

## Introducción

Java es un lenguaje basado en la programación orientada a objetos (POO), este tipo de programación va más allá del tipo de programación estructurada de otros lenguajes. La POO esta más orientada a solventar los problemas de la programación de una forma más natural o real, planteando para ello el concepto de objeto. El concepto de objeto representa o modela un objeto real de nuestro mundo como podría ser un vehiculo, conductor, semáforo, etc. Los objetos tienen dos características principales *el estado* y *el comportamiento*. El *estado* representa las características que marcan las diferencias con objetos del mismo tipo. Por ejemplo en un objeto vehiculo los valores de estado podrían ser color, marca, modelo, etc. Que son valores que definen el objeto pero no lo excluyen del tipo de objeto vehiculo. Por otra parte *el comportamiento* permite establecer las diferencias para distinguir a objetos de distinto tipo. Por ejemplo un objeto motocicleta, no se comporta igual que un objeto tren pero ambos pertenecen al objeto vehiculo.

Resumiendo con un ejemplo, en un tipo de objeto vehiculo, su estado podría ser ruedas, color, potencia, motor, etc. Mientras que sus comportamientos podrían ser acelerar, frenar, encender, apagar, etc.

La característica de *estado* viene definida por una serie de parámetros o variables a los cuales se les denomina *Atributos* o *Variables Miembro*, mientras que por otro lado el *comportamiento* o las acciones que puede realizar un objeto vienen establecidas por lo que se denomina *Métodos* o *Funciones Miembro*. Normalmente los términos más utilizados son *Métodos* y *Atributos*.

Otra de las ventajas de la POO es lo que se llama *encapsulación*, que consiste en que un objeto no puede acceder directamente a los atributos de otro objeto, ya que los atributos del primero se ocultan del resto de objetos y solo pueden o deben ser accesibles por métodos del propio objeto. De esta forma cuando un objeto quiere modificar los atributos de otro ha de hacerlo mediante los métodos que proporciona el primero.

Para realizar la *encapsulación* los objetos proporcionan dos partes: la *privada* y la *pública*. Siendo accesible por otros objetos solo la parte publica. La ventaja de la encapsulación es que permite crear objetos que muestren una serie de métodos públicos que pueden utilizar otros objetos ocultando por otra parte la forma en que los métodos modifican los atributos o realizan acciones. Es decir dan el comando para realizar una acción pero no dicen como lo hacen, a esto se le denomina *abstracción*.

Otro concepto de la POO son las *Clases*. Una *Clase* es la definición de un objeto, se puede plantear como una plantilla que define los atributos y métodos de un objeto pero no lo crea. A crear un objeto utilizando una clase se le denomina *crear una instancia*. De esta forma se puede reutilizar la misma plantilla (clase) para crear varios objetos.

Otra de las características de la POO es la *herencia*, que permite la reutilización de clases ya definidas para redefinir nuevas clases hijas que heredan las características de la clase madre pero a su vez tienen características propias. Por ejemplo se puede tener una clase vehículo en la cual se establecen métodos comunes a los vehículos como arrancar, parar, frenar, etc. Mientras que se pueden definir dos clases hijas como Avión y Tren que heredan los métodos de la clase madre pero a su vez pueden redefinir los métodos y atributos existentes en la clase madre o crear nuevos independientes de la clase genérica o madre. El tipo de herencia que soporta Java es el denominado *simple*, ya que una clase hija solo hereda de una clase madre.

Como se ha comentado anteriormente los métodos son las acciones que se pueden realizar con los objetos. Existen dos métodos especiales en la POO, denominados *constructor* y *destructor*. El método *constructor* se ejecuta automáticamente cuando se crea un objeto de la clase, actuando sobre la misma una vez se haya reservado la memoria para el objeto. Varios lenguajes de programación proporcionan lo que se llama *constructores por defecto*, pero lo ideal sería crear los constructores de la clase por el programador o diseñador de la misma. Por otra parte el *destructor* de una clase es ejecutado justo antes de que se libere la memoria del objeto y por tanto quede destruido. Normalmente el destructor se utiliza para realizar las acciones concernientes a liberación de memoria, grabación en archivos, inicialización, etc.

Otro concepto de la POO es el *polimorfismo* que permite que varias clases puedan tener funciones miembro con el mismo nombre pero con diferentes implementaciones. Siendo el lenguaje capaz de discernir entre ellas dependiendo del tipo de objeto sobre el que se ejecute la acción. Otra función similar es la *sobrecarga* que permite que una clase pueda tener métodos con el mismo nombre pero con distinto tipo y número de parámetros. El lenguaje es capaz de diferenciar entre las funciones miembro sobrecargadas dependiendo de los parámetros empleados en la llamada al método.

## Sintaxis del lenguaje

### Identificadores

Los *identificadores* son nombres que se aplican a las clases, atributos, métodos, variables, etc. De forma que el compilador pueda reconocerlos inequívocamente. Los identificadores son creados por el programador cuando los necesita. Pero se han de tener en cuenta ciertas condiciones a la hora de crearlos. Es conveniente que los nombres de identificador expresen su relación con el elemento al que están asociados o con la función que realizan. Por ejemplo si para un objeto factura se tuviera se tuviera que crear un atributo que refleje el nombre del cliente lo más lógico será utilizar el identificador *NombreCliente*, en vez de un identificador como *R2D2*, que aunque es un robot muy famoso puede llevar a error o confusión al programador al buscar el atributo del nombre del cliente.

Las condiciones a la hora de crear el identificador son las siguientes: se hace diferenciación entre mayúsculas y minúsculas, por tanto no es los mismo Precio que precio. Los identificadores tienen que comenzar por una letra, un símbolo de \$ o un símbolo de subrayado.

Aunque los identificadores pueden tener nombres muy extensos solo son significativos los 32 primeros caracteres para el compilador. Aun así tampoco es conveniente utilizar nombres extensos por la incomodidad que suponen para el programador a la hora de teclear los programas.

Una buena práctica para el programador es utilizar la norma de estilo denominada *Word-mixing*, en la cual la primera letra de las clases se escribe en mayúsculas y no se utilizan los signos de subrayado ni el símbolo de dólar. A los nombres de variables, objetos y métodos se aplica que su primera letra sea una minúscula.

Otra buena práctica, aunque tanto esta como la anterior son recomendaciones a la hora de programar y no son reglas estrictas, es la de utilizar en los identificadores de variables el tipo de dato al que pertenece la variable mediante la siguiente tabla:

<i>Carácter</i>	<i>Tipo</i>	<i>Ejemplo</i>
<i>a</i>	<i>array</i>	<i>aNumeros</i>
<i>b</i>	<i>boolean</i>	<i>bCasado</i>
<i>c</i>	<i>char</i>	<i>cEstadoCivil</i>
<i>f</i>	<i>float</i>	<i>fPrecio</i>
<i>i</i>	<i>integer</i>	<i>iCantidad</i>
<i>l</i>	<i>long</i>	<i>lSueldo</i>
<i>s</i>	<i>string</i>	<i>sNombre</i>

Mientras que siguiendo con esta norma para las variables que hacen referencia a objetos se suele utilizar tres letras resumiendo el nombre del objeto. Como es lógico y al igual que en la mayoría de los lenguajes de programación. No se pueden utilizar palabras clave de lenguaje como identificadores. Las palabras clave son aquellas que forman parte del lenguaje de programación y el compilador las reconoce como tal. En la siguiente tabla se muestran cuales son las palabras clave del lenguaje Java:

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

Además, el lenguaje se reserva unas cuantas palabras más, pero que hasta ahora no tienen un cometido específico. Son las siguientes:

<i>cast</i>	<i>future</i>	<i>generic</i>	<i>inner</i>
<i>operator</i>	<i>outer</i>	<i>rest</i>	<i>var</i>

## Literales

Los literales son elementos de un programa que no pueden variar, a estos también se le denominan *constantes*. Representan valores de datos de los tipos predefinidos que utiliza el lenguaje. Normalmente se utilizan para la inicialización de variables y comparaciones en expresiones. En el lenguaje Java existen expresiones literales para datos de tipo carácter, enteros, string, float y boléanos.

### Literales de tipo entero

En el lenguaje Java se pueden expresar literales de enteros en base decimal, octal y hexadecimal. Dependiendo de la precisión del número entero se pueden dividir en los siguientes tipos: byte (8 bits), short (16 bits), int (32 bits) y long (64 bits).

Los valores de los tipos enteros se almacenan por defecto como de tipo int, o sea un valor de 32 bits con signo. Si por algún motivo se necesita almacenar un literal como de tipo long se ha de añadir al final del literal una letra *l* o *L*. Esta acción hará que se guarde como un entero de 64 bits.

Para expresar los literales enteros en base decimal, no se requiere ningún tipo de formato especial. Mientras que para expresarlo en formato octal se ha de anteponer un 0 inicial al literal, y para la notación en hexadecimal se ha de anteponer el prefijo 0x o 0X. El siguiente ejemplo muestra el número en decimal 65 en las tres notaciones:

65 = Decimal

0X41 = Hexadecimal

0101 = Octal

### Literales de tipo flotante

Los literales en coma flotante constan de dos partes. La parte entera y la parte decimal, siendo de dos tipos float y double. Los float se almacenan en 32 bits y los double en 64 bits. La diferencia entre ellos es la precisión significativa de la parte decimal. Para expresar un literal en formato float se ha de añadir el carácter *f* o *F* al final del literal. Mientras que si es un double el carácter a utilizar es *d* o *D*. Otra diferencia con los literales enteros es que por defecto se almacenan como tipo double. Ambos tipos se pueden expresar en formato decimal o en notación científica.

## Literales boléanos

Los literales boléanos se expresan con dos palabras reservadas *true* y *false*. Ya que Java procede en parte de C/C++, *true* es cualquier valor que no sea 0, mientras que *false* es el valor 0.

## Literales de caracteres

Los literales que representan un solo carácter se representan entre comillas simples, mientras que los literales que representan una cadena de caracteres se representan entre comillas dobles. Los literales de un solo carácter también pueden representar los caracteres de control no imprimibles, siendo estos los siguientes:

<code>\\</code>	<i>Barra Invertida</i>
<code>\</code>	<i>Continuación</i>
<code>\b</code>	<i>Retroceso</i>
<code>\r</code>	<i>Retroceso de carro</i>
<code>\f</code>	<i>Alimentación de formulario</i>
<code>\t</code>	<i>Tabulador horizontal</i>
<code>\n</code>	<i>Línea nueva</i>
<code>'</code>	<i>Comillas simples</i>
<code>"</code>	<i>Comillas dobles</i>
<code>\udd</code>	<i>Carácter Unicode</i>
<code>\ddd</code>	<i>Carácter Octal</i>

## Literal null

El literal null se corresponde con el tipo null (nulo) que solo tiene un valor o referencia nula.

## Comentarios

Los comentarios los suelen utilizar los programadores para documentar sus programas. Mientras que para el programador son una buena ayuda, el compilador simplemente los ignora. En Java se pueden utilizar tres tipos de comentarios, expresándose de las siguientes formas:

<code>\\Comentario</code>	<i>Se ignoran todos los caracteres desde las dos barras hasta el final de línea.</i>
<code>/* Comentario */</code>	<i>Se ignoran todos los caracteres entre los dos símbolos</i>
<code>** Comentario */</code>	<i>Se ignoran todos los caracteres entre los dos símbolos</i>

## Tipos de datos en Java

También se les denomina *tipos primitivos*, y al igual que los literales existen de diferentes tipos. Estos tipos se utilizan para crear variables que el programador utiliza en sus programas. Java dispone de ocho tipos primitivos de variables, siendo estos los siguientes:

Tipo	Tamaño	Valores
<i>Boolean</i>	1 byte	<i>true</i> o <i>false</i>
<i>Char</i>	2 bytes	Unicode (Comprende el código ASCII)
<i>Byte</i>	1 byte	Entero entre -128 a 127
<i>Short</i>	2 bytes	Entero entre -32768 a 32767
<i>Int</i>	4 bytes	Entero entre -2.147.483.648 a 2.147.483.647
<i>Long</i>	8 bytes	Entero entre -9.223.372.036.854.775.808 a -9.223.372.036.854.775.807
<i>Float</i>	4 bytes	de 6 a 7 dígitos de precisión decimal.
<i>Double</i>	8 bytes	de 14 a 15 dígitos de precisión decimal.

Cuando se quiere declarar una variable para uno de los tipos anteriores se ha de utilizar cualquiera de las siguientes sintaxis:

```
<TipoVariable> <Identificador>;
<TipoVariable> <Identificador>=<ValorInicial>;
<TipoVariable> <Identificador>=<ValorInicial>,<Identificador>=<ValorInicial>;
<TipoVariable> <Identificador>,<Identificador>=<ValorInicial>;
```

Se ha de tener en cuenta, algunas características importantes de los tipos primitivos, siendo estas las siguientes:

El tipo *boolean* no es un valor numérico, y aunque Java en parte sus conceptos descienden de *c/c++*, el *boolean* no se identifica con el igual o distinto de cero. Por lo tanto solo admite valores *true* o *false*.

El tipo *char* es de tipo Unicode y ocupan 16 bits por carácter y comprenden prácticamente los caracteres de cualquier idioma.

Los tipos numéricos *byte*, *short*, *int* y *long* son números enteros que pueden ser positivos o negativos. Pero no existe el tipo *unsigned* a diferencia de *c/c++*.

Se utiliza la palabra *void* para indicar la ausencia de un tipo determinado. Y a diferencia de otros lenguajes en Java un tipo en concreto siempre utiliza la misma cantidad de memoria independientemente del sistema operativo.

Los siguientes ejemplos muestran como se crean diferentes tipos de variables de los distintos tipos primitivos:

```
Byte edad;  
Short alturaRectangulo=100;  
Int precioInicial=200,precioFinal=300;  
Long cantidadInicial,cantidadFinal=1000;  
Float precioIva=123.45;  
Double distanciaMm;  
Boolean verdad;  
Boolean correctoOk=true;  
Char letra;  
Char Inicial='F';  
String nombrePersona;  
String direccionWeb="www.aprendoencasa.com"
```