J2SE
Lenguaje Java

Antonio Espín Herranz

Comentarios

- Existen tres formas diferentes para introducir comentarios:

```
// comentario de una línea

/* Comentario de una o
mas líneas */

/** comentario de documentación */
```

→ **javadoc**

- Podemos generar de nuestros fuentes en java nuestra propia documentación con el mismo formato que la API de java.

Identificadores

- Son nombres de variables, clases o métodos.
- Pueden empezar con una letra, subrayado (_), o símbolo de dólar ($).
- No podemos usar ninguna palabra reservada para declarar un identificador.
- Se distinguen las mayúsculas de las minúsculas y no tienen una longitud máxima.

Ejemplos:

nombre
NomApe
Usr_nombre
_sys_var
$numero

Palabras Reservadas del Lenguaje

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

Operadores

- Aritméticos.
- Unarios.
- Relacionales / Condicionales.
- Sobre bits.
- De Asignación.
- Ternario / Condicional.

Aritméticos

- **+** Suma los operandos.
- **-** Resta el operando de la derecha al de la izquierda.
- ***** Multiplica los operandos.
- ***/*** Divide el operando de la izquierda entre el de la derecha.
- **%** Resto de la división del operando izquierdo entre el derecho.

Unarios

- **+** Indica un valor positivo.
- **-** Negativo, o cambia el signo algebraico.
- **++** Suma 1 al operando, como prefijo o sufijo.
- **--** Resta 1 al operando, como prefijo o sufijo.

Condicionales / Relacionales

- **>** El operando izquierdo es mayor que el derecho.
- **>=** El operando izquierdo es mayor o igual que el derecho.
- **<** El operando izquierdo es menor que el derecho
- **<=** El operando izquierdo es menor o igual que el derecho
- **==** El operando izquierdo es igual que el derecho.
- **!=** El operando izquierdo es distinto del derecho.

Condicionales / Relacionales

- **&&** Devuelve true si las expresiones izquierda **y** derecha son true. En caso contrario devuelve false.

- **||** Devuelve true o la expresión izquierda o al expresión de la derecha son true. Si ambas son false devuelve false.

- **!** Negación de la expresión, si es true devuelve false y vicebersa.

Sobre bits

- **>>** Desplaza bits del operando hacia la derecha las posiciones indicadas (con signo).
- **<<** Desplaza bits del operando hacia la izquierda las posiciones indicadas.
- **>>>** Desplaza bits del operando hacia la derecha las posiciones indicadas (sin signo).
- **&** Realiza una operación AND lógica entre los dos operandos.
- **|** Realiza una operación OR lógica entre los dos operandos
- **^** Realiza una operación lógica OR Exclusiva entre los dos operandos
- **~** Complementario del operando (unario).

Tablas lógicas: & | ^

A	B	A & B	A | B	A ^ B
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Ejemplos de rotación de bits

- >> Equivale a dividir.
- << Equivale a multiplicar.

- 128 >> 1 equivale a $128/2^1 = 64$

- 256 >> 4 equivale a $256/2^4 = 16$

- -256 >> 4 equivale a $-256/2^4 = -16$

- 2 << 2 equivale a $2 * 2^2 = 16$

Asignación

- **+=** Ejemplo x += 5 → x = x + 5
- **-=** y -= 6 → y = y - 6
- ***=**
- **/=**
- **%=**
- **&=**
- **|=**
- **^=**
- **<<=**
- **>>=**
- **>>>=**
- Representan una operación y una asignación.

Ternario / condicional

- Expresión **?** Sentencia1 **:** Sentencia2
- Si expresión es true evalúa sentencia1
- En caso contrario evalúa sentencia2
- Ejemplo:

```
cociente = (denominador == 0) ? 0 : numerador / denominador;
```

Variables

- Las variables se utilizan en la programación Java para almacenar datos que varían durante la ejecución del programa.
- Para usar una variable, hay que indicarle al compilador el tipo y nombre de esa variable, declaración de la variable.
- El tipo de la variable determinará el conjunto de valores que se podrán almacenar en la variable y el tipo de operaciones que se podrán realizar con ella.
- Por ejemplo, el tipo int solamente puede contener números completos (enteros). En Java, todas las variables de tipo int contienen valores con signo. C++, por el contrario, soporta el uso de tipos enteros con y sin signo (todos positivos).
- int i, Point p;

Constantes

- Java soporta constantes con nombre y la forma de crearlas es:

```
static final float PI = 3.14159;
// No puede modificar el valor

// Delante de static colocaríamos el modificador de acceso.

// final es una palabra reservada. En herencia
   nos indica que ya no se puede propagar mas.
   En el caso de un método.
```

Expresiones

- Una expresión es una determinada combinación de operadores y operandos que se evalúan para obtener un resultado particular.
- Los operandos pueden ser variables, constantes o llamadas a métodos.
- Una llamada a un método evalúa el valor devuelto por el método y el tipo de una llamada a un método es el tipo devuelto por ese método.

Tipos de Datos

- Java define ocho tipos primitivos de datos.
  - Lógica boolean
  - Texto char
  - Entero byte, short, int y long
  - Real double y float

Lógicas - boolean

- El tipo de dato boolean tiene dos posibles valores, true y false.

- Por ejemplo, la sentencia

```
boolean truth = true;
```

- Declara la variable truth como tipo boolean y le asigna el valor de true.

Texto - char

char

- Representa un carácter Unicode de 16-bit sin signo.
- Tiene que ir entre comillas simples (‘’).
- Utiliza la siguiente notación:

‘a’

‘\t’ Un tabulador

‘\u????’ Un carácter específico de Unicode, ????, se reemplaza exactamente con cuatro dígitos en hexadecimal.

Enteros - byte, short, int y long

- Tres formas - decimal, octal o hexadecimal.

2	El valor decimal es dos.
077	El cero inicial indica que es un valor en notación octal.
0xBAAC	0x indica un valor en hexadecimal.

- Por defecto tiene un int.

- Con La letra “L” o “l” se define un long.

Rango de los Enteros

- Cada tipo de entero tiene el siguiente rango

Longitud del entero	Nombre o Tipo	Rango
8 bits	byte	-2^{7} ...2^{7} -1
16 bits	short	-2^{15} ...2^{15} -1
32 bits	int	-2^{31} ...2^{31} -1
64 bits	long	-2^{63} ...2^{63} -1

Reales - float y double

- Por defecto es double.
- Una constante es de punto flotante si incluye un punto decimal o
  - una E o e (valor exponencial)
  - una F o f (float)
  - una D o d (double)

3.14	Un número real simple (double)
6.02E23	Número real grande
2.718F	Un valor real de tamaño float
123.4E+306D	Un real muy grande con una D redundante

Rango de los Reales

- Cada tipo de real tiene el siguiente rango

Longitud del float	Nombre del tipo
32 bits	float
64 bits	double

Cadenas

Objeto String

- **No es un tipo básico, es una clase**.
- Tiene que ir entre comillas dobles (“”).

  “Esto es un String”

- Se puede utilizar:

  String saludo = “Buenos días \n”;

Conversión de Tipos

- Si en una asignación se pierde información, el programador tiene que confirmar la asignación con una conversión.

- La asignación de short a char requiere de una conversión explícita.

```
long bigValue = 99L;
int squashed = (int)(bigValue);
long bigval = 6;          // 6 es de tipo entero. OK.
int smallval = 99L;       // 99L es un long. Incorrecto.
float z = 12.414F;        // 12.4141 es un float. OK.
float zl = 12.414;        // 12.414 es un double. Incorrecto.
```

Propagación y conversión de expresiones

- Las variables se pueden convertir automáticamente a un tipo superior (por ejemplo, de int a long).

- Una expresión es una asignación compatible, si el tipo de la variable es, al menos, tan grande como el tipo de la expresión (el mismo número de bits).

Separadores

- **( )** paréntesis. Para contener listas de parámetros en la definición y llamada a métodos. También se utiliza para definir precedencia en expresiones, contener expresiones para control de flujo y rodear las conversiones de tipo.

- **{ }** llaves. Para contener los valores de matrices inicializadas automáticamente. También se utiliza para definir un bloque de código, para clases, métodos y ámbitos locales.

- **[ ]** corchetes. Para declarar tipos matriz. También se utiliza cuando se referencian valores de matriz.

Separadores

- **;** punto y coma. Separa sentencias.
- **,** coma. Separa identificadores consecutivos en una declaración de variables. También se utiliza para encadenar sentencias dentro de una sentencia for.
- **.** punto. Para separar nombres de paquete de subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

Secuencias de escape

\n	→	Nueva Linea.
\t	→	Tabulador.
\r	→	Retroceso de Carro.
\f	→	Comienzo de Pagina.
\\	→	El caracter \
\'	→	El caracter '
\"	→	El caracter "

Impresión en Pantalla

- La clase System está predefinida y proporciona acceso al sistema.
- Esta clase define:
  - in: entrada standard o teclado.
  - out: salida standard o consola.
  - err: salida standard de error (a la consola).
- Impresión:
  - System.out.println(“Una frase …”);
  - System.out.print(“Una frase … sin salto de línea”);
  - System.out.println(“El valor de i: \n“ + i);

- Lectura de números enteros:
  - System.in.read() throws IOException

Estructura de un programa en Java

- notepad **Ejemplo**.java

```
public class Ejemplo {
    public static void main(String args[]){
        System.out.println(“Ejemplo en java”);
    }
}
```

- javac **Ejemplo**.java
- java **Ejemplo**
- Ejemplo en java

Control de Flujo

- Condicionales:
  - if
  - switch
- Bucles:
  - for
  - while
  - do while

Condicional if

- Las sentencias if y else

```
if (expresion boolean) {
        Sentencia o bloque;
}

if (condicion cierta) {
        Sentencia o bloque;
} else {
        Sentencia o bloque;
}
```

Condicional switch

- La sentencia switch

```
switch (expr1) {
        case expr2:
                Sentencias;
                break;
        case expr3:
                Sentencias;
                break;
        default:
                Sentencias;
                break;
}
```

Bucle while

```
[inicialización]
while (condición){  // La condición de permanencia
      Sentencias
      [incremento]
}
```

Ejemplo:

```
int i = 0;
while ( i < 10){
     ….
     i++;
}
```

Bucle for

```
for( inicialización; condición; iteración ) {
        sentencias;
}
```

- Inicialización de la variable de control.
- Comprobación del valor de la variable de control en una expresión condicional.
- **Para no tener problemas con la interpretación del bucle for, traducir la condición *por mientras se cumpla la condición*.**
- Actualización de la variable de control.

Ejemplos:

```
for (; ;);
for (int i = 0, int j = 5 ; i < 10 ; i++, j+=5){ … }
```

Bucle do while

```
[inicialización;]
   do {
      sentencias;
      [iteración;]
   } while(condición);  // Condición de permanencia
```

Ejemplo:

```
int i = 0;
do {
    i ++;
} while (i < 10);
```

Sentencias de Salto Incondicional

- **break**: Rompe el bucle: for, while, do while.
- **continue**: Fuerza una nueva iteración en un bucle. En while / for vuelve a evaluar la condición, y en el caso de do while, ejecuta la primera instrucción del bucle.
- **return**: Devuelve el valor de una función. Rompe la ejecución de un método.

break / continue

- Estas dos instrucciones se pueden utilizar con etiquetas, de tal forma que podamos controlar el punto donde queremos romper o continuar:

```
bucle1:
for (int i = 0; i < 10; i++) {
  System.out.println("Bucle i. i = "+i);
  for (int j = 0; j < 10; j++) {
    System.out.println("Bucle j. j = "+j);
    for (int k = 0; k < 10; k++) {
      System.out.println("Bucle k. k = "+k);
      break bucle1;  // Si quitamos la etiqueta, sólo rompe el bucle de la k.
    }
  }
}
```