígito debe ser cero.

Por ejemplo:

01234

025

Literal Java de tipo entero en Base hexadecimal

Está formado por 1 o más dígitos del 0 al 9 y letras de la A a la F (mayúsculas o minúsculas).

Debe comenzar por 0x ó 0X.

Por ejemplo:

0x1A2

0x430

0xf4

LITERAL JAVA DE TIPO REAL

Son números en base 10, que deben llevar un parte entera, un punto decimal y una parte fraccionaria. Si la parte entera es cero, puede omitirse.

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

Por ejemplo:

12.2303

-3.44

+10.33

0.456

.96

También pueden representarse utilizando la notación científica. En este caso se utiliza una E (mayúscula o minúscula) seguida del exponente (positivo o negativo). El exponente está formado por dígitos del 0 al 9.

Por ejemplo, el número en notación científica $14*10^{-3}$ se escribe: 14E-3

Más ejemplos de literal Java de tipo real:

2.15E2 -> $2.15 * 10^{2}$

.0007E4 -> $0.0007 * 10^{4}$

-50.445e-10 -> $-50.445 * 10^{-10}$

El tipo de estos literales es siempre **double**, a no ser que se añada el sufijo F ó f para indicar que es float.

Por ejemplo:

2.15 literal real de tipo double

2.15F literal real de tipo float

También se pueden utilizar los sufijos d ó D para indicar que el literal es de tipo double:

12.002d literal real de tipo double

LITERAL JAVA DE TIPO CARÁCTER

Contiene un solo carácter encerrado entre comillas simples.

Es de tipo **char**.

Las secuencias de escape se consideran literales de tipo carácter.

Por ejemplo:

'a'

'4'

'\n'

'\u0061'

LITERAL JAVA DE CADENAS DE CARACTERES

Está formado por 0 ó más caracteres encerrados entre comillas dobles.

Pueden incluir secuencias de escape.

Por ejemplo:

“Esto es una cadena de caracteres”

“Pulsa \”C\” para continuar”

“” -> cadena vacía

“T” -> cadena de un solo carácter, es diferente a ‘t’ que es un carácter

Las cadenas de caracteres en Java son objetos de tipo **String**.

2. VARIABLES

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

Para facilitar la referencia a las posiciones de memoria, se puede sustituir la cadena binaria de ceros y unos que indica la dirección por un identificador.

Un identificador es el nombre que se le da a un componente dentro de un programa (una variable, una constante, un método, una clase, etc.).

Ejemplo de identificadores: nombre, matriculaAlumno, $suma5, año, etc.

Una variable es una posición de memoria que se referencia con un identificador, conocido como nombre de la variable, donde se almacena el valor de un dato que puede cambiar durante la ejecución del programa.

Una variable tiene tres características básicas:

· **Nombre** o identificador de la variable.

· **Tipo**. Conjunto de valores que puede tomar la variable (numérico, carácter, etc.).

· **Valor**. Información que almacena.

Para poder utilizar una variable en un programa, primero tenemos que declararla.

Declarar una variable significa asignarle un nombre y un tipo.

Por ejemplo:

int a; //declaramos la variable a de tipo int

char b; //declaramos la variable b de tipo char

A las variables declaradas dentro de un método no se les asigna un valor automáticamente. Es nuestra responsabilidad asignarles valores iniciales.

int contador = 0; // declara la variable contador de tipo int y se le asigna el valor 0

3. CONSTANTES

Los programas de ordenador contienen ciertos valores que no deben cambiar durante su ejecución. Estos valores se llaman constantes.

Podemos decir que una constante es una posición de memoria que se referencia con un identificador, conocido como nombre de la constante, donde se almacena el valor de un dato que no puede cambiar durante la ejecución del programa.

Una constante en Java se declara de forma similar a una variable, anteponiendo la palabra **final**

Ejemplo:

```
final double PI = 3.141591;
final int diasSemana = 7;
```

Operadores Java

OPERADORES JAVA ARITMÉTICOS

Los operadores aritméticos en java son:

+ Suma. Los operandos pueden ser enteros o reales

- Resta. Los operandos pueden ser enteros o reales

* Multiplicación. Los operandos pueden ser enteros o reales

/ División. Los operandos pueden ser enteros o reales. Si ambos son enteros el resultado es entero. En cualquier otro caso el resultado es real.

% Resto de la división. Los operandos pueden ser de tipo entero o real.

Ejemplo de operaciones aritméticas:

int a = 10, b = 3;
double v1 = 12.5, v2 = 2.0;
char c1='P', c2='T';

Operación	Valor	Operación	Valor	Operación	Valor
a+b	13	v1+v2	14.5	c1	80
a-b	7	v1-v2	10.5	c1 + c2	164
a*b	30	v1*v2	25.0	c1 + c2 + 5	169
a/b	3	v1/v2	6.25	c1 + c2 + ‘5’	217
a%b	1	v1%v2	0.5		

En aquellas operaciones en las que aparecen operandos de distinto tipo, java convierte los valores al tipo de dato de mayor precisión de todos los datos que intervienen. Esta conversión es de forma temporal, solamente para realizar la operación. Los tipos de datos originales permanecen igual después de la operación.
Los tipos short y byte se convierten automáticamente a int.
Por ejemplo:
int i = 7;
double f = 5.5;
char c = 'w';

Operación	Valor	Tipo
i + f	12.5	double
i + c	126	int
i + c – ‘0’	78	int
(i + c) – (2 * f / 5)	123.8	double

OPERADORES JAVA RELACIONALES

Los operadores relacionales comparan dos operandos y dan como resultado de la comparación verdadero ó falso.

Los operadores relacionales en java son:

< Menor que

> Mayor que

<= Menor o igual

>= Mayor o igual

!= Distinto

== Igual

Los **operandos** tienen que ser de **tipo primitivo**.

Por ejemplo:

int a = 7, b = 9, c = 7;

Operación	Resultado
a==b	false
a >=c	true
b < c	false
a != c	false

OPERADORES JAVA LÓGICOS

Los operadores lógicos se utilizan con operandos de tipo boolean. Se utilizan para construir expresiones lógicas, cuyo resultado es de tipo true o false.

Los operadores lógicos en Java son:

&& AND. El resultado es verdadero si los dos operandos son verdaderos. El resultado es falso en caso contrario. Si el primer operando es falso no se evalúa el segundo, ya que el resultado será falso.

|| OR. El resultado es falso si los dos operandos son falsos. Si uno es verdadero el resultado es verdadero. Si el primer operando es verdadero no se evalúa el segundo.

! NOT. Se aplica sobre un solo operando. Cambia el valor del operando de verdadero a falso y viceversa.

Las definiciones de las operaciones OR, AND y NOT se recogen en unas tablas conocidas como **tablas de verdad.**

A	B	A OR B
F	F	F
F	V	V
V	F	V
V	V	V

A	B	A AND B
F	F	F
F	V	F
V	F	F
V	V	V

A	NOT A
F	V
V	F

F: Falso

V: Verdadero

Como ejemplo, en la siguiente tabla vemos una serie de expresiones lógicas y su valor:

int i = 7;

float f = 5.5F;

char c = ‘w’;

Expresión	Resultado
(i >= 6) && (c == ‘w’)	true
(i >= 6) || (c == 119)	true
(f < 11) && (i > 100)	false
(c != ‘p’) || ((i + f) <= 10)	true
i + f <= 10	false
i >= 6 && c == ‘w’	true
c != ‘p’ || i + f <= 10	true

Las expresiones lógicas en java se evalúan sólo hasta que se ha establecido el valor cierto o falso del conjunto. Cuando, por ejemplo, una expresión va a ser seguro falsa por el valor que ha tomado uno de sus operandos, no se evalúa el resto de expresión.

OPERADORES JAVA UNITARIOS.

Los operadores unitarios en java son:

– + signos negativo y positivo

++ -- incremento y decremento

~ complemento a 1

! NOT. Negación

Estos operadores **afectan a un solo operando**.

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

Ejemplo de operador incremento:

```
int i = 1;
i++; // Esta instrucción incrementa en 1 la variable i.
// Es lo mismo que hacer i = i + 1; i toma el valor 2
```

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

Ejemplo de operador decremento:

```
int i = 1;
i--; // decrementa en 1 la variable i.
// Es lo mismo que hacer i = i - 1; i toma el valor 0
```

Los operadores incremento y decremento pueden utilizarse como prefijo o sufijo, es decir, pueden aparecer antes o después de la variable.

Por ejemplo:

```
i = 5;
i++; // i vale ahora 6
++i; // i vale ahora 7
```

Cuando estos operadores intervienen en una expresión, si preceden al operando (++i), el valor se modificará antes de que se evalúe la expresión a la que pertenece.

En cambio, si el operador sigue al operando (i++), entonces el valor del operando se modificará después de evaluar la expresión a la que pertenece.

Por ejemplo:

```
int x, i = 3;
x = i++;
```

En esta asignación a x se le asigna el valor 3 y a continuación i se incrementa, por lo tanto, después de ejecutarla:

x contiene 3, i contiene 4.

Si las instrucciones son:

```
int x, i = 3;
x = ++i;
```

En esta instrucción primero se incrementa i y el resultado se asigna a x. Por lo tanto, después de ejecutarla:

x contiene 4, i contiene 4.

Otro ejemplo:

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

Estas instrucciones mostrarían por pantalla:

```
1
2
2
```

En cambio, si se cambia la posición del operador:

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

Estas instrucciones mostrarían por pantalla:

```
1
1
2
```

El operador complemento a 1 ~ cambia de valor todos los bits del operando (cambia unos por ceros y ceros por unos). Solo puede usarse con datos de tipo entero. El carácter ~ es el ASCII 126.

Por ejemplo:

```
int a = 1, b = 0, c = 0;
c = ~a;
```

OPERADORES JAVA A NIVEL DE BITS

Realizan la manipulación de los bits de los datos con los que operan.

Los datos deben ser de tipo entero.

Los operadores a nivel de bits en java son:

& and a nivel de bits
| or a nivel de bits
^ xor a nivel de bits
<< desplazamiento a la izquierda, rellenando con ceros a la derecha
>> desplazamiento a la derecha, rellenando con el bit de signo por la izquierda
>>> desplazamiento a la derecha rellenando con ceros por la izquierda

OPERADORES JAVA DE ASIGNACIÓN.

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

Los operadores de asignación en java son:

= 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

Si los dos operandos de una expresión de asignación (el de la izquierda y el de la derecha) son de distinto tipo de datos, **el valor de la expresión de la derecha se convertirá al tipo del operando de la izquierda**.

Por ejemplo, una expresión de tipo real (float, double) se truncará si se asigna a un entero, o una expresión en de tipo double se redondeará si se asigna a una variable de tipo float.

En Java están permitidas las asignaciones múltiples.

Ejemplo: x = y = z = 3; equivale a x = 3; y = 3; z = 3;

Ejemplo de asignaciones en Java:

a += 3; equivale a a = a + 3;

a *= 3; equivale a a = a * 3;

En general:

variable **op=** expresión equivale a: variable = variable **op** expresión

En la siguiente tabla vemos más ejemplos de asignaciones:

int i = 5, j = 7, x = 2, y = 2, z = 2;

float f = 5.5F, g = -3.25F;

Expresión	Expresión equivalente	Valor final
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
x *= -2 * (y + z) / 3	x = x * (-2 * (y + z) / 3)	-4

OPERADOR JAVA CONDICIONAL

Se puede utilizar en sustitución de la sentencia de control if-else, pero hace las instrucciones menos claras.

El operador condicional java se forman con los caracteres **?** y **:**

Se utiliza de la forma siguiente:

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

Si expresión1 es cierta entonces se evalúa expresión2 y éste será el valor de la expresión condicional. Si expresión1 es falsa, se evalúa expresión3 y éste será el valor de la expresión condicional.

Ejemplo de operador condicional:

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

Esta expresión asigna a j el valor 100. Su significado es: si el valor de i es menor que 0 asigna a j el valor 0, sino asigna a j el valor 100. Como i vale 10, a j se le asigna 100.

La instrucción anterior es equivalente a escribir:

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

Más ejemplos de operador condicional:

```
int a=1, b=2, c=3;
c+=(a>0 && a<= 10) ? ++a : a/b;   // c toma el valor 5
int a=50, b=10, c=20;
c+=(a>0 && a<=10) ? ++a : a/b;   // c toma el valor 25
```

PRIORIDAD Y ORDEN DE EVALUACIÓN DE LOS OPERADORES EN JAVA

La siguiente tabla muestra todos los operadores Java ordenados de mayor a menor prioridad. La primera línea de la tabla contiene los operadores de mayor prioridad y la última los de menor prioridad. Los operadores que aparecen en la misma línea tienen la misma prioridad.

Una expresión entre paréntesis siempre se evalúa primero y si están anidados se evalúan de más internos a más externos.

Operador	Asociatividad
() [] .	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

Estructuras de control en Java

Las estructuras de control determinan la secuencia de ejecución de las sentencias de un programa.

Las estructuras de control se dividen en tres categorías:

Secuencial

Condicional o Selectiva

Iterativa o Repetitiva.

1. ESTRUCTURA SECUENCIAL

El orden en que se ejecutan por defecto las sentencias de un programa es secuencial. Esto significa que las sentencias se ejecutan en secuencia, una después de otra, en el orden en que aparecen escritas dentro del programa.

La estructura secuencial está formada por una sucesión de instrucciones que se ejecutan en orden una a continuación de la otra.
Cada una de las instrucciones están separadas por el carácter punto y coma (;).
Las instrucciones se suelen agrupar en bloques.
El bloque de sentencias se define por el carácter llave de apertura ({) para marcar el inicio del mismo, y el carácter llave de cierre (}) para marcar el final.
Ejemplo:

```
{
    instrucción 1;
    instrucción 2;
    instrucción 3;
}
```

En Java si el bloque de sentencias está constituido por una única sentencia no es obligatorio el uso de las llaves de apertura y cierre ({ }), aunque sí recomendable.

Ejemplo de programa Java con estructura secuencial: Programa que lee dos números por teclado y los muestra por pantalla.

```
/* Programa que lea dos números por teclado y los muestre por pantalla.
 */
import java.util.*;
public class Main {
  public static void main(String[] args){
    //declaración de variables
    int n1, n2;
    Scanner sc = new Scanner(System.in);
    //leer el primer número
    System.out.println("Introduce un número entero: ");
```

```
    n1 = sc.nextInt();      //lee un entero por teclado
    //leer el segundo número
    System.out.println("Introduce otro número entero: ");
    n2 = sc.nextInt();      //lee un entero por teclado

    //mostrar resultado
    System.out.println("Ha introducido los números: " + n1 + " y " + n2);
  }
}
```

Ejemplo de programa Java con estructura secuencial: Programa que lee dos números de tipo double por teclado y calcula y muestra por pantalla su suma, resta y multiplicación.

```
/*
 * Programa que lee dos números de tipo double por teclado
 * y muestra su suma, resta y multiplicación.
 */
import java.util.*;

public class Main {

  public static void main(String[] args){

    Scanner sc = new Scanner(System.in);

    double numero1, numero2;

    System.out.println("Introduce el primer número:");

    numero1 = sc.nextDouble();

    System.out.println("Introduce el segundo número:");

    numero2 = sc.nextDouble();

    System.out.println("Números introducido: " + numero1 + "  " + numero2);

    System.out.println

     (numero1 + " + " + numero2 + " = " + (numero1+numero2));

    System.out.println

     (numero1 + " - " + numero2 + " = " + (numero1-numero2));

    System.out.println

     (numero1 + " * " + numero2 + " = " + numero1*numero2);

  }
```

}

Para modificar el orden de ejecución de las instrucciones de un programa Java se utilizan las estructuras condicionales y repetitivas.

2. ESTRUCTURA CONDICIONAL, ALTERNATIVA O SELECTIVA

La estructura condicional determina si se ejecutan unas instrucciones u otras según se cumpla o no una determinada condición.

En java la estructura condicional se implementa mediante:

- Instrucción if.
- Instrucción switch.
- Operador condicional ? :

2.1 INSTRUCCION if

Puede ser del tipo:

- Condicional simple: if
- Condicional doble: if ... else ...
- Condicional múltiple: if .. else if ..

La condición debe ser una **expresion booleana** es decir debe dar como resultado un valor booleano (**true ó false**).

Condicional simple: se evalúa la condición y si ésta se cumple se ejecuta una determinada acción o grupo de acciones. En caso contrario se saltan dicho grupo de acciones.

```
if(expresión_booleana){
  instrucción 1
  instrucción 2
    .......
 }
```

Si el bloque de instrucciones tiene **una sola instrucción** no es necesario escribir las llaves { } aunque para evitar confusiones se recomienda escribir las llaves siempre.

Ejemplo de programa Java con estructura condicional: Programa que pide por teclado la nota obtenida por un alumno y muestra un mensaje si el alumno ha aprobado.

```
/*
 * Programa que pide una nota por teclado y muestra un mensaje si la nota es
 * mayor o igual que 5
 */
import java.util.*;
public class Ejemplo0If {
   public static void main( String[] args ){
```

```
      Scanner sc = new Scanner( System.in );
      System.out.print("Nota: ");
      int nota = sc.nextInt();
      if (nota >= 5 ){
         System.out.println("Enorabuena!!");
         System.out.println("Has aprobado");
      }
   }
}
```

Condicional doble: Se evalúa la condición y si ésta se cumple se ejecuta una determinada instrucción o grupo de instrucciones. Si no se cumple se ejecuta otra instrucción o grupo de instrucciones.

```
if(expresión booleana){
  instrucciones 1
}
else{
     instrucciones 2
}
```

Ejemplo de programa Java que contiene una estructura condicional doble: Programa que lee la nota de un alumno y muestra si el alumno ha aprobado o no.

```
/*
 * Programa que pide una nota por teclado y muestra si se ha aprobado o no
 */
import java.util.*;
public class Ejemplo0If {
   public static void main( String[] args ){
      Scanner sc = new Scanner( System.in );
      System.out.print("Nota: ");
      int nota = sc.nextInt();
      if (nota >= 5 ){
         System.out.println("Enorabuena!!");
         System.out.println("Has aprobado");
      }
      else
         System.out.println("Lo Siento, has suspendido");
   }
}
```

Otro ejemplo de programa Java que contiene una estructura condicional doble: Calcular si un número es par. El programa lee un número por teclado y muestra un mensaje indicando si es par o impar.

```
/*
 * programa que pide un número por teclado y calcula si es par o impar
 */
import java.util.*;
public class EjemploIf {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int num;
      System.out.println("Introduzca numero: ");
      num = sc.nextInt();
      if ((num%2)==0)
        System.out.println("PAR");
      else
         System.out.println("IMPAR");
   }
}
```

Condicional múltiple: Se obtiene anidando sentencias if ... else. Permite construir estructuras de selección más complejas.

```
if (expresion_booleana1)
        instruccion1;
else if (expresion_booleana2)
        instruccion2;
     else
        instruccion3;
```

Cada else se corresponde con el if más próximo que no haya sido emparejado.
Una vez que se ejecuta un bloque de instrucciones, la ejecución continúa en la siguiente instrucción que aparezca después de las sentencias if .. else anidadas.

Ejemplo de programa Java que contiene una estructura condicional múltiple: Programa que lee una hora (número entero) y muestra un mensaje según la hora introducida.

```
/*
 * Programa que muestra un saludo distinto según la hora introducida
 */
import java.util.*;
public class Ejemplo2If {
   public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     int hora;
```

```
    System.out.println("Introduzca una hora (un valor entero): ");
    hora = sc.nextInt();
    if (hora >= 0 && hora < 12)
       System.out.println("Buenos días");
    else if (hora >= 12 && hora < 21)
        System.out.println("Buenas tardes");
    else if (hora >= 21 && hora < 24)
         System.out.println("Buenas noches");
    else
         System.out.println("Hora no válida");
  }
}
```

Ejemplo de programa Java que contiene una estructura condicional múltiple: Programa que lee una nota (número entero entre 0 y 10) y muestra la calificación equivalente en forma de texto.

```
/*
 * programa que lee una nota y escribe la calificación correspondiente
 */
import java.util.*;
public class Ejemplo3If {
   public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    double nota;
    System.out.println("Introduzca una nota entre 0 y 10: ");
    nota = sc.nextDouble();
    System.out.println("La calificación del alumno es ");
    if(nota < 0 || nota > 10)
      System.out.println("Nota no válida");
    else if(nota==10)
        System.out.println("Matrícula de Honor");
    else if (nota >= 9)
        System.out.println("Sobresaliente");
    else if (nota >= 7)
        System.out.println("Notable");
    else if (nota >= 6)
        System.out.println("Bien");
    else if (nota >= 5)
        System.out.println("Suficiente");
    else
        System.out.println("Suspenso");
```

```
  }
}
```

Comparar String en Java

Para comparar el contenido de dos Strings en Java se usa el método **equals**:

```
if ((cadena1.equals(cadena2))
```

En caso de que una cadena coincida exactamente con una constante se puede usar ==

```
String nombre = "Lucas";
if (nombre == "Lucas")
```

Para comparar Strings en el orden alfabético se usa el método **compareTo**

```
if (cadena1.compareTo(cadena2) < 0)  // cadena1 antes que cadena2
if (cadena1.compareTo(cadena2) > 0)  // cadena1 después que cadena2
if (cadena1.compareTo(cadena2) == 0)  // cadena1 igual que cadena2
```

2.2 INSTRUCCION switch

Se utiliza para seleccionar una de entre múltiples alternativas.

La forma general de la instrucción switch en Java es la siguiente:

```
switch (expresión){
        case valor 1:
                instrucciones;
                break;
        case valor 2:
                instrucciones;
                break;
        . . .
        default:
                instrucciones;
}
```

La instrucción switch se puede usar con datos de tipo byte, short, char e int. También con tipos enumerados y con las clases envolventes Character, Byte, Short e Integer. A partir de Java 7 también pueden usarse datos de tipo String en un switch.

Funcionamiento de la instrucción switch:

- Primero se evalúa la expresión y salta al case cuya constante coincida con el valor de la expresión.
- Se ejecutan las instrucciones que siguen al case seleccionado hasta que se encuentra un break o hasta el final del switch. El break produce un salto a la siguiente instrucción a continuación del switch.
- Si ninguno de estos casos se cumple se ejecuta el bloque default (si existe). No es obligatorio que exista un bloque default y no tiene porqué ponerse siempre al final, aunque es lo habitual.

  Ejemplo de programa Java que contiene una instrucción switch: Programa que lee por teclado un mes (número entero) y muestra el nombre del mes.

```
/*
 * Programa que pide un número de mes y muestra el nombre correspondiente
 */
import java.util.*;
public class Ejemplo0Switch {
   public static void main(String[] args) {
      int mes;
      Scanner sc = new Scanner(System.in);
      System.out.print("Introduzca un numero de mes: ");
      mes = sc.nextInt();
      switch (mes)
      {
            case 1: System.out.println("ENERO");
                    break;
            case 2: System.out.println("FEBRERO");
                    break;
            case 3: System.out.println("MARZO");
                    break;
            case 4: System.out.println("ABRIL");
                    break;
            case 5: System.out.println("MAYO");
                    break;
            case 6: System.out.println("JUNIO");
                    break;
            case 7: System.out.println("JULIO");
                    break;
            case 8: System.out.println("AGOSTO");
                    break;
            case 9: System.out.println("SEPTIEMBRE");
                    break;
            case 10: System.out.println("OCTUBRE");
                     break;
            case 11: System.out.println("NOVIEMBRE");
                     break;
            case 12: System.out.println("DICIEMBRE");
                     break;
            default : System.out.println("Mes no válido");
      }
   }
}
```

Ejemplo de programa Java que contiene una instrucción switch: Programa que lee dos números enteros por teclado y un operador (de tipo carácter) y muestra el resultado de la operación.

```
/*
 * Programa que pide dos números y un operador y muestra el resultado
 */
import java.util.*;
import java.io.*;
public class Ejemplo1Switch {
   public static void main(String[] args) throws IOException{
      int A,B, Resultado = 0 ;
      char operador;
      boolean calculado = true;
      Scanner sc = new Scanner(System.in);
      System.out.print("Introduzca un numero entero:");
      A = sc.nextInt();
      System.out.print("Introduzca otro numero entero:");
      B = sc.nextInt();
      System.out.print("Introduzca un operador (+,-,*,/):");
      operador = (char)System.in.read();
      switch (operador) {
            case '-' : Resultado = A - B;
                    break;
            case '+' : Resultado = A + B;
                    break;
            case '*' : Resultado = A * B;
                    break;
            case '/' : if(B!=0)
                      Resultado = A / B;
                    else{
                      System.out.println("\nNo se puede dividir por cero");
                      calculado = false;
                    }
                    break;
            default : System.out.println("\nOperador no valido");
                   calculado = false;

      }
      if(calculado){
         System.out.println("\nEl resultado es: " + Resultado);
      }
```

```
  }
}
```

2.3 OPERADOR CONDICIONAL ? :

Se puede utilizar en sustitución de la sentencia de control if-else.

Los forman los caracteres **?** y **:**

Se utiliza de la forma siguiente:

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

Si expresión1 es cierta entonces se evalúa expresión2 y éste será el valor de la expresión condicional. Si expresión1 es falsa, se evalúa expresión3 y éste será el valor de la expresión condicional.

Ejemplo de programa Java que contiene un operador condicional: Programa que calcula y muestra si un número que se lee por teclado es par o impar.

```
/*
 * programa que pide un número por teclado y calcula si es par o impar
 */
import java.util.*;
public class Ejemplo1OperadorCondicional {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int num;
      System.out.println("Introduzca numero: ");
      num = sc.nextInt();
      System.out.println((num%2)==0 ? "PAR" : "IMPAR");
   }
}
```

3. ESTRUCTURA ITERATIVA O REPETITIVA

Permiten ejecutar de forma repetida un bloque específico de instrucciones.

Las instrucciones se repiten mientras o hasta que se cumpla una determinada condición. Esta condición se conoce como **condición de salida**.

Tipos de estructuras repetitivas:

- ciclo while
- ciclo do – while
- ciclo for

3.1 CICLO WHILE

Las instrucciones se repiten mientras la condición sea cierta. La condición **se comprueba al principio** del bucle por lo que las acciones se pueden ejecutar **0 ó más veces**.

La ejecución de un bucle while sigue los siguientes pasos:

1. Se evalúa la condición.
2. Si el resultado es false las instrucciones no se ejecutan y el programa sigue ejecutándose por la siguiente instrucción a continuación del while.

3. Si el resultado es true se ejecutan las instrucciones y se vuelve al paso 1

Ejemplo de programa Java que contiene una instrucción while:
Programa que lee números por teclado. La lectura acaba cuando el número introducido sea negativo. El programa calcula y muestra la suma de los números leídos.

```
/*
 * Programa que lee números hasta que se lee un negativo y muestra la
 * suma de los números leídos
 */
import java.util.*;
public class Ejemplo1While {
   public static void main(String[] args) {
      int suma = 0, num;
      Scanner sc = new Scanner(System.in);
      System.out.print("Introduzca un número: ");
      num = sc.nextInt();
      while (num >= 0){
           suma = suma + num;
           System.out.print("Introduzca un número: ");
           num = sc.nextInt();
      }
      System.out.println("La suma es: " + suma );
   }
}
```

Ejemplo de programa Java que contiene una instrucción while:
Programa que lee un número entero N y muestra N asteriscos.

```
/*
 * programa que lee un número n y muestra n asteriscos
 */
import java.util.*;
public class Ejemplo2While {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n, contador = 0;
      System.out.print("Introduce un número: ");
      n = sc.nextInt();
      while (contador < n){
           System.out.println(" * ");
           contador++;
      }
   }
```

```
}
```

Ejemplo de programa Java con una instrucción while:

```
/*
 * programa que muestra una tabla de equivalencias entre
 * grados Fahrenheit y grados celsius
 */
public class Ejemplo3While {
   public static void main(String[] args) {
      final int VALOR_INICIAL = 10;  // limite inf. tabla
      final int VALOR_FINAL = 100;  // limite sup. tabla
      final int PASO = 10 ; // incremento
      int fahrenheit;
      double celsius;
      fahrenheit = VALOR_INICIAL;
      System.out.printf("Fahrenheit \t Celsius \n");
      while (fahrenheit <= VALOR_FINAL ){
           celsius = 5*(fahrenheit - 32)/9.0;
           System.out.printf("%7d \t %8.3f \n", fahrenheit, celsius);
           fahrenheit += PASO;
      }
   }
}
```

3.2 CICLO DO – WHILE

Las instrucciones se ejecutan mientras la condición sea cierta.

La condición **se comprueba al final** del bucle por lo que el bloque de instrucciones se ejecutarán **al menos una vez**. Esta es la diferencia fundamental con la instrucción while. Las instrucciones de un bucle while es posible que no se ejecuten si la condición inicialmente es falsa.

La ejecución de un bucle do - while sigue los siguientes pasos:

1. Se ejecutan las instrucciones a partir de do{
2. Se evalúa la condición.
3. Si el resultado es false el programa sigue ejecutándose por la siguiente instrucción a continuación del while.
4. Si el resultado es true se vuelve al paso 1

Ejemplo de programa Java que contiene una instrucción do while:

Programa que lee un número entero N. El número debe ser menor que 100.

```
/*
 * Programa que obliga al usuario a introducir un número menor que 100
 */
import java.util.*;
```

```
public class Ejemplo1DoWhile {
   public static void main(String[] args) {
      int valor;
      Scanner in = new Scanner( System.in );
      do {
         System.out.print("Escribe un entero < 100: ");
         valor = in.nextInt();
      }while (valor >= 100);
      System.out.println("Ha introducido: " + valor);
   }
}
```

Ejemplo de programa Java con una instrucción do while:

```
/*
 * Programa que lee un número entre 1 y 10 ambos incluidos
 */
import java.util.*;
public class Ejemplo2DoWhile {
   public static void main(String[] args) {
      int n;
      Scanner sc = new Scanner( System.in );
      do {
         System.out.print("Escribe un número entre 1 y 10: ");
         n = sc.nextInt();
      }while (n<1 || n >10);
      System.out.println("Ha introducido: " + n);
   }
}
```

3.3 CICLO FOR

Hace que una instrucción o bloque de instrucciones se repitan un **número determinado de veces mientras se cumpla la condición**.

La estructura general de una instrucción for en Java es la siguiente:

```
for(inicialización; condición; incremento/decremento){
        instrucción 1;
        ..........
        instrucción N;
}
```

A continuación de la palabra for y entre paréntesis debe haber siempre **tres zonas separadas por punto y coma**:

- zona de inicialización.
- zona de condición
- zona de incremento ó decremento.

Si en alguna ocasión no es necesario escribir alguna de ellas se pueden dejar en blanco, pero los dos punto y coma deben aparecer.

Inicialización es la parte en la que la variable o variables de control del bucle toman su valor inicial. Puede haber una o más instrucciones en la inicialización, separadas por comas. La inicialización se realiza solo una vez.

Condición es una expresión booleana que hace que se ejecute la sentencia o bloque de sentencias mientras que dicha expresión sea cierta. Generalmente en la condición se compara la variable de control con un valor límite.

Incremento/decremento es una expresión que decrementa o incrementa la variable de control del bucle.

La ejecución de un bucle for sigue los siguientes pasos:

1. Se inicializa la variable o variables de control (inicialización)
2. Se evalúa la condición.
3. Si la condición es cierta se ejecutan las instrucciones. Si es falsa, finaliza la ejecución del bucle y continúa el programa en la siguiente instrucción después del for.
4. Se actualiza la variable o variables de control (incremento/decremento)
5. Se vuelve al punto 2.

Ejemplo de programa Java que contiene una instrucción for:

```
/*
 * programa que muestra los números del 1 al 10
 */
public class Ejemplo0For {
   public static void main(String[] args) {
      int i;
      for(i=1; i<=10;i++)
         System.out.println(i + " ");
   }
}
```

La instrucción for del ejemplo anterior la podemos interpretar así:

Asigna a i el valor inicial 1, mientras que i sea menor o igual a 10 muestra i + “ “ , a continuación incrementa el valor de i y comprueba de nuevo la condición.

Ejemplo de programa Java con una instrucción for:

```
/*
 * programa que muestra los números del 10 al 1
 */
public class Ejemplo2For {
   public static void main(String[] args) {
      int i;
      for(i=10; i>0;i--)
         System.out.println(i + " ");
   }
}
```

Ejemplo de programa Java con una instrucción for:

```
/*
 * programa que muestra una tabla de equivalencias entre
 * grados Fahrenheit y grados celsius
 */
public class Ejemplo1For {
   public static void main(String[] args) {
      final int VALOR_INICIAL = 10; // limite inf. tabla
      final int VALOR_FINAL = 100; // limite sup. tabla
      final int PASO = 10 ; // incremento
      int fahrenheit;
      double celsius;
      fahrenheit = VALOR_INICIAL;
      System.out.printf("Fahrenheit \t Celsius \n");
      for (fahrenheit = VALOR_INICIAL; fahrenheit <= VALOR_FINAL;
           fahrenheit+= PASO) {
          celsius = 5*(fahrenheit - 32)/9.0;
          System.out.printf("%7d \t %8.3f \n", fahrenheit, celsius);
      }
   }
}
```

En las zonas de inicialización e incremento/decremento puede aparecer más de una variable. En ese caso deben ir separadas por comas.

Ejemplo:

```
/*
 * programa que muestra el valor de a, b y su suma mientras que la suma de
 * ambas es menor de 10. En cada iteración el valor de a se incrementa en
 * 1 unidad y el de b en 2
 */
public class Ejemplo3For {
   public static void main(String[] args) {
      int a, b;
      for(a = 1, b = 1; a + b < 10; a++, b+=2){
         System.out.println("a = " + a + "  b = " + b + "  a + b = " + (a+b));
      }
   }
}
```

La salida de este programa es:

```
a = 1  b = 1  a + b = 2
a = 2  b = 3  a + b = 5
```

a = 3 b = 5 a + b = 8

Aunque la instrucción repetitiva for, al igual que las instrucciones while y do- while, se puede utilizar para realizar repeticiones cuando no se sabe a priori el número de pasadas por el bucle, esta instrucción es especialmente indicada para bucles donde se conozca el número de pasadas.

Como regla práctica podríamos decir que las instrucciones while y do-while se utilizan generalmente cuando no se conoce a priori el número de pasadas, y la instrucción for se utiliza generalmente cuando sí se conoce el número de pasadas.

Se ha de tener cuidado con escribir el punto y coma (;) después del paréntesis final del bucle for. Un bucle for generalmente no lleva punto y coma final.

Por ejemplo el bucle:

```
int i;
for (i = 1; i <= 10; i++);
{
    System.out.println("Elementos de Programación");
}
```

no visualiza la frase "Elementos de Programación" 10 veces, ni produce un mensaje de error por parte del compilador.

En realidad lo que sucede es que se visualiza una vez la frase "Elementos de Programación", ya que aquí la sentencia for es una sentencia vacía al terminar con un punto y coma (;).

La sentencia for en este caso hace que i empiece en 1 y acabe en 11 y tras esas iteraciones, se ejecuta la sentencia

System.out.println("Elementos de Programación");

3.4 BUCLES INFINITOS EN JAVA

Java permite la posibilidad de construir bucles infinitos, los cuales se ejecutarán indefinidamente, a no ser que provoquemos su interrupción. Tres ejemplos:

```
for(;;){
   instrucciones
}
for(;true;){
    instrucciones
}
while(true){
      instrucciones
}
```

3.5 BUCLES ANIDADOS

Bucles anidados son aquellos que incluyen instrucciones for, while o do-while unas dentro de otras.

Debemos tener en cuenta que las variables de control que utilicemos deben ser distintas.

Los anidamientos de estructuras tienen que ser correctos, es decir, que una estructura anidada dentro de otra lo debe estar totalmente.

Ejemplo de programa Java con bucles anidados:

```
/*
 * Programa que dibuja un rectángulo sólido de asteriscos.
 * El número de filas y columnas se pide por teclado
 */
import java.util.*;
public class Ejemplo1BuclesAnidados {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int filas, columnas;
      //leer número de filas hasta que sea un número > 0
      do{
        System.out.print("Introduce número de filas: ");
        filas = sc.nextInt();
      }while(filas<1);
      //leer número de columnas hasta que sea un número > 0
      do{
        System.out.print("Introduce número de columnas: ");
        columnas = sc.nextInt();
      }while(columnas<1);
      for(int i = 1; i<=filas; i++){    //filas
         for(int j = 1; j<=columnas; j++){   //columnas
             System.out.print(" * ");
         }
         System.out.println();
      }

   }
}
```

La salida de este programa para filas = 6 y columnas = 10 es:

```
 * * * * * * * * * * * *
 * * * * * * * * * * * *
 * * * * * * * * * * * *
 * * * * * * * * * * * *
 * * * * * * * * * * * *
 * * * * * * * * * * * *
```

Ejemplo de programa Java con bucles anidados:

```
/*
 * Programa que muestra una tabla con las potencias de x (x x2 x3 x4)
```

```
 * para valores de x desde 1 hasta XMAX
 */
public class JavaApplication22 {
   public static void main(String[] args) {
      final int XMAX = 10;
      int x, n;
      //mostrar la cabecera de la tabla
      System.out.printf("%10s%10s%10s%10s%n", "x","x^2","x^3","x^4");
      for (x = 1; x <= XMAX; x++){   //filas
         for (n = 1; n <= 4; n++){   //columnas
            System.out.printf("%10.0f", Math.pow(x,n));
         }
         System.out.println();
      }
   }
}
```

El programa muestra por pantalla la siguiente tabla de potencias:

```
         x       x^2       x^3       x^4
         1         1         1         1
         2         4         8        16
         3         9        27        81
         4        16        64       256
         5        25       125       625
         6        36       216      1296
         7        49       343      2401
         8        64       512      4096
         9        81       729      6561
        10       100      1000     10000
```