

Los operadores (aritméticos)

Todos los lenguajes de programación permiten realizar operaciones entre los tipos de datos básicos: suma, resta, producto, cociente, etc., de dos números. Otros lenguajes como el BASIC y Java permiten "sumar", concatenar cadenas de caracteres.

En la página titulada "La primera aplicación", hemos aprendido a crear un proyecto nuevo, y la clase que describe la aplicación mínima que contiene la función estática *main*. Luego, le hemos añadido código para dar cierta funcionalidad a la aplicación.

Los operadores aritméticos

Java tiene cinco operadores aritméticos cuyo significado se muestra en la tabla adjunta

Operador	Nombre	Ejemplo
+	Suma	3+4
-	Diferencia	3-4
*	Producto	3*4
/	Cociente	20/7
%	Módulo	20%7

El cociente entre dos enteros da como resultado un entero. Por ejemplo, al dividir 20 entre 7 nos da como resultado 2.

El operador módulo da como resultado el resto de la división entera. Por ejemplo 20%7 da como resultado 6 que es el resto de la división entre 20 y 7.

El operador módulo también se puede emplear con números reales. Por ejemplo, el cociente entre 7.5 y 3.0 es 2.5 y el resto es cero, es decir, $7.5 = 3.0 \cdot 2.5 + 0$. El operador módulo, funciona de la siguiente forma $7.5 = 3.0 \cdot 2 + 1.5$, calcula la diferencia entre el dividendo (7.5) y el producto del divisor (3.0) por la parte entera (2) del cociente, devolviendo 1.5. Así pues, la operación $7.5 \% 3.0$ da como resultado 1.5.

El operador asignación

Nos habremos dado cuenta que el operador más importante y más frecuentemente usado es el operador asignación `=`, que hemos empleado para la inicialización de las variables. Así,

```
int numero;  
numero=20;
```

la primera sentencia declara una variable entera de tipo `int` y le da un nombre (*numero*). La segunda sentencia usa el operador asignación para inicializar la variable con el número 20.

Consideremos ahora, la siguiente sentencia.

```
a=b;
```

que asigna a a el valor de b . A la izquierda siempre tendremos una variable tal como a , que recibe valores, a la derecha otra variable b o expresión que tiene un valor. Por tanto, tienen sentido las expresiones

```
a=1234;  
double area=calculaArea(radio);  
superficie=ancho*alto;
```

Sin embargo, no tienen sentido las expresiones

```
1234=a;  
calculaArea(radio)=area;
```

Las asignaciones múltiples son también posibles. Por ejemplo, es válida la sentencia

```
c=a=b;           //equivalente a c=(a=b);
```

la cual puede ser empleada para inicializar en la misma línea varias variables

```
c=a=b=321;           //asigna 321 a a, b y c
```

El operador asignación se puede combinar con los operadores aritméticos

Expresión	Significado
$x+=y$	$x=x+y$
$x-=y$	$x=x-y$
$x*=y$	$x=x*y$
$x/=y$	$x=x/y$

Así, la sentencia

```
x=x+23;
```

evalúa la expresión $x+23$, que es asignada de nuevo a x . El compilador lee primero el contenido de la porción de memoria nombrada x , realiza la suma y guarda el resultado en la misma porción de memoria. Se puede escribir la sentencia anterior de una forma equivalente más simple

```
x+=23;
```

Concatenación de strings

En Java se usa el operador $+$ para concatenar cadenas de caracteres o strings. Veremos en el siguiente apartado una sentencia como la siguiente:

```
System.out.println("la temperatura centígrada es "+tC);
```

El operador + cuando se utiliza con strings y otros objetos, crea un solo string que contiene la concatenación de todos sus operandos. Si alguno de los operandos no es una cadena, se convierte automáticamente en una cadena. Por ejemplo, en la sentencia anterior el número del tipo **double** que guarda la variable *tC* se convierte en un string que se añade al string "la temperatura centígrada es ".

Como veremos más adelante, un objeto se convierte automáticamente en un string si su clase redefine la función miembro [*toString*](#) de la clase base *Object*.

Como vemos en el listado, para mostrar un resultado de una operación, por ejemplo, la suma de dos números enteros, escribimos

```
iSuma=ia+ib;
System.out.println("El resultado de la suma es "+iSuma);
```

Concatena una cadena de caracteres con un tipo básico de dato, que convierte automáticamente en un string.

El operador += también funciona con cadenas.

```
String nombre="Juan ";
nombre+="García";
System.out.println(nombre);
```

```
public class OperadorAp {
    public static void main(String[] args) {
        System.out.println("Operaciones con enteros");
        int ia=7, ib=3;
        int iSuma, iResto;
        iSuma=ia+ib;
        System.out.println("El resultado de la suma es "+iSuma);
        int iProducto=ia*ib;
        System.out.println("El resultado del producto es "+iProducto);
        System.out.println("El resultado del cociente es "+(ia/ib));
        iResto=ia%ib;
        System.out.println("El resto de la división entera es "+iResto);

        System.out.println("*****");
        System.out.println("Operaciones con números decimales");
        double da=7.5, db=3.0;
        double dSuma=da+db;
        System.out.println("El resultado de la suma es "+dSuma);
        double dProducto=da*db;
        System.out.println("El resultado del producto es "+dProducto);
        double dCociente=da/db;
        System.out.println("El resultado del cociente es "+dCociente);
        double dResto=da%db;
        System.out.println("El resto de la división es "+dResto);
    }
}
```

La precedencia de operadores

El lector conocerá que los operadores aritméticos tienen distinta precedencia, así la expresión

$$a+b*c$$

es equivalente a

$$a+(b*c)$$

ya que el producto y el cociente tienen mayor precedencia que la suma o la resta. Por tanto, en la segunda expresión el paréntesis no es necesario. Sin embargo, si queremos que se efectúe antes la suma que la multiplicación tenemos de emplear los paréntesis

$$(a+b)*c$$

Para realizar la operación $\frac{a}{bc}$ escribiremos

$$a/(b*c);$$

o bien,

$$a/b/c;$$

En la mayoría de los casos, la precedencia de las operaciones es evidente, sin embargo, en otros que no lo son tanto, se aconseja emplear paréntesis. Como ejemplo, estudiemos un programa que nos permite convertir una temperatura en grados Fahrenheit en su equivalente en la escala Celsius. La fórmula de conversión es

$$tC = \frac{(tF - 32) * 5}{9}$$

cuya codificación es

$$tC = (tF - 32) * 5 / 9;$$

Las operaciones se realizan como suponemos, ya que si primero se realizase el cociente $5/9$, el resultado de la división entera sería cero y el producto por el resultado de evaluar el paréntesis sería también cero. Si tenemos dudas sobre la precedencia de operadores podemos escribir

$$tC = ((tF - 32) * 5) / 9;$$

```
public class PrecedeApp {
    public static void main(String[] args) {
        int tF=80;
        System.out.println("la temperatura Fahrenheit es "+tF);
        int tC=(tF-32)*5/9;
        System.out.println("la temperatura centígrada es "+tC);
    }
}
```

La conversión automática y promoción (casting)

Cuando se realiza una operación, si un operando es entero (**int**) y el otro es de coma flotante (**double**) el resultado es en coma flotante (**double**).

```
int a=5;
double b=3.2;
double suma=a+b;
```

Cuando se declaran dos variables una de tipo **int** y otra de tipo **double**.

```
int entero;
double real=3.20567;
```

¿qué ocurrirá cuando asignamos a la variable *entero* el número guardado en la variable *real*?. Como hemos visto se trata de dos tipos de variables distintos cuyo tamaño en memoria es de 32 y 64 bits respectivamente. Por tanto, la sentencia

```
entero=real;
```

convierte el número real en un número entero eliminando los decimales. La variable *entero* guardará el número 3.

Se ha de tener cuidado, ya que la conversión de un tipo de dato en otro es una fuente frecuente de error entre los programadores principiantes. Ahora bien, supongamos que deseamos calcular la división $7/3$, como hemos visto, el resultado de la división entera es 2, aún en el caso de que tratemos de guardar el resultado en una variable del tipo **double**, como lo prueba la siguiente porción de código.

```
int ia=7;
int ib=3;
double dc=ia/ib;
```

Si queremos obtener una aproximación decimal del número $7/3$, hemos de promocionar el entero *ia* a un número en coma flotante, mediante un procedimiento denominado promoción o *casting*.

```
int ia=7;
int ib=3;
double dc=(double)ia/ib;
```

Como aplicación, consideremos el cálculo del valor medio de dos o más números enteros

```
int edad1=10;
int edad2=15;
double media=(double) (edad1+edad2)/2;
```

El valor medio de 10 y 15 es 12.5, sin la promoción se obtendría el valor erróneo 12.

Existen también conversiones implícitas realizadas por el compilador, por ejemplo cuando pasamos un entero **int** a una función cuyo único parámetro es de tipo **long**.

Los operadores unarios

Los operadores unarios son:

- ++ Incremento
- -- Decremento

actúan sobre un único operando. Se trata de uno de los aspectos más confusos para el programador, ya que el resultado de la operación depende de que el operador esté a la derecha $i++$ o a la izquierda $++i$.

Conoceremos, primero el significado de estos dos operadores a partir de las sentencias equivalentes:

```
i=i+1;           //añadir 1 a i
i++;
```

Del mismo modo, lo son

```
i=i-1;           //restar 1 a i
i--;
```

Examinemos ahora, la posición del operador respecto del operando. Consideremos en primer lugar que el operador unario ++ está a la derecha del operando. La sentencia

```
j=i++;
```

asigna a j , el valor que tenía i . Por ejemplo, si i valía 3, después de ejecutar la sentencia, j toma el valor de 3 e i el valor de 4. Lo que es equivalente a las dos sentencias

```
j=i;
i++;
```

Un resultado distinto se obtiene si el operador ++ está a la izquierda del operando

```
j=++i;
```

asigna a j el valor incrementado de i . Por ejemplo, si i valía 3, después de ejecutar la sentencia $j \leftarrow i + 1$ toman el valor de 4. Lo que es equivalente a las dos sentencias

```
++i;  
j=i;
```

```
public class UnarioApp {
    public static void main(String[] args) {
        int i=8;
        int a, b, c;
        System.out.println("\ntantes\tdurante\tdespués");
        i=8; a=i; b=i++; c=i;
        System.out.println("i++\t"+a+'\t'+b+'\t'+c);
        i=8; a=i; b=i--; c=i;
        System.out.println("i--\t"+a+'\t'+b+'\t'+c);

        i=8; a=i; b=++i; c=i;
        System.out.println("++i\t"+a+'\t'+b+'\t'+c);
        i=8; a=i; b=--i; c=i;
        System.out.println("--i\t"+a+'\t'+b+'\t'+c);
    }
}
```

La salida del programa es, la siguiente

	antes	durante	después
i++	8	8	9
i--	8	8	7
++i	8	9	9
--i	8	7	7

La primera columna (antes) muestra el valor inicial de i , la segunda columna (durante) muestra el valor de la expresión, y la última columna (después) muestra el valor final de i , después de evaluarse la expresión.

Se deberá de tener siempre el cuidado de inicializar la variable, antes de utilizar los operadores unarios con dicha variable.