1. Caso práctico.

Unidad Didáctica III

Caso práctico

La programación suele resultar apasionante para los que no la conocen, eso de hacer programas y que el resto de la gente los utilice como herramienta imprescindible de trabajo porque son útiles, es algo que desde fuera se ve como una labor para elegidos. Pero para los que nos dedicamos a programar ordenadores, es sin duda mucho más que eso, se trata de construir algo siguiendo mecanismos y métodos preestablecidos a los que siempre se les añade algo de creatividad y aportaciones personales del programador. Sin duda es una labor satisfactoria, aunque en ocasiones muy compleja, ya que el programador de una aplicación, debe estudiar y conocer a la perfección el problema al que se enfrenta en cada nuevo trabajo.

Carmen *es sin duda una programadora que disfruta con su trabajo y cada vez se siente más segura como parte del equipo de programación de* ***Soluciones Informáticas Andalucía, S.C.A*** *(en adelante* ***SI Andalucía****). Le gustaba la informática y al hacer el ciclo formativo de Desarrollo de Aplicaciones Informáticas ha descubierto que su verdadera vocación es la programación. Conoce varios lenguajes pero actualmente está programando en Java y aunque al principio le resultó algo difícil, ahora se siente cómoda y le consta que están contentos con su forma de trabajar.*

Operadores, expresiones e instrucciones

2. Introducción

Unidad Didáctica III

Introducción

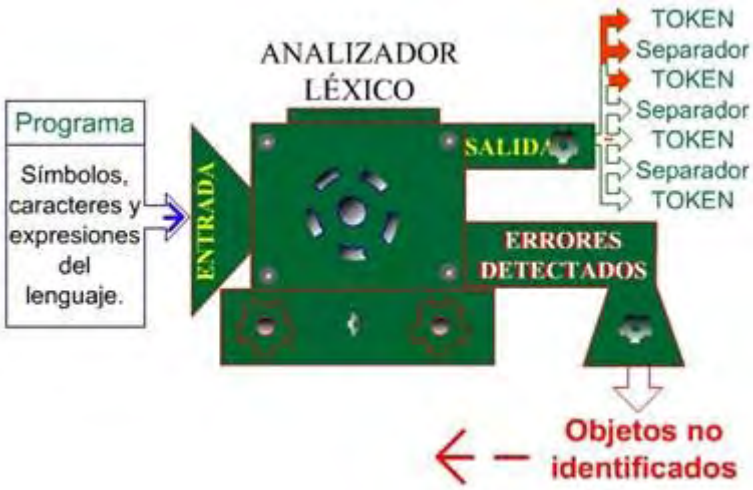
Una noche que sale con unos amigos, en un bar del Puerto de Aguadulce conoce a un chico que se interesa por su trabajo.
Durante la conversación ella le explica cuál es su labor actual en la empresa y que una de las cosas que más le gusta es que antes de la compilación de un programa es preciso hacer un estudio previo y conocer el problema exactamente.
Su amigo se sorprende mucho y dice que "compilación" es una palabra que jamás había oído antes. ***Carmen*** *le explica que simplemente consiste en que las instrucciones deben pasar al ordenador de forma que lo entienda sin ningún tipo de ambigüedad, lo que requiere una identificación de cada uno de los tokens que se le pasan para posteriormente traducirlo todo a código máquina.*
Su amigo le dice que perdone pero que él es de Filología (Lengua) y eso de "token" le suena a divinidad primitiva. Entonces ***Carmen*** *le explica que es un término anglosajón. El ordenador tiene que hacer un análisis léxico y los tokens son los grupos de símbolos que reconoce del lenguaje en cuestión, se trata de nombres, caracteres, operadores, separadores, etc.*
Algo que siguiendo una determinada estructura del lenguaje de programación va a permitir suministrar conjuntos de instrucciones a la máquina para que realice determinada tarea. Su amigo el filólogo le comenta que algo así ocurre cuando leemos cualquier cosa, primero identificamos los símbolos (sintaxis) y luego al agruparlos adquieren un significado (semántica).

En las unidades anteriores mencionábamos que para poder ejecutar un programa era necesario traducirlo a código máquina, y que una de las formas más habituales de hacerlo era mediante el uso de compiladores.

¿Te has planteado cómo consigue el compilador realizar ese proceso?

Parece evidente que tendrá que analizar el texto buscando e identificando elementos que tengan un significado especial. Uno de estos elementos serán las **instrucciones**, pero de la misma forma que cuando hablamos usamos frases u oraciones, que se componen de sintagmas, las instrucciones de un lenguaje de programación contienen otros elementos, que llamamos **expresiones**. Y al igual que el predicado de una oración necesita casi siempre de un verbo, las expresiones se suelen construir usando operadores. El proceso viene a ser el siguiente, de forma aproximada:

- **Cuando el compilador empieza a analizar el texto del programa escrito en lenguaje de alto nivel, lo que recibe es una secuencia de caracteres pertenecientes al alfabeto de ese lenguaje** (Unicode, en el caso de Java, ASCII, en otros lenguajes)
- **La primera tarea que debe realizar es aislar los elementos que tienen significado por sí mismos**, lo que usando la analogía con el lenguaje natural serían las palabras que forman ese texto (ese programa, en nuestro caso). A esa fase se le llama **análisis léxico**.
- **Para esto se sirve de unos elementos especiales que denominamos separadores.** Un **separador** no es más que un carácter, un símbolo o una palabra escrita en el texto del programa que le indica al compilador dónde termina una palabra con significado propio para el lenguaje y dónde empieza la siguiente. Cada una de las "palabras" que reconoce el compilador recibe en programación el nombre de **token**.



Simulación de lo que puede ser un analizador léxico.

Operadores, expresiones e instrucciones

2.1. Tipos de separadores

Unidad Didáctica III

Tipos de separadores

¿Cuáles serán esos elementos que se usan como separadores?¿Se parecerán a los que usamos en el lenguajes escrito para separar unas palabras de otras o para terminar una frase? Justamente es lo que se ha hecho en los lenguajes de programación, imitar una vez más al lenguaje natural.

En la mayoría de los lenguajes, los elementos que se consideran como separadores son los siguientes:

- **Espacio en blanco**. El espacio en blanco es un carácter más del alfabeto del lenguaje. Es el separador típico, y al igual que hacemos en el castellano escrito usándolo para separar palabras distintas, en programación se usa para separar tokens distintos.
- **Return**. El carácter de cambio de línea, o enter, o intro o new line, o return (todos esos nombres recibe) se usa con el mismo significado que el espacio en blanco. La única diferencia es la visual a la hora de leer el texto escrito. No obstante, podríamos escribir un programa entero en una sola línea (sin usar return), o con cada token en una línea distinta (sin usar espacios en blanco) o con cualquier mezcla de ambas situaciones, y el resultado para el compilador sería exactamente el mismo, ya que lo que hace realmente es sustituir cualquier separador o cualquier secuencia de separadores consecutivos por un único espacio en blanco.
- **Tabulador**. Al igual que return, se trata de otro posible separador equivalente al espacio en blanco.
- **Comentario**. Un comentario cumple asímismo las funciones de separador. Los comentarios tienen la función de documentación del código fuente, pero desde el punto de vista léxico del lenguaje, no son más que separadores. No representan ninguna acción a realizar, y por tanto se pueden suprimir al realizar la traducción a código máquina. Los lenguajes disponen de algún modo de indicar que parte del texto es un comentario, normalmente encerrándolo entre algunos caracteres especiales. Por ejemplo, en Java hay varios modos:

```
/* comentario */
/**
   comentario
    */
// comentario hasta fin de línea
```

En los 2 primeros ejemplos, la palabra comentario no tiene por qué estar compuesta de una sola línea.

El segundo tipo de comentario recibe el nombre de comentario javadoc. Son comentarios especiales, ya que el lenguaje Java es capaz de generar a partir de ellos documentación sobre la aplicación en formato html, formato web, que directamente se pueda publicar en Internet. No obstante, para el compilador, son comentarios similares a los del primer tipo, que se ignoran.

AUTOEVALUACIÓN

Víctor está preocupado porque tiene que escribir una expresión muy larga, que no es cómoda de leer en la pantalla ya que se sale de los márgenes, pero si la parte en dos líneas, lo mismo no la entiende el compilador, o la interpreta como dos instrucciones distintas.

La expresión en cuestión es la contenida en el código siguiente. Los números son válidos (literales de tipo double válidos), los métodos son correctos y las expresiones no tienen ningún problema de tipos incompatibles. De hecho, Víctor sabe que esta expresión, escrita en una línea funcionará. La única duda que tiene Víctor es si el hecho de que esté escrita en 3 líneas será correcto o no.

```
b=(37895432145454654d +Math.sqrt(Math.abs(c)
)-4564546678444554454d )/ Double.parseDouble
("45698887565.454");
```

¿Piensas que el compilador la entenderá correctamente?:

- a) Sí, porque todos los separadores son iguales para el compilador, y los cambios de línea y las tabulaciones son separadores.
- b) No, porque el primer paréntesis de la expresión marca un bloque que debe estar en una sola línea para que el compilador pueda entenderlo como bloque.
- c) No, porque la última línea, al comenzar con un nombre de clase será siempre interpretada como otra sentencia distinta
- d) Sí, porque la expresión no se considera terminada hasta que se encuentra una llave de cierre.

Comprobar

Para saber más:
Si quieres profundizar un poco más acerca del funcionamiento de los compiladores y cómo realizan la transformación del código fuente para obtener el código objeto, visita el siguiente enlace:

Compiladores [Versión en Caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)

Para descargar el programa Acrobat Reader pulsa aquí.

AUTOEVALUACIÓN

Señala la afirmación correcta:

- a) La fase en la que el compilador aísla los elementos que tienen significado por sí mismos se denomina análisis léxico.
- b) Un separador no es más que un carácter, un símbolo o una palabra escrita en el texto del programa y que le indica al compilador donde termina una palabra y donde empieza la siguiente
- c) Cada una de las "palabras" que reconoce el compilador recibe en programación el nombre de token.
- d) Todas las anteriores son correctas.

Comprobar

3. Tipos de Tokens.

Unidad Didáctica III

Tipos de Tokens

El filólogo le comenta que si hace un paralelismo con el idioma podría distinguir diferentes tokens como signos de puntuación, sustantivos, adjetivos, números, operadores aritméticos, símbolos matemáticos, etc. ***Carmen*** *asiente aunque le dice que un lenguaje de programación es algo mucho más simple, sólo contiene cuatro o cinco tipos de tokens, precisamente para facilitar el análisis sintáctico.*

Piensa en la analogía entre el lenguaje de programación y el lenguaje natural, el que hablamos todos los días. A fin de cuentas ambos son lenguajes, y tienen ciertas similitudes. ¿Son todas las palabras iguales a la hora de formar una frase con sentido, bien construida? ¿Es indiferente el orden de las palabras? La respuesta no hace falta decir que es NO. No interpretamos igual los verbos que los artículos, ni podemos usarlos en el orden que queramos.

Lo mismo ocurre en los lenguajes de programación. En el proceso de análisis léxico, se han detectado los tokens que forman el programa. Pero existen distintos tipos de tokens, y en los siguientes apartados debemos aprender a identificarlos.

Operadores, expresiones e instrucciones

3.1. Palabras reservadas.

Unidad Didáctica III

Palabras reservadas

Son aquellos tokens que tienen asignada una función específica en el lenguaje y que los programadores no pueden utilizar más que para lo que el lenguaje establece.

Por ejemplo, en Java, la palabra **int** sólo puede usarse para declarar una variable de ese tipo entero. Gran parte del aprendizaje de un lenguaje consiste en conocer el significado y la función de las palabras reservadas de ese lenguaje. La lista de palabras reservadas de un lenguaje, en contra de lo que pueda parecer, no es excesivamente larga. En el caso de Java, está formada por menos de 50 palabras:

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

Operadores, expresiones e instrucciones

3.2. Literales

Unidad Didáctica III

Literales

Son valores concretos para un tipo de datos básico del lenguaje.

Por ejemplo, 3 es un literal para el tipo **int** en Java, y 3.0 es un literal para el tipo **double**, 'a' es un literal para el tipo **char**, y "Hola a todos" es un literal de tipo **String** (cadena de caracteres).

Operadores, expresiones e instrucciones

3.3. Identificadores.

Unidad Didáctica III

Identificadores

Todos los elementos que yo defina en mi programa (variables, constantes, clases, métodos o procedimientos, etc.) deberé nombrarlos para poder referirme a ellos. A ese nombre, que justamente me permite identificar y referirme a cada uno de esos elementos de forma inequívoca, es a lo que llamamos identificador.

Los identificadores son pues, los nombres que el programador decide asignar a los elementos que crea en su programa.

Por **ejemplo**, yo decido si una variable de tipo entero se va a llamar numero, o contador, o *n*, o xz347w_j. Como programador decido libremente el nombre que le voy a dar, y ése es su identificador. No obstante, conviene decir que aparte de las limitaciones que imponga cada lenguaje (habrá símbolos que no se podrán usar, obligará o no a empezar con caracteres no numéricos, limitará o no el tamaño del identificador, etc.) es conveniente usar nombres lo más descriptivos posible, para que sirvan por sí mismos de aclaración al funcionamiento del programa. Al mismo tiempo, evitaremos en la medida de lo posible usar nombres excesivamente largos, que harían tediosa la tarea de escribirlos a lo largo del programa. Por ejemplo, si la variable entera la uso para contar el número de alumnos de una clase, según se van matriculando, lo ideal es que se llame numeroDeAlumnos, o nAlumnos, o algo similar. xz347w_j no parecería un nombre adecuado ya que no es nada descriptivo. Tampoco lo sería numeroDeAlumnosMatriculadosEnElCurso, por que aunque es muy descriptivo, es innecesariamente largo y engorroso de escribir. Por cierto, que los identificadores nunca admiten espacios en blanco. Si queremos usar varias palabras, deberemos ponerlas juntas, o separadas por el signo de subrayado.

El convenio que se sigue en Java y en otros lenguajes es:

Escribir cada una de las palabras que forman el identificador con sólo la primera letra en mayúscula.

- **Todas las demás letras de cada palabra irán en minúsculas.**
- **La primera letra de la primera palabra será también minúscula (tal y como aparece en los ejemplos usados)**
- **Si se trata del nombre de una clase, la primera letra de la primera palabra del nombre empezará por mayúscula también**
- **Las constantes de clase (algo que se verá qué significa más adelante) son la excepción a esta regla en Java. Sus nombres se escriben con todas las letras mayúsculas, y las distintas palabras separadas por el signo de subrayado, también llamado guión bajo. MAXIMO_DE_ALUMNOS_MATRICULABLES podría ser un ejemplo.**

Fíjate bien en que decimos que es un convenio. **No es en absoluto obligatorio cumplir estas reglas para que los programas funcionen perfectamente**, **pero resulta muy útil respetarlas**. Permiten distinguir a qué tipo de elemento nos estamos refiriendo sin más que ver su nombre. No es necesario ver donde está definido para tener esa información.

Operadores, expresiones e instrucciones

3.4. Operadores.

Unidad Didáctica III

Operadores

Son tokens especiales, porque actúan de separadores, manteniéndose a sí mismos como tokens, con significado propio.

Expresan operaciones a realizar con los datos o **expresiones** que les acompañan. Al igual que en el lenguaje natural existen elementos que permiten unir unas palabras con otras para formar unidades mayores con significado (sujeto, predicado, complementos, etc.) los operadores permiten unir literales e identificadores (y también expresiones ya formadas) para formar expresiones más complejas, más elaboradas, y con mayor significado.

Ejemplo: en la siguiente expresión Java, que declara una variable llamada numero de tipo int y le asigna el valor 38, podemos encontrar los cuatro tipos de tokens

```
int numero = 38
```

int	**Es una** palabra reservada**, que se usa para declarar variables de ese tipo entero.**
Numero	**Es un** identificador**. El nombre elegido por el programador para la variable.**
=	**Es el** operador **de asignación. Asigna a la variable de la izquierda el resultado de evaluar la expresión de la derecha. En este caso, la expresión de la derecha es directamente el valor a asignar.**
38	**Es un** literal **del tipo int. Un valor concreto de entre los muchos que incluye el rango de los números enteros de tipo int.**

Para saber más:
Un sitio interesante donde completar tus conocimientos sobre los fundamentos de Java, especialmente tipos de tokens entre otras cosas es:
Tokens de Java **[Versión en Caché]**

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)

Operadores, expresiones e instrucciones

4. Expresiones

Unidad Didáctica III

Expresiones

*A veces cuando están programando en la oficina de **SI Andalucía**, reciben la visita de **Antonio**, hermano de **José**, un apasionado de la informática, que le gusta pasar la tarde en la empresa interesándose por todo lo que hacen. **Antonio** es más de manipulación del hardware, no le va la programación pero se ha hecho muy amigo de **Víctor** y se interesa por todo lo que hace.*

***Víctor** le explica que tiene que dar las instrucciones al ordenador mediante expresiones en las que combina identificadores, operadores y literales, junto a algunas palabras reservadas del lenguaje. Es algo así como hacer frases con una estructura tal que las entienda la máquina, manteniendo cierto orden en la ubicación de cada uno de los elementos que forman la expresión.*

Hemos visto en el apartado anterior cómo consigue el compilador aislar las distintas palabras (los tokens), y hemos visto que pude haber distintos tipos de tokens. ¿Cómo unir esas palabras para formar algo que tenga sentido para el ordenador? ¿Cual es la siguiente unidad mayor en el camino hasta formar órdenes válidas para el ordenador?

Los literales, junto con los identificadores y los operadores se pueden combinar para formar unidades de mayor significado, que denominamos expresiones. De esta forma la expresión podrá evaluarse, (evaluar una expresión consiste en calcularla, sustituir cada elemento por su valor, y hacer las operaciones indicadas) dando como resultado un valor determinado. La expresión tendrá asociado un tipo, y ese tipo será el del valor resultante de su evaluación.

Para entender el concepto de **expresión**, el símil con el lenguaje natural vendría a ser algo así como que uniendo un nombre a un artículo y a un adjetivo, en un determinado orden, estaríamos formando un sintagma nominal, que podría ser el sujeto de una oración. Por ejemplo: *El perro verde y el gato negro*. Hemos unido palabras de distinto tipo (aquí el operador es la conjunción "y" y el propio orden de escritura) para formar algo más elaborado, con más significado, que puede ser el sujeto de una oración, pero que por sí solo no tiene sentido completo (aún no es una oración).

Descomposición de una expresión en tokens.

En programación podríamos pensar en la expresión Java:

```
n  + 1
```

Por sí misma aún no indica nada, salvo una operación que genera un resultado con el que no sabemos qué hacer, pero tiene más sentido que cada una de las partes por separado. Sin embargo si le decimos que el resultado lo guarde en una variable llamada suma, ya si tenemos una acción completa, una sentencia o instrucción.

```
suma = n  + 1;
```

Las expresiones, combinadas con algunas palabras reservadas y a veces por sí mismas, forman sentencias o instrucciones.

Además de las expresiones, en las que de alguna forma se lleva a cabo la manipulación aritmética, lógica y relacional de la información, son necesarios elementos adicionales del lenguaje que controlen el orden de ejecución de las sentencias o instrucciones.

Operadores, expresiones e instrucciones

4.1. Concepto de expresión.

Unidad Didáctica III

Concepto de expresión

***Víctor** le explica además que el ordenador entiende también que una expresión puede estar formada por expresiones. **Antonio** entonces hace un gesto de estar perdido. Pero **Víctor** le explica que a él le pasaba lo mismo pero que **José** le dijo que sólo hay que ser ordenado y pensar como lo hace el ordenador es casi más sencillo que pensar como lo hace un pez.*

En el apartado anterior empezamos a intuir el concepto de expresión. A veces esa intuición es mejor que la más completa definición formal. Pero a veces también es necesaria esa definición. ¿Piensas que con el ejemplo anterior ya tienes claro qué es una expresión? ¿Sabrías identificar como tal cualquier expresión?

Seguramente necesitas leer la definición del concepto de expresión. Es una definición **recurrente** o **recursiva** ,que son dos formas de decir lo mismo.

Vamos a introducir esa definición a través de un ejemplo simple:

En el programa de nóminas que está realizando Víctor, es necesario sumar el sueldo base con los complementos de cada trabajador para calcular su sueldo bruto. Tanto **sueldoBase** como **complementos** son dos variables de tipo double. Además, para calcular el sueldo total hay que descontarle las retenciones para el IRPF, que son del 15**(porciento)**, por lo que ese total habrá que multiplicarlopor 0.85, ya que si se descuenta el 15% sólo se le pagará el 85**(porciento)**. La expresión resultante para hacer el cálculo es:

```
double sueldoTotal = (sueldoBase + complementos)* 0.85
```

1. **Todo literal de un determinado tipo es también una expresión de ese mismo tipo**.

0.85 es por sí mismo una expresión de tipo double, porque es un literal de tipo double.

2. **Todo identificador de un determinado tipo (una variable o constante) es también una expresión de ese mismo tipo.**

sueldoTotal es por sí misma una expresión de tipo double, por ser un identificador de una variable de ese tipo.

3. **Dado un operador *n*-ario que requiere *n* operandos (expresiones) de los tipos t_1, t_2, ..., t_n, y dados *n* operandos de los tipos correspondientes, el resultado es una expresión del tipo generado por el operador.**

El operador suma (**+**) para los números reales de tipo double es binario, ya que requiere dos sumandos de tipo double, cada uno de los cuales podría ser un literal double o a su vez una expresión de tipo double. Permite formar otra expresión double, ya que el resultado del operador suma de reales (double) es a su vez un número real (double). **(sueldoBase + complementos)** es por tanto una nueva expresión de tipo double.

Por el mismo motivo, al usar el operador binario (*****) , el resultado vuelve a ser una expresión double, porque el producto de números reales da como resultado un número real (double)

Por tanto en la expresión anterior, usamos el operador ***** con una expresión de tipo double a su izquierda, **(sueldoBase + complementos)** y con un literal de tipo double a su derecha, **0.85,** que

también es una expresión double. El resultado es la expresión double **(sueldoBase + complementos)* 0.85**

A continuación, todo el valor de esa expresión, que será un double, lo asignamos a la variable sueldoTotal, que también es de tipo double. El operador de asignación forma una expresión del tipo que se asigna, y el valor que devuelve la asignación es el propio valor asignado. De esta forma, **sueldoTotal = (sueldoBase + complementos)* 0.85** es toda ella a su vez una expresión de tipo double.

Si posteriormente escribimos

sueldoTotal < 1500 obtenemos una expresión de tipo lógico, ya que **<** es un operador de comparación que devuelve verdadero o falso. En este caso, cuando evaluemos la primera expresión de tipo entero (**sueldoTotal**) y la comparemos (**<**) con la segunda expresión de tipo entero (**1500**) obtendremos un valor verdadero si realmente el sueldo total es menor que 1500 y falso si es mayor o igual a 1500.

La noción de expresión es bastante potente debido a la recursividad de la definición.

AUTOEVALUACIÓN

Señala la afirmación correcta:

- a) Todo literal de un determinado tipo es también una expresión de cualquier tipo.
- b) Todo identificador de un determinado tipo (una variable o constante) es también una expresión de ese mismo tipo
- c) Todo operador de un determinado tipo es también una expresión de ese mismo tipo
- d) Todas las anteriores son correctas.

Comprobar

Para saber más:
Si después de este apartado quieres ampliar contenidos, uno de los mejores sitios para comprender bien la recursividad puedes encontrarlo en el siguiente enlace. Pero no olvides que más adelante tendrás que estudiar toda una unidad dedicada a la recursividad, que es una potente herramienta para resolver algunos tipos de problemas.
Iniciarse en la Recursividad [Versión en Caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)

Operadores, expresiones e instrucciones

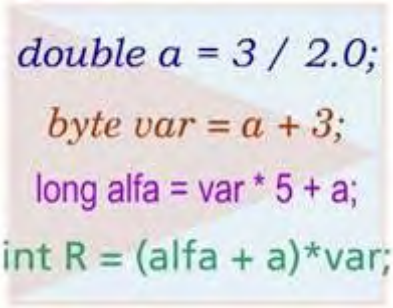
5. Expresiones matemáticas o aritméticas.

Unidad Didáctica III

Expresiones matemáticas o aritméticas

*En el desarrollo de la aplicación de gestión de nóminas para **Servicios Locales S.L.**, **José** se está encontrando ante la tarea de sintetizar como expresiones matemáticas una serie de fórmulas para la elaboración de la nómina de un trabajador. **José** ya tiene cierta experiencia y a pesar de ello es una labor difícil y laboriosa. Su mejor baza es la buena planificación del trabajo que ha realizado antes de comenzar con la programación, lo que le ha permitido preparar todos los casos y facilitar el trabajo de **Carmen** y **Víctor**.*

Acabamos de ver la definición de expresión, y hemos visto que estaba ligada al concepto de tipo: todas las expresiones son de un tipo, están asociadas a un tipo de datos. ¿Quiere decir eso que hay tantos tipos de expresiones como de datos? Justamente es eso lo que ocurre. ¿A qué tipos de datos estarán entonces asociadas las expresiones matemáticas?



Las expresiones matemáticas o aritméticas se forman mediante el uso de los operadores matemáticos o aritméticos asociados a los tipos numéricos básicos, junto a operadores de asignación. Generarán como resultado un valor numérico de alguno de los tipos definidos en el lenguaje.

Es posible que en una expresión matemática participen literales, identificadores (variables o constantes) o subexpresiones de distintos tipos numéricos, si bien es cierto que para operar entre ellos, habrá que hacer ciertas conversiones de tipo (también denominadas **casting**) para hacer posibles las operaciones. Esas conversiones de tipo pueden ser implícitas (las realiza por su cuenta el compilador) o explícitas (debe obligar a realizarlas el programador indicándolo expresamente).

Por ejemplo:

```
double a = 3 / 2.0;
```

- El operador / es la división, tanto entera como real.
- Si alguno de los dos operandos es de tipo real, será la división real la que se efectúe, es decir, obteniendo decimales.
- El compilador pasa por su cuenta el literal entero de tipo int 3 a 3.0, que es un literal real de tipo double, y realiza la operación real, obteniendo como resultado un double.
- **Eso es un casting o conversión explícita, también llamada promoción automática de tipos.**

Otro ejemplo:

```
byte var = 100;
var = (byte) (var + var);
```

- El tipo byte no tiene operadores propios sino que usa los del tipo int.
- Por tanto para hacer la suma (var + var) debe convertir el valor de var a int de forma implícita, para poder operar.
- Pero si quiero guardar el resultado de la operación, que es de tipo int y se almacena usando 32 bits, deberé forzar la conversión a byte (almacenado en 8 bits).
- En Java eso se consigue indicando entre paréntesis, y delante de la expresión, el tipo al que hay que convertir el resultado de la misma.
- **Eso es una conversión explícita (o casting explícito).**

5.1. Operadores matemáticos o aritméticos.

Unidad Didáctica III

Operadores matemáticos o aritméticos

Las expresiones matemáticas deben dar como resultado de ser evaluadas, un valor numérico. ¿Qué tipos de operadores nos permitirán asociar distintos literales y expresiones para obtener como resultado ese valor numérico? Parece evidente que serán los **operadores matemáticos**.

El conjunto de operadores aritméticos que nos proporcionan la mayoría de los lenguajes para sus tipos numéricos suele ser bastante similar. Es por ello que directamente vamos a indicar sus nombres, junto a los símbolos que se emplean en el caso concreto del lenguaje Java. Son los operadores que nos permiten formar la mayoría de las expresiones aritméticas, aunque no todas, como veremos en el apartado dedicado a las funciones en esta misma unidad.

Suma	**+**	**Operador binario (necesita dos operandos)**
Número positivo o signo	**+**	**Operador monario o unario (necesita un operando)**
Resta	**-**	**Operador binario**
Número negativo o signo	**-**	**Operador monario o unario**
Multiplicación	*****	**Operador binario**
División	**/**	**Operador binario**
Resto-Módulo	**%**	**Operador binario**

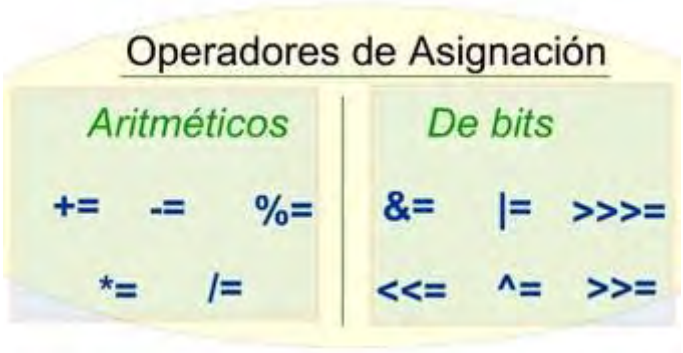
Operadores, expresiones e instrucciones

5.2. Operador u operadores de asignación.

Unidad Didáctica III

Operador u operadores de asignación

¿Cual crees que será la operación más frecuente en la mayoría de los lenguajes? Todos los programas usan variables, y siempre es obligatorio darles un valor antes de poder usarlas. ¿Qué operador nos permite asignarle valores a las variables y constantes? El operador de **asignación**.



Es el operador más usado sin duda en la mayoría de los programas, y el primero que tendremos necesidad de usar cuando aprendemos a programar. La función que realiza es justamente la que indica su nombre.

Permite asignar un valor (que puede ser un literal o el resultado de evaluar una expresión) a un identificador (constante o variable).

Todos los lenguajes deben tener al menos un operador de asignación básico, aunque desde la aparición del lenguaje C, casi todos los lenguajes disponen de todo un conjunto de operadores combinados de asignación que se obtienen uniendo el operador de asignación básico con otros operadores.

No todos los lenguajes usan el mismo símbolo para este operador. En concreto, suele usarse el signo igual, y es el que hemos venido viendo que se usa en Java.

Así, la **sentencia** **a = 3**; indica en Java que a la variable **a** hay que asignarle el valor **3**, es decir, que en la posición de memoria asociada a la variable **a** se escriba ese valor, con independencia del valor que ya tuviera. Si tenía otro valor, será machacado (anulado) por el nuevo valor que se asigna. Por ejemplo en otro lenguaje, Pascal, la asignación se hace con el operador := para distinguirlo del operador de comparación de valores, que es = Sin embargo en Java es a este último al que se le da un símbolo distinto, = = (dos iguales consecutivos comparan en Java)

```
Un operador combinado de asignación será de la forma:
variable OP = expresión

donde OP es un operador binario. Se analiza como:

variable = variable OP expresión
```

Siguiendo este esquema, los operadores de asignación permitidos en Java, además del básico, se obtienen usando operadores aritméticos y de bits junto con el operador =. La lista completa de operadores es:

Combinando con operadores matemáticos:

+=	-=	*=	/=	%=

Combinando con operadores de bits:

&=	|=	^=	<<=	>>=	>>>=

Este último grupo rara vez se usará en programas de gestión, que son los que nos ocupan en este módulo profesional.

Operadores, expresiones e instrucciones

5.3. Operadores de incremento y decremento.

Unidad Didáctica III

Operadores de incremento y decremento

¿Crees que al escribir programas será frecuente la operación de sumar uno o restar uno a una variable?

Realmente sí son operaciones muy frecuentes, tanto como para que merezca la pena permitir una escritura abreviada de esas operaciones. Cada vez que necesitemos contar el número de veces que ocurre algo, por ejemplo, tendremos que sumar uno cada vez que suceda ese algo. O restarle uno si lo que llevamos es la cuenta de las veces que todavía debe suceder.

También desde su aparición en el lenguaje C casi todos los lenguajes incorporan el operador de incremento y el de decremento. Por supuesto, también Java.

- **El operador de incremento es ++ y se encarga de sumarle uno a la variable que acompañe**.

La sentencia **a++;** equivale a totalmente a la sentencia **a = a + 1;**

- **El operador de decremento es -- y se encarga de restarle uno a la variable que acompañe.**

La sentencia **a--;** equivale a totalmente a la sentencia **a = a - 1;**

Ambos operadores pueden usarse en **notación prefija o postfija**. La diferencia es apreciable sólo cuando intervienen en una sentencia de asignación, ya que:

- **En notación prefija (operador delante de la variable), primero se hace el incremento o decremento y luego la asignación.**
- **En notación postfija (operador detrás de la variable) primero se hará la asignación con el valor que tuviera la variable, y luego se realiza el incremento**.

Veámoslo con un ejemplo:

María está haciendo una aplicación de contabilidad, en la que hay que anotar la fecha de amortización de cada elemento productivo. Esa fecha se obtiene sumándole el periodo de amortización del componente en cuestión al año actual en el momento de comprarlo. Para el caso concreto de los ordenadores, el periodo de amortización es de 1 año, por lo que María ha pensado que las sentencias siguientes solucionan su problema .

```
int añoActual = 2005;
int añoAmortizacion = añoActual++;
```

Sin embargo, la segunda sentencia equivale a:

```
añoAmortizacion = añoActual;
añoActual = añoActual +1;
```

(En ese orden) por lo que **añoAmortizacion** valdrá **2005**, que es el año en el que se ha comprado, y no en el que se amortiza.

¿Qué debería haber hecho?

```
int añoActual = 2005;
int añoAmortizacion = ++añoActual;
```

Ahora la segunda sentencia equivale a :

```
añoActual = añoActual +1;
añoAmortizacion = añoActual;
```

(En ese orden) por lo que **añoAmortizacion** valdrá **2006**, que es el valor correcto.

AUTOEVALUACIÓN

Dados a =9 y b=10 señale la afirmación correcta:

- a) int c = a++; c valdrá 10.
- b) int c = a++; c valdrá 11.
- c) int c = a++; c valdrá 9.
- d) Ninguna de las anteriores es correcta

Comprobar

Dados a =9 y b=10 señale la afirmación correcta:

- a) int c = ++b; c valdrá 11.
- b) int c = ++b; c valdrá 10.
- c) int c = ++b; c valdrá 9.
- d) Todas las respuestas anteriores son correctas.

Comprobar

Operadores, expresiones e instrucciones

6. Expresiones lógicas o booleanas.

Unidad Didáctica III

Expresiones lógicas o booleanas

En una aplicación del tipo que tiene ocupados ahora a nuestros amigos, es inevitable tomar muchas decisiones debido a las grandes diferencias en la nómina de los trabajadores; diferentes niveles salariales, retenciones, dietas, ausencias, retrasos, pagas extras, etc. Por ello se hace necesario hacer comparaciones y tomar diferentes caminos de actuación según las particularidades de cada trabajador, de su trabajo y del momento en que se realice la nómina. Este tipo de situaciones se tratan con comparaciones y evaluación de determinados valores que permiten decidir el tratamiento a seguir en cada caso. ***Carmen*** *le explica a* ***Víctor*** *que hay que variar los cálculos a aplicar si el trabajador es casado o tiene hijos, según el puesto que ocupe en la empresa y también influye si la nómina es la de Junio o Diciembre, en las que tiene paga extra.*

Piensa en la cantidad de situaciones en las que en un programa deberá comprobar determinadas condiciones y en función de que se cumplan o no deberás realizar unas operaciones u otras. Pues bien, todas esas condiciones serán expresiones lógicas, que en cada momento serán ciertas o falsas. Se llaman expresiones lógicas.

Las expresiones lógicas se obtienen por el uso de operadores de tipo lógico y relacional, dando como resultado un valor lógico o booleano.

Recuerda que en la unidad 2 vimos que los datos booleanos son aquellos que sólo pueden tomar dos valores, concretamente verdadero o falso (true o false en Java, que es como se escribe en inglés).

Es decir, el resultado de **evaluar una expresión** de tipo lógico será obligatoriamente un valor de tipo lógico, que sólo puede tomar los valores verdadero o falso (true o false respectivamente, para Java).

Operadores, expresiones e instrucciones

6.1. Operadores relacionales.

Unidad Didáctica III

Operadores relacionales

En el apartado anterior hemos comentado la necesidad de comprobar condiciones. Esas condiciones, en muchos casos serán **comparaciones de igualdad, desigualdad o de orden** (ver si un elemento es mayor que otro).

Por ejemplo, si quiero calcular la raíz cuadrada de un número, lo primero que debo hacer es comprobar que el número en cuestión sea positivo (mayor o igual que cero) Si lo es, calcularé su raíz cuadrada, y si no lo es, indicaré que no es posible calcular la raíz cuadrada de números negativos.

Los operadores relacionales nos permiten formar expresiones lógicas por comparación de los valores usados como operandos. El resultado de estos operadores es un valor lógico (true o false, en Java). Para que puedan ser comparables, los valores deben corresponder a un tipo ordenado, comparable, tales como los números.

El conjunto clásico de operadores relacionales se expone a continuación, junto a su forma concreta en el caso del lenguaje Java (que es la más común, por otra parte, aunque no la única posible):

>	Mayor que
>=	Mayor o igual que
<	Menor que
< =	Menor o igual que
= =	Igual
!=	Distinto (no igual)

Y como estos operadores funcionan exactamente como cabe esperar, no les dedicaremos más atención por ahora, aunque los utilizaremos con mucha frecuencia cuando comencemos a hacer programas con Java.

En Java, para otros tipos no primitivos, tales como objetos, existen métodos que permiten hacer las comparaciones, siempre que exista algún orden definido entre esos objetos. Alguno de esos métodos son equals() , compareTo(),... Se estudiarán más adelante, ya que es necesario el aprendizaje previo de algunas características del lenguaje.

Operadores, expresiones e instrucciones

6.2. Operadores lógicos o condicionales.

Unidad Didáctica III

Operadores lógicos o condicionales

Al hablar, sin darte cuenta usas constantemente operadores lógicos. Por ejemplo, una frase que perfectamente podrías decir: "**Para aprobar la asignatura necesito estudiar mucho y hacer bien todas las prácticas o hacer bien el examen final**".

A --> estudiar mucho

B --> hacer bien todas las prácticas

C --> hacer bien el examen final

La condición que debe ser cierta es: "**(A Y B) O C"**.

No basta estudiar mucho. Además debes hacer bien todas las prácticas para aprobar. Pero sí bastaría con hacer bien el examen final, aunque no hubieras estudiado mucho ni hubieras hecho bien las prácticas. Ese significado es el que se deriva de la conjunción Y y la disyunción O.

Los operadores lógicos o condicionales nos permiten formar expresiones lógicas directamente a partir de los valores de otras expresiones lógicas.

Los tres operadores lógicos básicos son : negación (u operador **Not**), **conjunción** (o Y-lógico u operador **And**) y **disyunción** (u O-lógico u operador **Or**). Su forma en Java es la que se indica a continuación, aunque otros lenguajes suelen usar directamente sus nombres en inglés

!	**Negación (not)**
&&	**Conjunción (and)**
||	**Disyunción (or)**

Aunque estos operadores funcionan tal y como cabría esperar,

- la **negación** obtiene el valor de verdad opuesto al de la expresión a la que se aplica,
- la **conjunción** es cierta si los dos argumentos lo son, y
- la **disyunción** es cierta si al menos alguno de los argumentos lo es), el comportamiento de conjunción y disyunción en el caso del lenguaje Java presenta una peculiaridad especial, que se explica en el siguiente apartado.

Operadores, expresiones e instrucciones

6.3. Evaluación de expresiones lógicas.

Unidad Didáctica III

Evaluación de expresiones lógicas

En la aplicación de nóminas que está realizando SI Andalucía, hay que calcular el valor de un complemento salarial, que va ligado al puesto que se ocupa en la empresa, y a la categoría profesional del trabajador.

Existen tres puestos de trabajo: Directivo, Administrativo y Operario.

Existen tres categorías profesionales: Eventuales, En prácticas, Fijos.

Carmen tiene que comprobar la situación de cada trabajador para calcular el sueldo adecuadamente. Concretamente sabe que los eventuales cobran el complemento básico de su categoría, los que están en prácticas, 1,5 veces el complemento básico y los fijos el doble del complemento básico.

Para comprobar la situación de un trabajador Carmen debe hacer condiciones que comprobarán los valores tanto del puesto de trabajo como de la categoría profesional, usando conjunción para averiguar el complemento a asignar. Un ejemplo puede ser:

```
puestoTrabajo=="Operario" && categoriaProfesional=="Eventual"
```

En este caso, aparentemente será necesario evaluar tanto la condición **puestoTrabajo=="Operario"** como la condición **categoriaProfesional=="Eventual"**, y aplicar el operador condicional "conjunción" a los valores obtenidos de esa evaluación. El resultado será cierto si las dos condiciones lo son.

No obstante, nos podemos preguntar **qué ocurre si tras evaluar la condición puestoTrabajo=="Operario" se obtiene que ésta es falsa.En este caso, el resultado será falso independientemente del valor de la condición categoriaProfesional=="Eventual"**.

Pues bien**, Java aplica esa estrategia, por lo quees posible que en una condición formada por operadores lógicos, no todos los componentes de la expresión se lleguen a evaluar**. **En una conjunción, si la primera parte es falsa, la segunda parte no se evalúa.** Es importante tenerlo en cuenta, porque veremos a lo largo del curso casos en los que el orden en que se escribe la conjunción es significativo. (aunque aparentemente tanto da decir A y B como decir B y A, debido a esta característica de Java, no siempre es indiferente el orden usado)

En el caso de la disyunción se produce un fenómeno similar. Carmen sabe que si el puesto es Directivo, no importa la categoría profesional, y se cobrará siempre lo máximo, el doble del complemento básico. La expresión que decide usar en la comprobación antes de asignar el complemento es:

```
puestoTrabajo=="Directivo" || categoriaProfesional=="Fijo"
```

SipuestoTrabajo=="Directivo" es cierta, el resultado de la expresión completa será verdadero independientemente del valor decategoriaProfesional=="Fijo", por lo que esta segunda expresión sólo se evaluará cuando **puestoTrabajo=="Directivo"** sea falsa.

Operadores, expresiones e instrucciones

7. Expresiones de bits.

Unidad Didáctica III

Expresiones de bits

*Durante la programación de la aplicación de gestión de nóminas de la empresa **Servicios Locales**, **Víctor** le pregunta a **José**, qué sentido tiene la existencia de instrucciones a nivel de bits cuando eso ya no es necesario, ya que el compilador traduce siempre a binario. **José** le explica que en una aplicación de gestión como la que están realizando ahora, no se utilizan pero que en una ocasión él si los empleó para realizar directamente operaciones sobre los registros del procesador, al programar uno de los puertos del ordenador para un brazo mecánico en la empresa en la que hizo las prácticas.*

Algunos lenguajes de alto nivel, como C o Java, permiten realizar también operaciones que pueden considerarse como de bajo nivel, tales como manipular directamente los valores almacenados en los **registros del microprocesador**, o realizar operaciones directamente en binario, Estas operaciones resultan muy útiles al programar dispositivos electrónicos, por ejemplo, o al hacer algoritmos eficientes que realicen algunas operaciones, como la división de forma muy rápida.

Por eso incluyen una serie de operadores que permiten trabajar directamente con datos expresados en binario, operando a nivel de **bit**. Esto suele realizarse para mejorar la eficiencia de aplicaciones en las que el tiempo de respuesta es un factor crítico, y en algunas otras situaciones concretas. Sin embargo se trata de operaciones muy poco frecuentes en el ámbito de la gestión, que son el tipo de aplicaciones a las que se orienta este módulo profesional y este ciclo formativo. Por eso no entraremos en muchos detalles.

Las expresiones de bits, serán por tanto las que trabajan directamente con bits, usando operadores de bits.

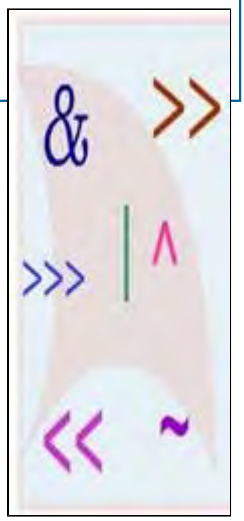
Operadores, expresiones e instrucciones

7.1. Operadores de bits.

Unidad Didáctica III

Operadores de bits

Por los motivos mencionados en el apartado anterior, no vamos a explicar el funcionamiento de los operadores de bits, ni vamos a entrar en más detalles sobre su utilidad.



Sólo se presenta la lista de los operadores de bits que incluye el lenguaje Java, y un enlace al final del apartado para quien quiera consultar más información acerca de su funcionamiento. A fin de cuentas, también forman parte del lenguaje.

~	Negación (complemento de bits)
&	Conjunción binaria
|	Disyunción binaria (inclusiva)
^	Disyunción binaria (exclusiva)
<<	Desplazamiento a la izquierda, rellenando con ceros los bits que quedan libres a la derecha.
>>	Desplazamiento a la derecha, rellenando con el bit mayor (de signo) los bits que quedan libres a la izquierda.
>>>	Desplazamiento a la derecha, rellenando con ceros los bits que quedan libres a la izquierda.

Para saber más:
Operadores de Bits en C++ y Java [Versión en Caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)

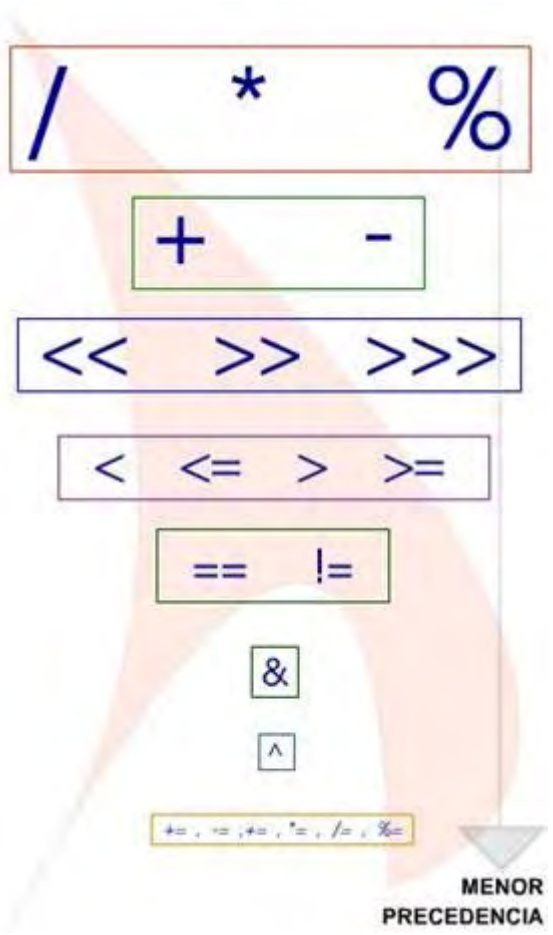
Operadores, expresiones e instrucciones

8. Orden de precedencia de los operadores matemáticos y lógicos.

Unidad Didáctica III

Orden de precedencia de los operadores matemáticos y lógicos

José *le explica a* ***Víctor*** *que en aquella ocasión en la que programó los movimientos del brazo mecánico tuvo algunos problemas al no utilizar correctamente los paréntesis en las expresiones lógicas y el brazo se levantaba cuando le pedía coger. Detectó el problema cuando uno de sus compañeros le recomendó que repasara las condiciones en que debía coger un objeto. Efectivamente al no poner un paréntesis cambiaba completamente el sentido de la expresión.*



Piensa en lo que has aprendido en matemáticas. Cuando escribes una operación matemática en la que intervienen sumas, restas, multiplicaciones, divisiones, potencias, etc. no es necesario especificar con paréntesis el orden en que se van a realizar todas las operaciones. Sabes que:

- las potencias se hacen antes que los productos, y
- los productos de hacen antes que las sumas.

En los lenguajes de programación ocurre lo mismo.

Cuando en una expresión intervienen varios operadores de distintos tipos, hay que dejar claro cuál va a ser el orden en el que se van a ejecutar. Esto es siempre posible mediante el uso de paréntesis, pero abusar de los paréntesis complica las expresiones hasta hacerlas a veces ininteligibles.

Por eso, al igual que se hace en el lenguaje matemático habitual, **se establecen reglas de precedencia o prioridad de los operadores que permiten reducir considerablemente el uso de paréntesis.**

Por ejemplo, ante una expresión como la siguiente,

```
a + b * c
```

normalmente esperamos :

- Que en primer lugar se realice la multiplicación b * c
- Que al resultado de esa multiplicación se le sume a.
- Que en definitiva todo se haga como aprendimos en matemáticas, donde la multiplicación tiene más fuerza o prioridad que la suma.

En la mayoría de los casos, sobre todo para las expresiones aritméticas, basta con conocer el orden de precedencia que se usa en matemáticas, pero el gran número de operadores que existen en un lenguaje de programación hace aconsejable imponer más reglas de precedencia.

Lo ideal sería recordarlas todas, pero tampoco es estrictamente necesario. Basta con recordar las más lógicas u obvias. Ante cualquier duda, debemos pensar que cualquier otro programador que vea nuestro programa también podría tener la misma duda, y todo será más claro y fácil si usamos paréntesis.

La siguiente tabla muestra todos los operadores existentes en Java de mayor a menor precedencia. La lista no es exhaustiva, debido a que en Java existen más operadores de los que han aparecido hasta el momento, pero sí es bastante completa. De hecho, aparece algún operador, como **new**, de los que aparecen en la lista que aún no ha sido visto y que no es ni matemático ni lógico, pero que será usado frecuentemente con posterioridad.

Los operadores que poseen la misma precedencia se encuentran en el mismo grupo separados por comas.

Operadores postfijos	**expr++, expr--**
Operadores prefijos y unarios	**++expr , --expr , +expr , -expr , ~ , !**
Creación y cast	**new , (tipo)expr**
Multiplicativos	*** , / , %**
Aditivos	**+ , -**
Desplazamiento (bits)	**<< , >> , >>>**
Relacionales	**< , <= ,> , >=**
Igualdad	**= = , !=**
Conjunción bits (AND)	**&**
Disy. excl. bits (XOR)	**^**
Disy. incl. bits (OR)	**|**
Conjuncion lógica	**&&**
Disyunción lógica	**||**
Condicional	**? :**
Asignación	**+= , -= ,+= , *= , /= , %= , &= , |= , ^= , <<= , >>= , >>>=**

Por ejemplo, de acuerdo con estas reglas, la expresión

```
b * b - 4 * a * c < 0
```

realmente se analiza como:

```
((b * b) - (4 * a * c)) < 0
```

- En primer lugar se ejecutan las multiplicaciones que son las que más precedencia tienen del conjunto de operadores de la expresión.
- A continuación se ejecuta la resta.
- El resultado de la resta se compara con 0, ya que el operador relacional posee menor precedencia que el - (que es de tipo aditivo).

Operadores, expresiones e instrucciones

9. Regla de asociatividad de los operadores matemáticos y lógicos.

Unidad Didáctica III

Regla de asociatividad de los operadores matemáticos y lógicos

José *le explica a* ***Víctor*** *que desde aquel episodio del brazo robotizado, no ha vuelto a cometer el error de no usar paréntesis en las expresiones, ya que nunca entorpecen la ejecución de la expresión y siempre resultan aclaratorios y evitan dobles sentidos en las instrucciones.*

El orden de precedencia soluciona muchos de los casos, pero no todos. ¿Qué ocurre cuando aparecen seguidos dos o más operadores de la misma precedencia? Consideremos por ejemplo, la expresión:

```
8 / 4 / 2
```

Es ambigua, ya que se pueden obtener las dos interpretaciones siguientes:

- (8 / 4) / 2 = 1
- 8 / (4 / 2) = 4

Éstos son los problemas que resuelve la idea de **asociatividad** en un operador. Evidentemente, esta propiedad sólo se aplica a los operadores binarios (con dos argumentos).

La regla de asociatividad es bastante simple: todos los operadores son asociativos por la izquierda, es decir, a igual nivel de precedencia y a falta de paréntesis, se realizarán en el mismo orden que están escritos, excepto la asignación, que es asociativa por la derecha.

Así, la expresión

```
((b * b) - (4 * a * c)) < 0
```

se analiza en realidad como

```
((b * b) - ((4 * a) * c)) < 0
```

Mientras que una expresión como:

```
a = b = c = 1
```

al ser la asignación asociativa por la derecha, se evaluará como:

```
a = (b = (c = 1))
```

Es decir, en primer lugar se asigna 1 a la variable c. **El resultado completo del operador de**

asignación es el mismo valor asignado (en este caso 1), el cual se asigna a continuación a b, y posteriormente a la variable a.

Si la asociatividad fuera también por la izquierda para la asignación, la expresión a = b = c = 1 no sería evaluable, a pesar de ser claro lo que pretende hacer, ya que se intentaría asignar a la variable a el valor de b, pero b todavía no tendría ningún valor asignado.

AUTOEVALUACIÓN

Señale la afirmación correcta:

- a) Todos los operadores son asociativos por la izquierda, es decir, a igual nivel de precedencia y a falta de paréntesis, se realizarán en el mismo orden que están escritos, excepto la asignación, que es asociativa por la derecha
- b) Resumiendo, podemos afirmar que en general todos los operadores son asociativos por la derecha.
- c) Todos los operadores son asociativos por la derecha, excepto la asignación, que es asociativa por la izquierda
- d) Todos los operadores son asociativos por la derecha, es decir, a igual nivel de precedencia y a falta de paréntesis, se realizarán en el mismo orden que están escritos, excepto la asignación, que es asociativa por la izquierda.

Comprobar

Operadores, expresiones e instrucciones

10. Funciones.

Unidad Didáctica III

Funciones

*Una de los recursos más utilizados por **José** a la hora de programar, son las funciones. Siempre ha dicho que para qué programar algo que ya está hecho y probado sobradamente. **Víctor** recuerda que le dice que un buen programador debe conocer la mayoría de las funciones que el compilador le proporciona y utilizarlas de forma correcta.*

El lenguaje nos proporciona una amplia variedad de operadores, pero la imaginación humana es infinita, y las necesidades de operar de un programa impredecibles, o en cualquier caso mucho mayores que la lista de operadores básicos del lenguaje. ¿Piensas que el lenguaje tendrá más posibilidades que las que nos ofrecen esos operadores básicos? ¿Qué hacer si queremos realizar algún tipo de operación que no está definida por el lenguaje o que nadie había necesitado antes?

La mejor solución parece ser crear un mecanismo flexible para que se puedan proporcionar una serie de operaciones que se puedan usar además de los operadores de los tipos básicos, o incluso permitir la definición de nuevos operadores por el programador que se adapten a sus necesidades.

Además de los operadores vistos hasta ahora para los distintos tipos, la mayoría de los lenguajes definen toda una serie de funciones que permiten realizar múltiples operaciones sobre los datos, obteniendo datos de los más diversos tipos.

Por ejemplo, es normal que los lenguajes incorporen toda una batería de funciones matemáticas, tales como raíces, valor absoluto, redondeo, funciones trigonométricas, logaritmos, potenciación, etc. que son utilizables con datos de tipo numérico y que pueden formar parte de cualquier expresión matemática, ya que la llamada desde el programa a estas funciones se sustituye por el valor que devuelven (el valor que calculan).

Por ejemplo, en Java, todas esas funciones matemáticas se encuentran disponibles en una clase llamada Math, y se pueden formar expresiones matemáticas como la que sigue, en la que se le asigna ala variable **a** el valor absoluto de -3, o sea 3:

```
int a = Math.abs( -3);
```

Pero el concepto de función en un lenguaje es mucho más general que el de función matemática, y realmente se extiende a cualquier procedimiento o método que devuelve un dato, de cualquier tipo.

Así, por ejemplo, un método o procedimiento que devuelve una subcadena a partir de una cadena, puede considerarse como una función.

En Java, ese método se llama **substring().**

Un ejemplo de su uso lo encontramos cuando Carmen, en la aplicación de nóminas, necesita hacer un método de búsqueda para encontrar todos los trabajadores que tengan un determinado nombre, con independencia de sus apellidos, y mostrarlos en un listado. Es necesario porque los jefes, al consultar los datos, no siempre recuerdan los apellidos de todos los trabajadores. Para ellos es más

fácil buscarlo en una lista más reducida de trabajadores que tienen ese nombre, en vez de buscar en toda la lista de trabajadores de la empresa.

Carmen ha definido una variable nombreTrabajador, que en un momento dado recibe el valor que figura a continuación:

```
String nombreCompletoTrabajador;
nombreCompletoTrabajador = "Narciso Jáimez Toro";
```

Necesita extraer sólo el nombre de esa cadena, y sabe que el nombre es la palabra formada por todos los caracteres que hay desde el principio de la cadena nombreTrabajador hasta la aparición del primer espacio en blanco.

La primera letra está en la posición 0 de la cadena, pero ¿en qué posición está el primer blanco? Java también nos proporciona el método **indexOf(),** que puede considerarse también como una función, y que calcula esa posición:

```
int posicionPrimerBlanco;
posicionPrimerBlanco =  nombreCompletoTrabajador.indexOf(' ');
```

En este caso, el método **indexOf() nos** busca la posición del primer blanco, y devuelve el valor 7, ya que es en la posición 7 donde está el octavo carácter de la cadena, que es un espacio en blanco. Ese valor (7) se asigna a la variable **posicionPrimerBlanco.**

Ahora podemos usar el método (o función) **substring()** para decirle que obtenga la subcadena comprendida entre el primer carácter y el primer blanco de la variable **nombreCompletoTrabajador**, y la asigne a la variable **nombreTrabajador**. Ya tendremos identificado el nombre para hacer la búsqueda. ¿Cómo? Míralo.

```
String nombreTrabajador = nombreCompletoTrabajador.substring(0, posicionPrimerBlan
```

AUTOEVALUACIÓN

Respecto a las funciones en Java, podemos afirmar que:

- a) Como ejemplo de función, podemos afirmar que todo método o procedimiento que devuelve una subcadena a partir de una cadena, puede considerarse como una función.
- b) El concepto de función en un lenguaje se extiende a cualquier procedimiento o método que devuelve un dato de cualquier tipo.
- c) Como ejemplo de función, podemos afirmar que todo método o procedimiento que devuelve un valor numérico como resultado de unos cálculos matemáticos concretos dados unos valores de entrada, puede considerarse como una función.
- d) Todas las anteriores son ciertas.

Comprobar

Operadores, expresiones e instrucciones

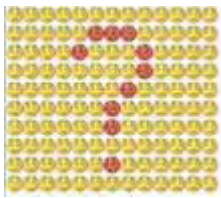
11. Instrucciones o sentencias.

Unidad Didáctica III

Instrucciones o sentencias

*Volviendo a la conversación que tiene **Carmen** con un amigo en el Puerto, ella le explica que realmente un lenguaje de programación tiene su propia gramática, eso sí mucho más sencilla que la de un idioma. Para una programadora como ella el diálogo con el ordenador es normalmente más bien un monólogo en el que **Carmen** le pide que realice determinadas tareas a través de sentencias, lo que en un idioma coloquial podríamos denominar frases más o menos elaboradas.*

Hasta ahora hemos estado viendo cómo a partir de los caracteres el compilador formaba palabras (tokens) y cómo a partir de los tokens se formaban las expresiones. Incluso hacíamos una comparación entre las expresiones y los sintagmas de una oración. Pero falta ese último paso, el que consiste en formar la oración, con significado pleno, a partir de los sintagmas. En el caso del lenguaje de programación, ¿cuál será la analogía?



En el lenguaje de programación, tendremos que formar las sentencias o instrucciones a partir de las distintas expresiones que hemos visto hasta ahora.

Las sentencias en un lenguaje de programación son el equivalente a las oraciones en el lenguaje natural. Es decir, las sentencias representan acciones completas a realizar por el programa. Algunas de las sentencias tendrán como misión controlar el orden de ejecución de las demás, y serán las llamadas sentencias de control del flujo de ejecución. Todo programa no es más que un conjunto de sentencias o instrucciones.

Debemos tener en cuenta que:

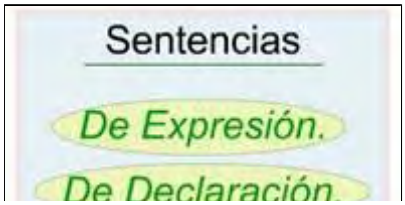
- **Sólo las expresiones de asignación forman sentencias por sí mismas, sin más que añadirles algún carácter o token de finalización de sentencia. Ese carácter en Java es el punto y coma.**
- **Todas las sentencias deberán terminar en punto y coma**.
- **Serán sentencias de asignación las obtenidas por el uso del operador de asignación básico o por cualquiera de los operadores combinados de asignación**.
- **También se consideran sentencias de asignación las obtenidas mediante los operadores de incremento o decremento, ya que llevan implícita una asignación.**

Ejemplos de sentencias de asignación:

```
a = 3;
a + = 3;
a++;
nombreTrabajador = nombreCompletoTrabajador.substring(0,posicionPrimerBlanco);
```

Existen tres tipos básicos de sentencias:

- **Sentencias de expresión**. **Están formadas por una expresión seguida del símbolo de terminación** (el punto y coma, en Java). En los párrafos anteriores hemos mencionado las sentencias de asignación, que constituyen el principal grupo de sentencias de expresión.



- **Sentencias de declaración**. **La declaración de una variable, en la que se le asocia al identificador de la variable un tipo, y opcionalmente un valor, genera una sentencia, al añadirle el símbolo de terminación**.

- **Sentencias de control de flujo**. **Se encargan de "contener" a las otras sentencias del programa, de forma que se indique el orden en que se van a ejecutar esas sentencias, y bajo qué condiciones**. Pueden considerarse como sentencias estructurales dentro del programa.

AUTOEVALUACIÓN

Señala la afirmación correcta:

a) Declarando una variable, asociando al identificador de la variable un tipo, y opcionalmente un valor, se genera una sentencia de declaración al añadirle el símbolo de terminación.

b) Las sentencias de control de flujo pueden considerarse como sentencias estructurales dentro del programa.

c) Las sentencias de declaración están formadas por una expresión seguida de un símbolo de terminación

d) Todas las anteriores son correctas

Comprobar

Operadores, expresiones e instrucciones

11.1. Sentencias de expresión.

Unidad Didáctica III

Sentencias de expresión

En el apartado 10, dedicado a las funciones aparecen buenos ejemplos de sentencias de expresión, cuando Carmen usaba algunos métodos de Java para extraer sólo el nombre de una cadena que incluía el nombre y los apellidos. Un ejemplo:

```
nombreCompletoTrabajador = "Narciso Jáimez Toro";
```

No es más que una expresión de asignación, a la que se le ha añadido un punto y coma.

Hemos comentado que las sentencias de expresión se forman añadiendo punto y coma a una expresión. Pero ¿cualquier expresión será válida? Hemos visto que cualquier literal o cualquier identificador son en sí mismo expresiones, pero ¿tienen sentido completo las siguientes expresiones seguidas de punto y coma?

```
3.0;
numero;
"hola";
4;
```

Parece que desde luego la respuesta es no. Estas expresiones por sí solas no constituyen una "acción", algo que deba hacer el ordenador. Añadir punto y coma sin más a cualquier expresión no aumenta el significado de ésta.

No todas las expresiones pueden formar sentencias de expresión. O más concretamente, no todos los operadores dan lugar a expresiones susceptibles de formar sentencias.

Operadores, expresiones e instrucciones

11.2. Tipos de expresiones que generan sentencias de expresión.

Unidad Didáctica III

Tipos de expresiones que generan sentencias de expresión

Una vez visto que no todas las expresiones pueden generar sentencias sin más que añadir punto y coma, vamos a ver cuales son las que sí las generan, las expresiones que ya tienen significado por sí mismas:

- **Expresiones de asignación**: es decir, expresiones cuyo operador raíz es bien el igual (=) o un igual compuesto (op=). Ya nos hemos referido a ellas en el apartado anterior, a modo de introducción al hablar del concepto de sentencia. Una de las acciones más simples que le podemos dar a un ordenador es indicarle que almacene un valor en una variable. Siguiendo con el ejemplo del caso, la sentencia que le asigna valor a la variable posiciónPrimerBlanco, es una expresión de asignación. Basta con añadirle un punto y coma para que tenga sentido completo para el compilador. De hecho, tiene sentido por sí misma, pero el compilador necesita el punto y coma para saber que ahí termina la sentencia, y empieza la siguiente.

```
posicionPrimerBlanco =  nombreCompletoTrabajador.indexOf(' ')
```

- **Expresiones de incremento y decremento**: se trata de las expresiones formadas por las formas prefijas o postfijas de los operadores de incremento (++) o decremento (--). Estas expresiones en el fondo pueden considerarse como expresiones de asignación, por llevar una asignación implícita.

  Siguiendo con el ejemplo del nombre. Si lo que quisiéramos es sacar el listado de los trabajadores de un determinado apellido, necesitaríamos aislar ese primer apellido de la cadena **nombreCompletoTrabajador.** Para ello, una vez localizado el primer blanco, necesitaremos sumarle uno a la posición en la que se encontró. Esa será la posición en la que comenzará el apellido.¿Cómo sumar uno a la posición del primer blanco? **posicionPrimerBlanco++;** Realmente, estamos asignando un valor a esta variable, que es justamente el que resulta de sumarle 1.

- **Llamadas a métodos o procedimientos**: Un método es una función de una clase en Java que realiza una cierta operación sobre dicha clase. Por ejemplo, abs() es un método de la clase Math para calcular el valor absoluto de un número. La llamada a un método, pasando los correspondientes **argumentos**, es así mismo una expresión. Y dicha expresión, seguida del símbolo de terminación, constituye una sentencia del tipo denominado sentencias de expresión.

  En nuestro ejemplo, Carmen ha introducido todas las operaciones necesarias para sacar el listado de los trabajadores en un método llamado listarPorNombre(), al que se le pasa como argumento un String, que será el nombre a listar. Cuando queramos usarlo, tendremos que hacer una llamada a este método indicando el nombre concreto que queremos listar. Una llamada tendrá la forma:

```
listarPorNombre("Narciso")
```

  Tiene un significado claro. Queremos que se hagan todas las operaciones necesarias para que se muestre el listado de todos los trabajadores cuyo nombre sea "Narciso". Basta con añadirle un punto y coma, y tendremos una sentencia válida.

- **Expresiones de creación de objetos**: La utilización del operador **new** para crear objetos genera expresiones de creación de objetos. Estas expresiones, seguidas del símbolo de terminación, dan lugar también a sentencias.

En nuestro ejemplo tenemos claramente la necesidad de definir lo que para nosotros va a ser un trabajador. En la unidad 2 dedicada a los tipos de datos ya explicamos lo que eran las clases y los objetos. Carmen, al hacer la aplicación de nóminas, habrá tenido que definir lo que es un trabajador en una clase llamada Trabajador. Debe definir qué es un trabajador, y qué vamos a poder hacer con él. Posteriormente, cada vez que se de de alta a un nuevo trabajador en la empresa, la aplicación deberá crear un nuevo objeto de tipo Trabajador, una nueva instancia de la clase Trabajador. Eso conlleva que algún programa se debe encargar de buscar espacio libre en memoria donde alojar los datos correspondientes a ese **nuevo objeto**. Esto es lo que hace el operador new. Veamos un ejemplo.

```
Trabajador  t1=new Trabajador("Narciso Jáimez Toro")
```

Estamos indicándole al compilador que busque en memoria espacio libre para alojar un nuevo objeto de tipo Trabajador, que el Trabajador lo tiene que crear con el nombre "Narciso Jáimez Toro", y alojarlo en esa posición, a la que nos vamos a referir mediante la variable **t1.** Una vez más, basta con añadir un punto y coma para tener una sentencia válida.

Tanto las llamadas a métodos como la creación de objetos se estudiarán detenidamente en unidades siguientes, aunque ya se han introducido en algunos ejemplos.

AUTOEVALUACIÓN

Los tipos de expresiones que dan lugar a sentencias de expresión son:

- a) Expresiones de asignación, de incremento pero no de decremento, de llamadas a procedimientos y de creación de objetos.
- b) Expresiones de asignación, de incremento, de llamadas a procedimientos y de creación de variables.
- c) Expresiones de asignación, de incremento y decremento, de llamadas a procedimientos y de creación de variables
- d) Expresiones de asignación, de incremento y decremento, de llamadas a procedimientos y de creación de objetos

Comprobar

Operadores, expresiones e instrucciones

11.3. Sentencias de declaración.

Unidad Didáctica III

Sentencias de declaración

Hemos visto que las sentencias tienen sentido completo, indican alguna acción a realizar por el programa. Hemos visto también que el segundo tipo de sentencias lo constituyen las sentencias de declaración. ¿Recuerdas lo que es la declaración de una variable? Se trata sólo de indicar el tipo junto al nombre de una variable. ¿Qué tipo de acción estará indicando una declaración? Al declarar una variable, estamos dando una orden concreta al programa. ¿Recuerdas cuál?

Sería algo como lo que sigue:

- "Comprueba el tipo de la variable para ver qué tamaño ocupan en memoria los valores que va a almacenar."
- "Busca una zona de memoria que esté libre y donde quepan los valores de ese tipo."
- "Asocia esa zona de memoria con el nombre o identificador de la variable."
- "A partir de aquí, sustituye cualquier aparición del nombre de esa variable por la zona de memoria que tiene asociada."

Parece mucho, pero es lo que significa una simple sentencia como **int varEnt;** El tipo int indica que es un número entero, que ocupa 32 bits. Se busca una zona de memoria libre donde quepan esos 32 bits, para poder almacenar cualquier número entero, y se le asocia el nombre **varEnt.** Cada vez que en el programa aparezca escrito ese nombre, se sustituirá por el número entero que contenga la dirección de memoria asociada. En resumen: se crea una nueva variable que se llama **varEnt** y se indica que es de tipo entero.

Aunque las sentencias de declaración pueden ser variadas según los lenguajes que se usen, vamos a presentarlas tal y como son en Java, ya que en cualquier caso, **aunque la sintaxis varíe, el objetivo de estas sentencias es el mismo: indicar el tipo y el nombre de una variable o de una constante, y opcionalmente darle un valor inicial antes de utilizarla en el programa.**

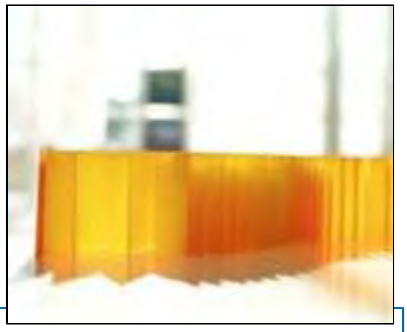
Operadores, expresiones e instrucciones

11.4. Posibles formas de declarar variables.

Unidad Didáctica III

Posibles formas de declarar variables

¿Cuál es uno de los factores que más influye en el coste de cualquier producto? La mano de obra, ¿verdad?.



Aplicado al desarrollo de programas, la productividad de un programador se mide en líneas de código.

Será interesante conseguir que se pueda reducir el número de líneas de código que tiene que teclear un programador siempre y cuando con menos líneas consigamos hacer lo mismo.

Abreviar la escritura de los programas se traduce en mayor productividad por programador, que redunda en menor coste de desarrollo.

¿Dónde queremos ir a parar.? Las sentencias de declaración son siempre numerosas en los programas, y si conseguimos reducir y abreviar el código necesario para escribirlas, habremos ayudado a simplificar nuestro programa. Por eso existen varias opciones que facilitan la labor de declarar e inicializar las variables de un programa:

- **Una sentencia de declaración puede declarar simultáneamente varias variables si son del mismo tipo:**

```
int var1,var2,var3,var4;
```

En esta sentencia se han declarado cuatro variables de tipo int. La sentencia anterior es en todo equivalente a:

```
int var1;
int var2;
int var3;
int var4;
```

- **Una sentencia de declaración permite asignar además del tipo un valor inicial a una variable:**

```
int var = 25;
```

Lo que se coloca en el lado derecho puede ser cualquier expresión del lenguaje, que devuelva un valor del tipo correspondiente a la declaración. De esta forma, estas sentencias de declaración e inicio se convierten en una mezcla donde interviene tanto una sentencia de declaración como una sentencia de expresión, en concreto, de asignación.

- **Es posible mezclar la capacidad de inicialización de la sentencia de declaración con la posibilidad de declaración de múltiples variables:**

```
int v1, v2 = 2, v3;
```

Pero en este caso es importante tener en cuenta que aunque se hayan declarado tres variables de tipo int, sólo v2 ha sido inicializada. Las dos variables restantes han sido declaradas, pero no se les ha asignado ningún valor inicial.

No todos los lenguajes obligan a declarar las variables antes de ser usadas. Pero el hecho de no saber cuál va a ser el tipo de una variable hace que no se sepa cuanto tamaño va a ocupar, e impide al compilador reservar en memoria el espacio necesario para la misma, con lo que no se puede gestionar de forma eficiente la memoria.

Por ello, la mayoría de los lenguajes modernos son fuertemente tipados, es decir que obligan a indicar siempre el tipo de todas las variables antes de usarlas (declararlas). Java es fuertemente tipado.

Cabe destacar el hecho de que **muchos lenguajes disponen de una sentencia nula**, que se puede usar para colocar en cualquier lugar donde la sintaxis del programa requiera de una sentencia pero no queramos que esa sentencia tenga ningún efecto. La sentencia nula en Java es el punto y coma sin más (;) No obstante, si las cosas se hacen bien, siempre puede evitarse el uso de este tipo de sentencias.

Operadores, expresiones e instrucciones

11.5. Sentencias de control de flujo.

Unidad Didáctica III

Sentencias de control de flujo.

Piensa que los programas pretenden resolver cualquier tipo de problema. ¿Serías capaz de resolver los problemas que te plantean sin comprobar condiciones, y en función de que sean ciertas o no hacer una cosa u otra? Algo parecido ocurre en los programas. A fin de cuentas, con los ordenadores y con los lenguajes de programación no hacemos otra cosa que intentar imitar nuestra lógica y nuestra forma de pensar y resolver los problemas.

En todos los lenguajes de programación son necesarias ciertas sentencias de control del flujo de ejecución del programa, que indiquen el orden de ejecución de las sentencias, bajo qué condiciones deben ejecutarse o no, o si deben ejecutarse repetidas veces.

Ésta es la función de las **sentencias de control de flujo, que pueden considerarse como estructuras dentro del programa**. Este tipo de sentencias se verán con más detalle en la próxima unidad de forma genérica para expresar algoritmos, y con la sintaxis concreta de Java en las unidades dedicadas a la introducción al lenguaje.

Esas sentencias o estructuras de control de flujo son básicamente 3:

- Secuencial.
- Condicional o selectiva.
- Cíclica, repetitiva o iterativa.

Aunque en cada lenguaje puede haber, y de hecho hay, más de una sentencia para cada tipo.

Por último añadir que en cualquier parte en que la sintaxis de un lenguaje de programación permita colocar una sentencia, también se podrá colocar todo un bloque de sentencias, delimitado de alguna forma, mediante algún símbolo o palabra reservada especial.

Por ejemplo, en Java los bloques de sentencias se delimitan con llaves.

```
{
   sentencia_1;
   sentencia_2;
 ...
   sentencia_n;
}
```

Operadores, expresiones e instrucciones

11.6. Cuadro resumen de los tipos de sentencias.

Unidad Didáctica III

Cuadro resumen de los tipos de sentencias

Para una rápida referencia, que ayude a recordar los conceptos tratados sobre tipos de sentencias, te ofrecemos el siguiente esquema:

- Sentencias
  - De expresión
    - De asignación
    - De incremento y decremento
    - Llamadas a métodos, procedimientos y funciones
    - Creación de objetos
  - De declaración
    - Declaración del tipo de una variable
    - Declaración e inicialización de una variable
    - Declaración e inicialización de varias variables del mismo tipo
  - De control de flujo
    - Secuencial
    - Condicional o Selectiva
    - Cíclica,repetitiva o iterativa

Operadores, expresiones e instrucciones