1. Operadores y expresiones.

Los **operadores** llevan a cabo operaciones sobre un conjunto de datos u operandos, representados por literales y/o identificadores. Los operadores pueden ser *unarios, binarios o terciarios*, según el número de operandos que utilicen sean uno, dos o tres, respectivamente. Los operadores **actúan sobre** los **tipos** de datos **primitivos** y devuelven también un tipo de dato primitivo.
Los operadores se combinan con los literales y/o identificadores para formar expresiones. Una **expresión** es una combinación de operadores y operandos que se evalúa produciendo un único resultado de un tipo determinado.

El resultado de una expresión puede ser usado como parte de otra expresión o en una sentencia o instrucción. Las expresiones, combinadas con algunas palabras reservadas o por sí mismas, forman las llamadas **sentencias** o **instrucciones**.
Por ejemplo, pensemos en la siguiente expresión Java:

```
i + 1
```

Con esta expresión estamos utilizando un operador aritmético para sumarle una cantidad a la variable i, pero es necesario indicar al programa qué hacer con el resultado de dicha expresión:

```
suma = i + 1;
```

Que lo almacene en una variable llamada `suma`, por ejemplo. En este caso ya tendríamos una **acción** completa, es decir, una **sentencia o instrucción**.
Más ejemplos de sentencias o instrucciones los tenemos en las declaraciones de variables, vistas en apartados anteriores, o en las estructuras básicas del lenguaje como sentencias condicionales o bucles, que veremos en unidades posteriores.
Como curiosidad comentar que las expresiones de asignación, al poder ser usadas como parte de otras asignaciones u operaciones, son consideradas tanto expresiones en sí mismas como sentencias.

1.1. Operadores aritméticos.

Los operadores aritméticos son aquellos operados que combinados con los operandos forman expresiones matemáticas o aritméticas.

Operadores aritméticos básicos			
Operador	**Operación Java**	**Expresión Java**	**Resultado**
-	Operador unario de cambio de signo	-10	-10
+	Adición	1.2 + 9.3	10.5
-	Sustracción	312.5 – 12.3	300.2
*	Multiplicación	1.7 * 1.2	1.02
/	División (entera o real)	0.5 / 0.2	2.5
%	Resto de la división entera	25 % 3	1

El resultado de este tipo de expresiones depende de los operandos que utilicen:

Resultados de las operaciones aritméticas en Java	
Tipo de los operandos	**Resultado**
Un operando de tipo `long` y ninguno real (`float` o `double`)	`long`
Ningún operando de tipo `long` ni real (`float` o `double`)	`int`
Al menos un operando de tipo `double`	`double`
Al menos un operando de tipo `float` y ninguno `double`	`float`

Otro tipo de operadores aritméticos son los operadores unarios incrementales y decrementales. Producen un resultado del mismo tipo que el operando, y podemos utilizarlos con **notación prefija**, si el operador aparece antes que el operando, o **notación postfija**, si el operador aparece después del operando. En la tabla puedes ver un ejemplo de de utilización de cada operador incremental.

- **notación prefija** : primero actúa el operador y después el resto de operaciones.
- **notación postfija** : primero actúan el resto de operaciones y después el operador.

Operadores incrementales en Java		
Tipo operador	**Expresión Java**	

Operadores incrementales en Java		

++ (incremental)	Prefija:
	x=3;
	y=++x;
	// x vale 4 e y vale 4
		Postfija:
		x=3;
		y=x++;
		// x vale 4 e y vale 3

-- (decremental)	Prefija:
	x=3;
	y=--x;
	// x vale 2 e y vale 2
		Postfija:
		x=3;<
		
		> y=x--;
		// x vale 2 e y vale 3

Ejemplo:

```
public class operadoresaritmeticos {
    public static void main(String[] args) {
        short x = 7;
        int y = 5;
        float f1 = 13.5f;
        float f2 = 8f;
        System.out.println("El valor de x es " + x + ", y es " + y);
        System.out.println("El resultado de x + y es " + (x + y));
        System.out.println("El resultado de x - y es " + (x - y));
        System.out.printf("%s\n%s%d\n","División entera:","x / y = ",(x/y));
        System.out.println("Resto de la división entera: x % y = " + (x % y));
        System.out.println("El valor de f1 es " + f1 + ", f2 es " + f2);
        System.out.println("El resultado de f1 / f2 es " + (f1 / f2));
    }
}
```

En el ejemplo vemos un programa básico que utiliza operadores aritméticos. Observa que usamos `System.out.printf` para mostrar por pantalla un texto formateado. El texto entre dobles comillas son los argumentos del método `printf` y si usamos más de uno, se separan con comas. Primero indicamos cómo queremos que salga el texto, y después el texto que queremos mostrar. Fíjate que con el primer `%s` nos estamos refiriendo a una variable de tipo `String`, o sea, a la primera cadena de texto, con el siguiente `%s` a la segunda cadena y con el último `%s` a la tercena. Con `%f` nos referimos a un argumento de tipo float, etc.

2.C.1 Linea por linea

1.2. Operadores de asignación.

El principal operador de esta categoría es el operador asignación `"="`, que permite al programa darle un valor a una variable, y ya hemos utilizado varias ocasiones en esta unidad. Además de este operador, Java proporciona otros operadores de asignación combinados con los operadores aritméticos, que permiten abreviar o reducir ciertas expresiones.

Por ejemplo, el operador `"+="` suma el valor de la expresión a la derecha del operador con el valor de la variable que hay a la izquierda del operador, y almacena el resultado en dicha variable. En la siguiente tabla se muestran todos los operadores de asignación compuestos que podemos utilizar en Java

Operadores de asignación combinados en Java		
Operador	**Ejemplo en Java**	**Expresión equivalente**
`+=`	`op1 += op2`	`op1 = op1 + op2`
`-=`	`op1 -= op2`	`op1 = op1 - op2`
`*=`	`op1 *= op2`	`op1 = op1 * op2`
`/=`	`op1 /= op2`	`op1 = op1 / op2`
`%=`	`op1 %= op2`	`op1 = op1 % op2`

```
public class operadoresasignacion {
    // clase principal main que inicia la aplicación
    public static void main(String[] args) {
        int x;
        int y;
        x = 5;  // operador asignación
        y = 3;  // operador asignación
        //operadores de asignación combinados
        System.out.printf("El valor de x es %d y el valor de y es %d\n", x,y);
        x += y;
        // podemos utilizar indistintamente printf o println
        System.out.println(" Suma combinada: x += y " + " ..... x vale " + x);
        x = 5;
        x -= y;
        System.out.println(" Resta combinada: x -= y " + " ..... x vale " + x);
```

```
        x = 5;
        x *= y;
        System.out.println(" Producto combinado: x *= y " + " ........ x vale " + x);
        x = 5;
        x /= y;
        System.out.println(" Division combinada: x /= y " + " ........ x vale " + x);
        x = 5;
        x %= y;
        System.out.println(" Resto combinada: x %= y " + " ........... x vale " + x);
    } // fin main
} // fin operadoresasignacion
```

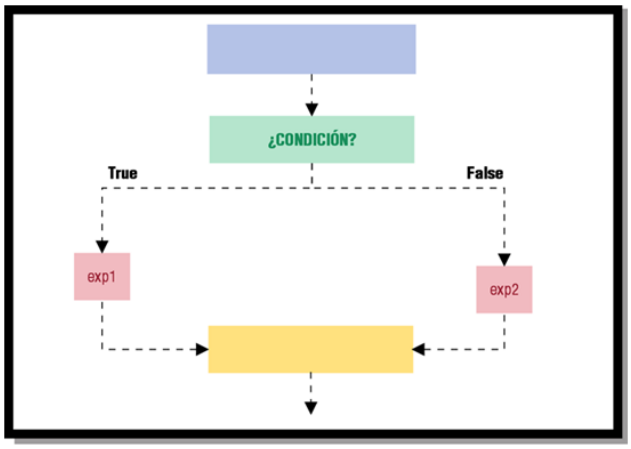
Para saber más

En el siguiente enlace tienes información interesante sobre cómo se pueden utilizar los caracteres especiales incluidos en la orden `printf` (en inglés):

Orden `printf`

1.3. Operador condicional.

El operador condicional "`?:`" sirve para evaluar una condición y devolver un resultado en función de si es verdadera o falsa dicha condición. Es el único operador ternario de Java, y como tal, necesita tres operandos para formar una expresión.
El primer operando se sitúa a la izquierda del símbolo de interrogación, y siempre será una expresión booleana, también llamada condición. El siguiente operando se sitúa a la derecha del símbolo de interrogación y antes de los dos puntos, y es el valor que devolverá el operador condicional si la condición es verdadera. El último operando, que aparece después de los dos puntos, es la expresión cuyo resultado se devolverá si la condición evaluada es falsa.



Operador condicional en Java	
Operador	Expresión en Java
?:	condición ? exp1 : exp2

Por ejemplo, en la expresión:

```
(x>y)?x:y;
```

Se evalúa la condición de si `x` es mayor que y, en caso afirmativo se devuelve el valor de la variable `x`, y en caso contrario se devuelve el valor de `y`.
El operador condicional se puede sustituir por la sentencia `if…then…else` que veremos en la unidad de Estructuras de control.

1.4. Operadores de relación.

Los operadores relacionales se utilizan para comparar datos de tipo primitivo (numérico, carácter y booleano). El resultado se utilizará en otras expresiones o sentencias, que ejecutarán una acción u otra en función de si se cumple o no la relación.
Estas expresiones en Java dan siempre como resultado un valor booleano true o false. En la tabla siguiente aparecen los operadores relacionales en Java.

Operadores relacionales en Java		
Operador	**Ejemplo en Java**	**Significado**
`==`	`op1 == op2`	op1 igual a op2
`!=`	`op1 != op2`	op1 distinto de op2
`>`	`op1 > op2`	op1 mayor que op2

Operadores relacionales en Java		
`<`	`op1 < op2`	op1 menor que op2
`>=`	`op1 >= op2`	op1 mayor o igual que op2
`<=`	`op1 <= op2`	op1 menor o igual que op2

Hasta ahora hemos visto ejemplos que creaban variables y se inicializaban con algún valor. Pero ¿y si lo que queremos es que el usuario introduzca un valor al programa? Entonces debemos agregarle interactividad a nuestro programa, por ejemplo utilizando la clase `Scanner`. Aunque no hemos visto todavía qué son las clases y los objetos, de momento vamos a pensar que la clase `Scanner` nos va a permitir leer los datos que se escriben por teclado, y que para usarla es necesario importar el paquete de clases que la contiene. El código del ejemplo lo tienes a continuación. El programa se quedará esperando a que el usuario escriba algo en el teclado y pulse la tecla intro. En ese momento se convierte lo leído a un valor del tipo `int` y lo guarda en la variable indicada. Además de los operadores relacionales, en este ejemplo utilizamos también el operador condicional, que compara si los números son iguales. Si lo son, devuelve la cadena iguales y si no, la cadena distintos.

```
import java.util.Scanner;
//importamos el paquete necesario para poder usar la clase Scanner
public class ejemplorelacionales {
   // método principal que inicia la aplicación
   public static void main( String args[] )
   {
      // clase Scanner para petición de datos
      Scanner teclado = new Scanner( System.in );
      int x, y;
      String cadena;
      boolean resultado;
      System.outprint( "Introducir primer número: " );
      x = teclado.nextInt(); // pedimos el primer número al usuario
      System.outprint( "Introducir segundo número: " );
      y = teclado.nextInt(); // pedimos el segundo número al usuario
      // realizamos las comparaciones
      cadena=(x==y)?"iguales":"distintos";
      System.outprintf("Los números %d y %d son %s\n",x,y,cadena);
      resultado=(x!=y);
      System.outprintln("x != y  // es " + resultado);
      resultado=(x < y );
      System.outprintln("x < y  // es " + resultado);
      resultado=(x > y );
      System.outprintln("x > y  // es " + resultado);
      resultado=(x <= y );
      System.outprintln("x <= y  // es " + resultado);
      resultado=(x >= y );
      System.outprintln("x >= y  // es " + resultado);
    } // fin método main
} // fin clase ejemplorelacionales
```

1.5. Operadores lógicos.

Los operadores lógicos realizan operaciones sobre valores booleanos, o resultados de expresiones relacionales, dando como resultado un valor booleano. Los operadores lógicos los podemos ver en la tabla que se muestra a continuación. Existen ciertos casos en los que el segundo operando de una expresión lógica no se evalúa para ahorrar tiempo de ejecución, porque con el primero ya es suficiente para saber cuál va a ser el resultado de la expresión.

Por ejemplo, en la expresión `a && b` si `a` es falso, no se sigue comprobando la expresión, puesto que ya se sabe que la condición de que ambos sean verdadero no se va a cumplir. En estos casos es más conveniente colocar el operando más propenso a ser falso en el lado de la izquierda. Igual ocurre con el operador `||`, en cuyo caso es más favorable colocar el operando más propenso a ser verdadero en el lado izquierdo.

Operadores lógicos en Java		
Operador	**Ejemplo en Java**	**Significado**
!	!op	Devuelve true si el operando es false y viceversa.
&	op1 & op2	Devuelve true si op1 y op2 son true
|	op1 | op2	Devuelve true si op1 u op2 son true
^	op1 ^ op2	Devuelve true si sólo uno de los operandos es true
&&	op1 && op2	Igual que &, pero si op1 es false ya no se evalúa op2
|	op1 | op2	Igual que |, pero si op1 es true ya no se evalúa op2

En el siguiente código puedes ver un ejemplo de utilización de operadores lógicos:

```
public class operadoreslogicos {
    public static void main(String[] args) {
```

```
        // **TODO** code application logic here
        System.out.println("OPERADORES LÓGICOS");

        System.out.println("Negacion:\n ! false es : " + (! false));
        System.out.println(" ! true es : " + (! true));

        System.out.println("Operador AND (&):\n 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("Operador OR (|):\n 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("Operador OR Exclusivo (^):\n 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("Operador &&:\n 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("Operador ||:\n 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));
    } // fin main
} // fin operadoreslogicos
```

1.6. Operadores de bits.

Los operadores a nivel de bits se caracterizan porque realizan operaciones sobre números enteros (o `char`) en su representación binaria, es decir, sobre cada dígito binario.
En la tabla tienes los operadores a nivel de bits que utiliza Java.

Operadores a nivel de bits en Java		
Operador	**Ejemplo en Java**	**Significado**
`~`	~op	Realiza el complemento binario de op (invierte el valor de cada bit)
`&`	op1 & op2	Realiza la operación AND binaria sobre op1 y op2
|	op1 | op2	Realiza la operación OR binaria sobre op1 y op2
`^`	op1 ^ op2	Realiza la operación OR-exclusivo (XOR) binaria sobre op1 y op2
`<<`	op1 << op2	Desplaza op2 veces hacia la izquierda los bits de op1
`>>`	op1 >> op2	Desplaza op2 veces hacia la derecha los bits de op1
`>>>`	op1 >>> op2	Desplaza op2 (en positivo) veces hacia la derecha los bits de op1

Para saber más
Los operadores de bits raramente los vas a utilizar en tus aplicaciones de gestión. No obstante, si sientes curiosidad sobre su funcionamiento, puedes ver el siguiente enlace dedicado a este tipo de operadores:
Operadores de bits

1.7. Precedencia de operadores.

El orden de precedencia de los operadores determina la secuencia en que deben realizarse las operaciones cuando en una expresión intervienen operadores de distinto tipo.

Las reglas de precedencia de operadores que utiliza Java coinciden con las reglas de las expresiones del álgebra convencional. Por ejemplo:

- La multiplicación, división y resto de una operación se evalúan primero. Si dentro de la misma expresión tengo varias operaciones de este tipo, empezaré evaluándolas de izquierda a derecha.
- La suma y la resta se aplican después que las anteriores. De la misma forma, si dentro de la misma expresión tengo varias sumas y restas empezaré evaluándolas de izquierda a derecha.

A la hora de evaluar una expresión es necesario tener en cuenta la **asociatividad** de los operadores. La asociatividad indica qué operador se evalúa antes, en condiciones de igualdad de precedencia. Los operadores de asignación, el operador condicional (`?:`), los operadores incrementales (`++, --`) y el casting son asociativos por la derecha. El resto de operadores son asociativos por la izquierda, es decir, que se empiezan a calcular en el mismo orden en el que están escritos: de izquierda a derecha. Por ejemplo, en la expresión siguiente:

```
10 / 2 * 5
```

Realmente la operación que se realiza es `(10 / 2 ) * 5`, porque ambos operadores, división y multiplicación, tienen igual precedencia y por tanto se evalúa primero el que antes nos encontramos por la izquierda, que es la división. El resultado de la expresión es `25`. Si fueran asociativos por la derecha, puedes comprobar que el resultado sería diferente, primero multiplicaríamos `2 * 5` y luego dividiríamos entre `10`, por lo que el resultado sería 1. En esta otra expresión:

```
x = y = z = 1
```

Realmente la operación que se realiza es `x = (y = (z = 1))`. Primero asignamos el valor de `1` a la variable `z`, luego a la variable `y`, para terminar asignando el resultado de esta última asignación a `x`. Si el operador asignación fuera asociativo por la izquierda esta operación no se podría realizar, ya que intentaríamos asignar a `x` el valor de `y`, pero `y` aún no habría sido inicializada.

En la tabla se detalla el orden de precedencia y la asociatividad de los operadores que hemos comentado en este apartado. Los operadores se muestran de mayor a menor precedencia.

Orden de precedencia de operadores en Java		
Operador	Tipo	Asociatividad
`++ --`	Unario, notación postfija	Derecha
`++ -- + -` `(cast) ! ~`	Unario, notación prefija	Derecha
`* / %`	Aritméticos	Izquierda
`+ -`	Aritméticos	Izquierda
`<< >> >>>`	Bits	Izquierda
`< <= > >=`	Relacionales	Izquierda
`== !=`	Relacionales	Izquierda
`&`	Lógico, Bits	Izquierda
`^`	Lógico, Bits	Izquierda
|	Lógico, Bits	Izquierda
`&&`	Lógico	Izquierda
||	Lógico	Izquierda
`?:`	Operador condicional	Derecha
`= += -= *=` `/= %=`	Asignación	Derecha

Reflexiona

¿Crees que es una buena práctica de programación utilizar paréntesis en expresiones aritméticas complejas, aún cuando no sean necesarios?