4. Operadores

Objetivos:
a) Describir los operadores (aritméticos, incrementales, de relación, lógicos y de asignación) y los tipos de dato primitivos sobre los que actúan
b) Evaluar expresiones que empleen datos primitivos, operadores y paréntesis
c) Construir expresiones que empleen combinaciones de datos primitivos, operadores y paréntesis

Un operador lleva a cabo operaciones sobre uno (operador *unario*), dos (operador *binario*) o tres (operador *ternario*) datos u *operandos* de tipo primitvo devolviendo un valor determinado también de un tipo primitivo. El tipo de valor devuelto tras la evaluación depende del operador y del tipo de los operandos. Por ejemplo, los operadores *aritméticos* trabajan con operandos numéricos, llevan a cabo operaciones aritméticas básicas y devuelven el valor numérico correspondiente. Los operadores se pueden clasificar en distintos grupos según se muestra en los siguientes apartados.

4.1. Operador asignación

El operador asignación, `=`, es un operador binario que asigna el valor del término de la derecha al operando de la izquierda. El operando de la izquierda suele ser el identificador de una variable. El término de la derecha es, en general, una expresión de un tipo de dato compatible; en particular puede ser una constante u otra variable. Como caso particular, y a diferencia de los demás operadores, este operador no se evalúa devolviendo un determinado valor.

Tabla 4.1 Operador asignación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
=	Operador asignación	`n = 4`	`n vale 4`

No debe confundirse el operador asignación (=) con el operador relacional de igualdad (==) que se verá más adelante. Además Java dispone de otros operadores que combinan la asignación con otras operaciones (operadores aritméticos *combinados*).

En el siguiente código se muestran algunos ejemplos de uso del operador asignación con datos de distintos tipos:

```
/**
 *  Demostracion del operador asignacion
 *  A. Garcia-Beltran - Abril, 2008
 */
public class opAsignacion {
   public static void main(String[] args) {
      int i,j;
      double x;
      char c;
      boolean b;
      String s;
      i = 15;
      j = i;
      x = 12.345;
      c = 'A';
      b = false;
      s = "Hola";
```

```
        System.out.println("i = " + i);
        System.out.println("j = " + j);
        System.out.println("x = " + x);
        System.out.println("c = " + c);
        System.out.println("b = " + b);
        System.out.println("s = " + s);
    }
}
```

Salida por pantalla del programa anterior:

```
$>javac opAsignacion.java↵

$>java opAsignacion↵
i = 15
j = 15
x = 12.345
c = A
b = false
s = Hola
```

4.2. Operadores aritméticos

El lenguaje de programación Java tiene varios operadores aritméticos para los datos numéricos enteros y reales. En la Tabla 4.2 se resumen los diferentes operadores de esta categoría.

Tabla 4.2 Operadores aritméticos básicos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
`-`	operador unario de cambio de signo	`-4`	`-4`
`+`	Suma	`2.5 + 7.1`	`9.6`
`-`	Resta	`235.6 - 103.5`	`132.1`
`*`	Producto	`1.2 * 1.1`	`1.32`
`/`	División (tanto entera como real)	`0.050 / 0.2` `7 / 2`	`0.25` `3`
`%`	Resto de la división entera	`20 % 7`	`6`

El resultado exacto depende de los tipos de operando involucrados. Es conveniente tener en cuenta las siguientes peculiaridades:

- El resultado es de tipo `long` si, al menos, uno de los operandos es de tipo `long` y ninguno es real (`float` o `double`).
- El resultado es de tipo `int` si ninguno de los operandos es de tipo `long` y tampoco es real (`float` o `double`).
- El resultado es de tipo `double` si, al menos, uno de los operandos es de tipo `double`.
- El resultado es de tipo `float` si, al menos, uno de los operandos es de tipo `float` y ninguno es `double`.
- El formato empleado para la representación de datos enteros es el complemento a dos. En la aritmética entera no se producen nunca desbordamientos (*overflow*) aunque el resultado sobrepase el intervalo de representación (`int` o `long`).

- La división entera se trunca hacia 0. La división o el resto de dividir por cero es una operación válida que genera una *excepción* `ArithmeticException` que puede dar lugar a un error de ejecución y la consiguiente interrupción de la ejecución del programa.
- La aritmética real (en coma flotante) puede desbordar al infinito (demasiado grande, *overflow*) o hacia cero (demasiado pequeño, *underflow*).
- El resultado de una expresión inválida, por ejemplo, dividir infinito por infinito, no genera una excepción ni un error de ejecución: es un valor NaN (*Not a Number*).

En el siguiente programa se emplean todos los operadores aritméticos anteriores:

```
/**
 *  Demostracion de los operadores aritmeticos basicos
 *  A. Garcia-Beltran - marzo, 2008
 */
public class OpAritmeticos {
   public static void main(String[] args) {
      int i,j;
      double a,b;
      i = 7;
      j = 3;
      System.out.println("* Operandos enteros: i = " + i + " ;  j = " + j);
      System.out.println("  Operador suma:     i + j = " + (i+j));
      System.out.println("  Operador resta:    i - j = " + (i-j));
      System.out.println("  Operador producto: i * j = " + (i*j));
      System.out.println("  Operador division: i / j = " + (i/j));
      System.out.println("  Operador resto:    i % j = " + (i%j));
      a = 12.5;
      b = 4.3;
      System.out.println("* Operandos reales:  a = " + a + " ; b = " + b);
      System.out.println("  Operador suma:     a + b = " + (a+b));
      System.out.println("  Operador resta:    a - b = " + (a-b));
      System.out.println("  Operador producto: a * b = " + (a*b));
      System.out.println("  Operador division: a / b = " + (a/b));
      System.out.println("  Operador resto:    a % b = " + (a%b));
   }
}
```

Salida por pantalla del programa anterior:

```
* Operandos enteros:     i = 7 ;  j = 3
  Operador suma:         i + j = 10
  Operador resta:        i - j = 4
  Operador producto:     i * j = 21
  Operador division:     i / j = 2
  Operador resto:        i % j = 1
* Operandos reales:      a = 12.5 ;  b = 4.3
  Operador suma:         a + b = 16.8
  Operador resta:        a - b = 8.2
  Operador producto:     a * b = 53.75
  Operador division:     a / b = 2.906976744186047
  Operador resto:        a % b = 3.9000000000000004
```

4.3. Operadores aritméticos incrementales

Los operadores aritméticos incrementales son operadores unarios (un único operando). El operando puede ser numérico o de tipo `char` y el resultado es del mismo tipo que el operando. Estos operadores pueden emplearse de dos formas dependiendo de su posición con respecto al operando (Tabla 4.3).

Tabla 4.3 Operadores aritméticos incrementales

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

++	Incremento
	i++ primero se utiliza la variable y luego se incrementa su valor
	++i primero se incrementa el valor de la variable y luego se utiliza
		4++
		a=5;
		b=a++;
		a=5;
		b=++a;
			5
			
			a vale 6 y b vale 5
			
			a vale 6 y b vale 6

--	decremento	4--	3

Estos operadores suelen sustituir a veces al operador asignación y también suelen aparecer en bucles `for`.

Ejemplo de programa que emplea operadores incrementales:

```
/**
 *  Demostracion de los operadores incrementales
 *  A. Garcia-Beltran - Abril, 2008
 */
class opIncrementales {
  public static void main(String[] args) {
    int i,j;           //   Variables enteras. Podrian ser reales o char
    i = 7;
    System.out.println("* Operando entero:  i = " + i + ";");
    System.out.println(" Operador ++:       j = i++; ");
    j = i++;
    System.out.println("                    // i vale " + i + "; j vale " + j);
    i = 7;
    System.out.println("                    i = " + i + ";");
    System.out.println("                    j = ++i; ");
    j = ++i;
    System.out.println("                    // i vale " + i + "; j vale " + j);
    i = 7;
    System.out.println("* Operando entero:  i = " + i + ";");
    System.out.println(" Operador --:       j = i--; ");
    j = i--;
    System.out.println("                    // i vale " + i + "; j vale " + j);
    i = 7;
    System.out.println("                    i = " + i + ";");
    System.out.println("                    j = --i; ");
    j = --i;
    System.out.println("                    // i vale " + i + "; j vale " + j);
  }
}
```

Salida por pantalla del programa anterior:

```
* Operando entero:  i = 7;
 Operador ++:       j = i++;
                    // i vale 8; j vale 7
                    i = 7;
                    j = ++i;
                    // i vale 8; j vale 8
* Operando entero:  i = 7;
 Operador --:       j = i--;
                    // i vale 6; j vale 7
                    i = 7;
```

```
j = --i;
// i vale 6; j vale 6
```

4.4. Operadores aritméticos combinados

Combinan un operador aritmético con el operador asignación. Como en el caso de los operadores aritméticos pueden tener operandos numéricos enteros o reales y el tipo específico de resultado numérico dependerá del tipo de éstos. En la Tabla 4.4 se resumen los diferentes operadores de esta categoría.

Tabla 4.4 Operadores aritméticos combinados

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
***=**	Producto combinado	a*=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

Ejemplo de programa que emplea operadores combinados:

```
/**
 *  Demostracion de los operadores aritmeticos combinados
 *  A. Garcia-Beltran - marzo, 2008
 */
public class OpCombinados {
   public static void main(String[] args) {
      int i,j;   // Variables enteras. Podrian ser reales
      i = 7;
      j = 3;
      System.out.println("* Operandos enteros:   i = "+ i +" ;  j = "+ j);
      i += j;
      System.out.println("  Suma combinada:      i += j " + "   // i vale " + i);
      i = 7;
      i -= j;
      System.out.println("  Resta combinada:     i -= j " + "   // i vale " + i);
      i = 7;
      i *= j;
      System.out.println("  Producto combinado:  i *= j " + "   // i vale " + i);
      i = 7;
      i /= j;
      System.out.println("  Division combinada:  i /= j " + "   // i vale " + i);
      i = 7;
      i %= j;
      System.out.println("  Resto combinada:     i %= j " + "   // i vale " + i);
   }
}
```

Salida por pantalla del programa anterior:

```
* Operandos enteros:   i = 7 ;  j = 3
  Suma combinada:      i += j    // i vale 10
```

```
  Resta combinada:     i -= j    // i vale 4
  Producto combinado:  i *= j    // i vale 21
  Division combinada:  i /= j    // i vale 2
  Resto combinada:     i %= j    // i vale 1
```

4.5. Operadores de relación

Realizan comparaciones entre datos compatibles de tipos primitivos (numéricos, carácter y booleanos) teniendo siempre un resultado booleano. Los operandos booleanos sólo pueden emplear los operadores de igualdad y desigualdad. En la Tabla 4.5 se resumen los diferentes operadores de esta categoría

Tabla 4.5 Operadores de relación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	`7 == 38`	`false`
!=	distinto que	`'a' != 'k'`	`true`
<	menor que	`'G' < 'B'`	`false`
>	mayor que	`'b' > 'a'`	`true`
<=	menor o igual que	`7.5 <= 7.38`	`false`
>=	mayor o igual que	`38 >= 7`	`true`

Todos los valores numéricos que se comparan con NaN dan como resultado `false` excepto el operador `!=` que devuelve `true`. Esto ocurre incluso si ambos valores son NaN.

Ejemplo de programa que emplea operadores relacionales:

```
/**
 *  Demostracion de los operadores relacionales
 *  A. Garcia-Beltran - marzo, 2008
 */
public class OpRelacionales {
   public static void main(String[] args) {
     int i,j;
     i = 7;
     j = 3;
     System.out.println("* Operandos enteros:          i = "+ i +" ;  j = "+ j);
     System.out.println("  Operador igualdad:          i == j es " + (i==j));
     System.out.println("  Operador desigualdad:       i != j es " + (i!=j));
     System.out.println("  Operador mayor que:         i > j  es " + (i>j));
     System.out.println("  Operador menor que:         i < j  es " + (i<j));
     System.out.println("  Operador mayor o igual que: i >= j es " + (i>=j));
     System.out.println("  Operador menor o igual que: i <= j es " + (i<=j));
   }
}
```

Salida por pantalla del programa anterior:

```
* Operandos enteros:          i = 7 ;  j = 3
  Operador igualdad:          i == j es false
  Operador desigualdad:       i != j es true
  Operador mayor que:         i > j  es true
  Operador menor que:         i < j  es false
  Operador mayor o igual que: i >= j es true
  Operador menor o igual que: i <= j es false
```

4.6. Operadores lógicos o booleanos

Realizan operaciones sobre datos booleanos y tienen como resultado un valor booleano. En la Tabla 4.6 se resumen los diferentes operadores de esta categoría.

Tabla 4.6 Operadores booleanos

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	`!false` `!(5==5)`	`true` `false`
|	Suma lógica – OR (binario)	`true | false` `(5==5)|(5<4)`	`true` `true`
^	Suma lógica exclusiva – XOR (binario)	`true ^ false` `(5==5)|(5<4)`	`true` `true`
&	Producto lógico – AND (binario)	`true & false` `(5==5)&(5<4)`	`false` `false`
||	Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true`	`true || false` `(5==5)||(5<4)`	`true` `true`
&&	Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false`	`false && true` `(5==5)&&(5<4)`	`false` `false`

Para mejorar el rendimiento de ejecución del código es recomendable emplear en las expresiones booleanas el operador && en lugar del operador &. En este caso es conveniente situar la condición más propensa a ser falsa en el término de la izquierda. Esta técnica puede reducir el tiempo de ejecución del programa. De forma equivalente es preferible emplear el operador || al operador |. En este caso es conveniente colocar la condición más propensa a ser verdadera en el término de la izquierda.

Ejemplo de programa que emplea operadores lógicos:

```
/**
 *  Demostracion de los operadores logicos
 *  A. Garcia-Beltran - marzo, 2008
 */
public class OpBooleanos {
   public static void main(String [] args) {
      System.out.println("Demostracion de operadores logicos");
      System.out.println("Negacion: ! false es       : " + (! false));
      System.out.println("          ! true es       : " + (! true));
      System.out.println("Suma:     false | false es : " + (false | false));
      System.out.println("          false | true es  : " + (false | true));
      System.out.println("          true | false es  : " + (true | false));
      System.out.println("          true | true es   : " + (true | true));
      System.out.println("Producto: false & false es : " + (false & false));
      System.out.println("          false & true es  : " + (false & true));
      System.out.println("          true & false es  : " + (true & false));
      System.out.println("          true & true es   : " + (true & true));
   }
}
```

Salida por pantalla del programa anterior:

```
Demostracion de operadores logicos
Negacion: ! false es       : true
          ! true es        : false
Suma:     false | false es : false
          false | true es  : true
          true | false es  : true
          true | true es   : true
Producto: false & false es : false
          false & true es  : false
          true & false es  : false
          true & true es   : true
```

4.7. El operador condicional

Este operador ternario tomado de C/C++ permite devolver valores en función de una expresión lógica. Sintaxis:

```
expresionLogica ? expresion_1 : expresion_2
```

Si el resultado de evaluar la expresión lógica es verdadero, devuelve el valor de la primera expresión, y en caso contrario, devuelve el valor de la segunda expresión.

Tabla 4.7 Operador condicional

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

?:	operador condicional	`a = 4;`
		`b = a == 4 ? a+5 : 6-a;`
		`b = a > 4 ? a*7 : a+8;`
			b vale 9
			b vale 12

La sentencia de asignación:

```
valor = (expresionLogica ? expresion_1 : expresion_2);
```

como se verá más adelante es equivalente a:

```
if (expresionLogica)
      valor = expresion_1;
else
      valor = expresion_2
```

Ejemplo de programa que emplea el operador condicional:

```
/**
 *  Demostracion del operador condicional
 *  A. Garcia-Beltran - Abril, 2006
 */
public class opCondicional {
   public static void main(String[] args) {
      int i,j,k;
      i = 1;
      j = 2;
      k = i > j ? 2*i : 3*j+1;
      System.out.println("i = " + i);
      System.out.println("j = " + j);
```

```
      System.out.println("k = " + k);
      i = 2;
      j = 1;
      k = i > j ? 2*i : 3*j+1;
      System.out.println("i = " + i);
      System.out.println("j = " + j);
      System.out.println("k = " + k);
   }
}
```

Salida por pantalla del programa anterior:

```
i = 1
j = 2
k = 7
i = 2
j = 1
k = 4
```

4.8. Operadores de bit

Tienen operandos de tipo entero (o `char`) y un resultado de tipo entero. Realizan operaciones con dígitos (ceros y unos) de la representación binaria de los operandos. Exceptuando al operador negación, los demás operadores son binarios. En la Tabla 4.8 se resumen los diferentes operadores de esta categoría.

Tabla 4.8 Operadores de bit

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
~	Negación ó complemento binario (unario)	`~12`	`-13`
|	Suma lógica binaria – OR (binario)	`12|10`	`8`
^	Suma lógica exclusiva – XOR (binario)	`12^10`	`6`
&	Producto lógico binario – AND (binario)	`12&10`	`14`
<<	Desplaza a la izquierda los bits del 1º operando tantas veces como indica el 2º operando (por la derecha siempre entra un cero)	`7<<2` `-7<<2`	`28` `-28`
>>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando (por la izquierda entra siempre el mismo bit más significativo anterior)	`7>>2` `-7>>2`	`1` `-2`
>>>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando – sin signo (por la izquierda entra siempre un cero).	`7>>>2` `-7>>>2`	`1` `1073741822`

Ejemplo de programa que emplea operadores de bit:

```
/**
 *  Demostracion de los operadores de bit enteros
 *  A. Garcia-Beltran - enero, 2003
 */
public class OpBitEnteros2 {
  public static void main(String[] args) {
    int i,j;
    i = 12;
    j = 10;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);
```

```
    System.out.println("  Negacion o complemento:      ~i es " + (~i));
    System.out.println("  Suma logica (binaria):       i & j es " + (i&j));
    System.out.println("  Suma exclusiva (binaria):    i ^ j es " + (i^j));
    System.out.println("  Producto logico (binaria):   i | j es " + (i|j));
    i = 12;
    j = -10;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);
    System.out.println("  Negacion o complemento:      ~i es " + (~i));
    System.out.println("  Suma logica (binaria):       i & j es " + (i&j));
    System.out.println("  Suma exclusiva (binaria):    i ^ j es " + (i^j));
    System.out.println("  Producto logico (binaria):   i | j es " + (i|j));
    i = 7;
    j = 2;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);
    System.out.println("  Despl. a izquierdas:         i << j es " + (i<<j));
    System.out.println("  Despl. a derechas:           i >> j es " + (i>>j));
    System.out.println("  Despl. a derechas sin signo: i >>> j  es " + (i>>>j));

    i = -7;
    j = 2;
    System.out.println("* Operandos enteros:        i = " + i + " ;  j = " + j);
    System.out.println("  Desplazamiento a izquierdas: i << j es " + (i<<j));
    System.out.println("  Despl. a derechas:           i >> j es "+ (i>>j));
    System.out.println("  Despl. a derechas sin signo: i >>> j  es " + (i>>>j));
   }
}
```

Salida por pantalla del programa anterior:

```
* Operandos enteros:           i = 12 ;  j = 10
  Negacion o complemento:      ~i es -13
  Suma logica (binaria):       i & j es 8
  Suma exclusiva (binaria):    i ^ j es 6
  Producto logico (binaria):   i | j es 14
* Operandos enteros:           i = 12 ;  j = -10
  Negacion o complemento:      ~i es -13
  Suma logica (binaria):       i & j es 4
  Suma exclusiva (binaria):    i ^ j es -6
  Producto logico (binaria):   i | j es -2
* Operandos enteros:           i = 7 ;  j = 2
  Despl. a izquierdas:         i << j es 28
  Despl. a derechas:           i >> j es 1
  Despl. a derechas sin signo: i >>> j  es 1
* Operandos enteros:           i = -7 ;  j = 2
  Desplazamiento a izquierdas: i << j es -28
  Despl. a derechas:           i >> j es -2
  Despl. a derechas sin signo: i >>> j  es 1073741822
```

4.9. Operador concatenación de cadenas

El operador concatenación, +, es un operador binario que devuelve una cadena resultado de concatenar las dos cadenas que actúan como operandos. Si sólo uno de los operandos es de tipo cadena, el otro operando se convierte implícitamente en tipo cadena.

Tabla 4.9 Operador concatenación

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+	Operador concatenación	"Hola" + "Juan"	"HolaJuan"

4.10. Separadores

Existen algunos caracteres que tienen un significado especial en el lenguaje Java. En la Tabla 4.10 se resumen los diferentes separadores que pueden encontrarse en el código fuente de un programa.

Tabla 4.10 Separadores en Java

Separador	Descripción
`()`	Permiten modificar la **prioridad de una expresión**, **contener expresiones** para el control de flujo y realizar **conversiones de tipo**. Por otro lado pueden contener la **lista de parámetros** o argumentos, tanto en la definición de un método como en la llamada al mismo.
`{}`	Permiten definir **bloques de código** y ámbitos y contener los valores **iniciales** de las variables `array`
`[]`	Permiten declarar **variables de tipo `array`** (vectores o matrices) y referenciar sus elementos
`;`	Permite **separar sentencias**
`,`	Permite **separar identificadores** consecutivos en la declaración de variables y en las listas de parámetros. También se emplea para encadenar sentencias dentro de un **bucle `for`**
`.`	Permite **separar el nombre de un atributo o método** de su instancia de referencia. También separa el identificador de un **paquete** de los de los subpaquetes y clases

4.11. Expresiones

Una expresión es una combinación de operadores y operandos que se evalúa generándose un único resultado de un tipo determinado.

4.12. Prioridad entre operadores

Si dos operadores se encuentran en la misma expresión, el orden en el que se evalúan puede determinar el valor de la expresión. En la Tabla 4.11 se muestra el orden o prioridad en el que se ejecutan los operadores que se encuentren en la misma sentencia. Los operadores de la misma prioridad se evalúan de izquierda a derecha dentro de la expresión.

Tabla 4.11 Prioridad 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