

### 3. Tipos primitivos de dato

#### Objetivos:

- a) Describir los tipos de datos primitivos (numéricos, booleano y de tipo carácter) en el lenguaje de programación Java y su formato de representación
- b) Escribir la declaración de constantes y variables de cualquiera de los tipos de datos primitivos
- c) Presentar el concepto de conversión y los distintos tipos de conversiones de datos de tipo primitivo en Java

Todo lenguaje de programación consta de elementos específicos que permiten realizar las operaciones básicas de la programación: tipos de datos, operadores e instrucciones o sentencias. En este apartado se introducen los distintos tipos de dato que pueden emplearse en la programación con Java. En concreto, se presentan los tipos primitivos en Java, así como las constantes y las variables. En los capítulos sucesivos se mostrarán el resto de elementos básicos de programación, incluyendo otras estructuras de datos más complejas.

#### 3.1. Categorías de tipos de datos

Los tipos de datos utilizados en Java se pueden clasificar según diferentes categorías:

- a) de acuerdo con el tipo de información que representan. Esta correspondencia determina los valores que un dato puede tomar y las operaciones que se pueden realizar con él. Según este punto de vista, pueden clasificarse en:
  - Datos de tipo **primitivo**: Representan un único dato simple que puede ser de tipo char, byte, short, int, long, float, double, boolean. Por ejemplo: 'a', 12345, 750.68, False,... Cada tipo de dato presenta un conjunto de valores o constantes *literales*.
  - Variables *referencia* (variables *arrays*, de una clase/instancias, interfaces . ..). Se implementan mediante un nombre o referencia (puntero) que contiene la dirección en memoria de un valor o conjunto de valores (objeto creado con new).
- b) según cambie su valor o no durante la ejecución del programa. En este caso, se tienen:
  - **Variables**: sirven para almacenar datos durante la ejecución del programa; el valor asociado puede cambiar varias veces durante la ejecución del programa.
  - **Constantes** o variables **finales**: también sirven para almacenar datos pero una vez asignado el valor, éste no puede modificarse posteriormente.
- c) según su papel en el programa. Pueden ser:
  - Variables *miembro* de una clase. Se definen dentro de una clase, fuera de los métodos. Pueden ser de tipos primitivos o referencias y también variables o constantes.
  - Variables *locales*. Se definen dentro de un método o, en general, dentro de cualquier bloque de sentencias entre llaves {}. La variable desaparece una vez finalizada la ejecución del método o del bloque de sentencias. También pueden ser de tipos primitivos o referencias.

A continuación se describen cada uno de estos tipos de dato.

### 3.2. Tipos de dato primitivos en Java

A todo dato (constante, variable o expresión) le corresponde un **tipo** específico en Java. Como se ha indicado anteriormente un tipo de dato determina los valores que pueden asignarse a un dato, el formato de representación correspondiente y las operaciones que pueden realizarse con dicho dato.

En Java casi todo es un objeto. Existen algunas excepciones como, por ejemplo, los tipos primitivos, tales como `int`, `char`, etc., que no se consideran objetos y se tratan de forma especial. Java tiene un conjunto de tipos *primitivos* para representar datos enteros (cuatro tipos diferentes), para datos numéricos reales en coma flotante (dos tipos diferentes), para caracteres y para datos lógicos o booleanos. Cada uno de ellos tiene **idéntico** tamaño y comportamiento en todas las versiones de Java y para cualquier tipo de ordenador. Esto implica que no hay directivas de compilación condicionales y asegura la **portabilidad** de los programas a diferencia de lo que ocurre, por ejemplo, con el lenguaje de programación C.

❖ **Nota de interés.** En otros lenguajes de programación el formato y tamaño de los tipos de dato primitivos puede depender de la plataforma o sistema operativo en la que se ejecute el programa. Sin embargo Java especifica el tamaño y formato de todos los tipos de dato primitivos para que el programador no tenga que preocuparse sobre las dependencias del sistema

Por otro lado, a partir de estos tipos primitivos de dato pueden construirse otros tipos de datos compuestos, *arrays*, clases e interfaces. En la Tabla 3.1 se muestran los tipos de dato primitivos de Java con el intervalo de representación de valores que puede tomar y el tamaño en memoria correspondiente.

Tabla 3.1. Tipos de dato primitivos de Java

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
<code>boolean</code>	<code>true</code> o <code>false</code>	1	N.A.	N.A.	<code>false</code>
<code>char</code>	Carácter Unicode	16	<code>\u0000</code>	<code>\uFFFF</code>	<code>\u0000</code>
<code>byte</code>	Entero con signo	8	-128	127	0
<code>short</code>	Entero con signo	16	-32768	32767	0
<code>int</code>	Entero con signo	32	-2147483648	2147483647	0
<code>long</code>	Entero con signo	64	-9223372036854775808	9223372036854775807	0
<code>float</code>	Coma flotante de precisión simple Norma IEEE 754	32	$\pm 3.40282347E+38$	$\pm 1.40239846E-45$	0.0
<code>double</code>	Coma flotante de precisión doble Norma IEEE 754	64	$\pm 1.79769313486231570E+308$	$\pm 4.94065645841246544E-324$	0.0

Los tipos de datos numéricos en Java no pueden representar *cualquier* número entero o real. Por ejemplo, el tipo de dato entero `int` tiene un intervalo de representación entre -2147483648 y 2147483647. Si se desea representar el valor correspondiente a la población mundial del planeta (más de 6 mil millones de habitantes) no puede hacerse con dato de tipo `int`. Los datos de tipo

`double` no tienen tanta limitación. Pueden alcanzar un valor del orden de  $10^{308}$ , pero tienen otro problema: la precisión. En concreto, el tipo `double` tiene una precisión de 15 dígitos significativos como se detallará más adelante.

### 3.3. Literales o constantes literales de tipos de dato primitivos

Un **literal**, valor literal ó constante literal es una constante cuyo nombre o identificador es la representación escrita de su valor y tiene ya ese significado en el código fuente de un programa Java. Seguidamente se muestran algunos ejemplos de **valores o constantes literales** pertenecientes a los tipos primitivos que pueden utilizarse directamente en un programa fuente de Java.

Las constantes literales **booleanas** son `false` y `true`.

Las constantes literales de tipo **carácter** aparecen entre comillas simples. Como ya se ha comentado anteriormente, los caracteres, cadenas e identificadores en Java se componen de caracteres pertenecientes al conjunto de caracteres Unicode (<http://www.unicode.org>). Un dato de tipo carácter representa un único carácter. El formato de Unicode utiliza 16 bits (2 bytes) para poder codificar un total de 65536 caracteres diferentes que incluyen caracteres y símbolos procedentes de distintas lenguas del mundo. En este conjunto de caracteres encontramos letras mayúsculas ('A', 'B', 'C',...), letras minúsculas ('a', 'b', 'c',...), signos de puntuación (',', ';', ':' ...), dígitos ('0', '1', '2',...), símbolos especiales ('#', '&', '%',...) y caracteres de control (tabulador, retorno de carro,...).

Hay algunos caracteres que pueden causar algún problema en el código fuente de un programa Java debido a que se utilizan para tareas específicas dentro del lenguaje. Por ejemplo, como se verá más adelante, el carácter correspondiente a las comillas dobles (") se emplea para delimitar una cadena de caracteres. Para solucionar este inconveniente, existen diferentes formas de indicar una constante literal de tipo carácter. Por ejemplo:

```
't'           // Representa la letra minúscula, con comillas simples
\t           // Con la barra de pendiente negativa: secuencia de escape
\u0234       // Con la barra y la letra u: secuencia Unicode
```

Una *secuencia de escape* es una serie de caracteres que comienza por el carácter `\` seguido de una letra, un conjunto de dígitos o la letra `u` seguida de un conjunto de dígitos y que representa a un carácter. En este último caso, la secuencia de caracteres o secuencia de escape `\uxxx` puede emplearse en cualquier lugar en un programa de Java para representar un carácter Unicode, siendo `xxxx` una secuencia de cuatro dígitos hexadecimales. En la Tabla 3.2 se muestran algunos ejemplos de secuencias de escape.

Tabla 3.2. Caracteres representados por secuencias de escape

Secuencia de escape	Valor
<code>\b</code>	Retroceso o <i>backspace</i> (equivalente a <code>\u0008</code> )
<code>\t</code>	Tabulador (equivalente a <code>\u0009</code> )
<code>\n</code>	Nueva línea (equivalente a <code>\u000A</code> )
<code>\f</code>	Salto de página (equivalente a <code>\u000C</code> )
<code>\r</code>	Retorno de carro (equivalente a <code>\u000D</code> )
<code>\"</code>	Doble comilla (equivalente a <code>\u0022</code> )
<code>\'</code>	Comilla simple (equivalente a <code>\u0027</code> )

<code>\\</code>	Barra diagonal (equivalente a <code>\u005C</code> )
<code>\xxx</code>	Carácter correspondiente al valor octal <code>xxx</code> (donde <code>x</code> es un número del 0 al 7)
<code>\uxxxx</code>	Carácter correspondiente al valor <code>xxxx</code> codificado según Unicode mediante cuatro dígitos hexadecimales

Aunque en realidad una cadena no es un tipo primitivo de Java pueden construirse constantes literales de cadenas de caracteres. Las constantes literales de **cadena**s de texto se indican entre comillas **dobles**. Por ejemplo:

```
"Hola, mundo" // Con comillas dobles: indica una cadena
```

Al construir una cadena de caracteres se puede incluir cualquier carácter Unicode excepto un carácter de nueva línea. Si se desea incluir un salto de línea en una cadena de caracteres debe utilizarse la secuencia de escape `\n`.

Las constantes **enteras** son secuencias de dígitos octales, decimales o hexadecimales en las que no se emplea el punto o coma decimal. Si comienza por un `0` indica un formato **octal**. Si comienza por un `0x` ó `0X` indica un formato **hexadecimal**. El resto se considera en formato **decimal**. Las constantes de tipo `long` se indican con una `l` o una `L` al final. Normalmente se emplea la `L` para no confundir con el `1` (uno). Si no se indica ninguna de estas terminaciones se supone un tipo `int`. A continuación se muestran algunos ejemplos de constantes enteras:

```
34 // de tipo int, solo digitos
-78 // digitos sin punto decimal
034 // en octal (equivale al 28 decimal)
0x1C // en hexadecimal (equivale al 28 decimal)
875L // de tipo long
```

Las constantes **reales** o en coma flotante se expresan con coma decimal y opcionalmente seguidos de un exponente. El valor puede finalizarse con una `f` o una `F` para indica el formato `float` (por defecto es `double`). Por ejemplo:

```
15.2 // de tipo double
15.2D // el mismo valor de tipo double
1.52e1 // el mismo valor de tipo double
0.152E2 // el mismo valor de tipo double
.8e10 // de tipo double
15.8f // de tipo float
15.8F // tambien de tipo float
```

Como se verá más adelante cada tipo de dato primitivo tiene una clase correspondiente (`Boolean`, `Character`, `Byte`, `Short`, `Integer`, `Long`, `Float` y `Double`), llamadas *wrappers*, que definen también constantes y métodos útiles.

### 3.4. Formato de representación de los datos numéricos reales

Los números reales se representan en el ordenador según un formato de coma flotante. Los bits empleados para la representación de un valor real se dividen en dos componentes: **mantisa** y **exponente**. En el sistema binario, la expresión matemática que relaciona la mantisa, el exponente y el valor del número real representado es:

$$\text{Valor} = \text{mantisa} \cdot 2^{\text{exponente}}$$

Un número real de tipo `float` utiliza 32 bits de los cuales 24 son para la mantisa y 8 para el exponente. La mantisa representa un valor entre  $-1.0$  y  $1.0$  y el exponente representa la potencia de 2 necesaria para igualar la mantisa al valor que se quiere representar conjuntamente.

Los números reales se caracterizan por dos magnitudes: la **precisión** y el **intervalo de representación**. La precisión es el número de dígitos significativos con los que se puede representar un número y el intervalo de representación es la diferencia entre el mayor y el menor número que se pueden representar.

La precisión de un número real depende del número de bits de su mantisa mientras que el intervalo depende del número de bits de su exponente.

Una mantisa de 24 bits como en el caso del tipo `float` puede representar aproximadamente  $\pm 2^{23}$ , o sea, cerca de 7 dígitos decimales significativos.

Un exponente de 8 bits puede representar multiplicadores entre  $2^{-128}$  y  $2^{127}$  de forma que el intervalo de representación es de  $10^{-38}$  a  $10^{38}$  aproximadamente. En consecuencia los datos de tipo `float` tienen un intervalo de representación mucho mayor en el mayor de los enteros pero con sólo siete dígitos significativos de precisión.

Cuando un valor con más de siete cifras significativas se almacena en una variable de tipo `float` sólo se consideran los siete dígitos decimales más significativos. Los dígitos restantes se pierden. Por ejemplo si se almacena el valor 1234,56789 en una variable de tipo `float`, el valor se redondeará a 1234,568. Esta diferencia entre el valor original y el valor representado en el ordenador se denomina error de redondeo. Por este motivo, es muy importante seleccionar el tipo de dato numérico real con la precisión suficiente con el fin de considerar las cifras necesarias para resolver un problema particular correctamente.

Por su parte, un número real de tipo `double` utiliza 64 bits en la memoria del ordenador dividida en 53 bits para la mantisa y 11 para el exponente. Una mantisa de 53 bits permite representar entre 15 y 16 dígitos decimales significativos así que la precisión es de 15 dígitos decimales. Un exponente de 11 bits permite representar multiplicadores entre  $2^{-1024}$  y  $2^{1024}$ , de forma que el intervalo de representación va de  $10^{-308}$  a  $10^{308}$  aproximadamente.

Una constante real se distingue de un entero porque contiene un punto decimal (no una coma) y/o un exponente. Si la constante es positiva puede escribirse con o sin el signo `+`. El tipo de una constante real se puede especificar añadiendo la letra `F` para el tipo `float` o la letra `D` para `double`. Por defecto, una constante real es de tipo `double`. En este caso debe estar en el intervalo de representación comprendido entre los valores numéricos reales  $-1.79769313486231570E+308$  y  $1.79769313486231570E+308$ .

### 📌 Nota de interés

En la programación con Java es recomendable emplear el tipo `double` cuando se trabaja con tipos de dato reales, considerando el funcionamiento de los algoritmos y métodos matemáticos y de los recursos de los sistemas informáticos que se emplean en la actualidad.

### 3.5. Declaraciones de variables

Una **variable** es un espacio de la memoria correspondiente a un dato cuyo valor puede modificarse durante la ejecución de un programa y que está asociado a un **identificador**. Toda variable ha de declararse antes de ser usada en el código de un programa en Java. En la declaración de una variable debe indicarse explícitamente el identificador de la variable y el **tipo** de dato asociado. El tipo de dato determina el espacio reservado en memoria, los diferentes valores que puede tomar la variable y las operaciones que pueden realizarse con ella. La **declaración** de una variable en el código fuente de un programa de Java puede hacerse de la siguiente forma:

```
tipo_de_dato identificador_de_la_variable;
```

o bien, la declaración de múltiples variables (con los correspondientes identificadores separados por comas) del mismo tipo:

```
tipo_de_dato ident_1, ident_2, . . . , ident_n;
```

Por ejemplo:

```
int n;
double x, y;
```

En el primer ejemplo se declara `n` como una variable de tipo `int`. En el segundo ejemplo se declaran dos variables `x` e `y` de tipo `double`. En Java una variable queda definida únicamente dentro del bloque de sentencias (entre llaves `{ }`) en el que ha sido declarada. De esta forma queda determinado su ámbito o alcance (*scope*) en el que puede emplearse.

El identificador elegido para designar una variable debe respetar las normas de construcción de identificadores de Java. Además, por convención:

- los identificadores de las variables comienzan con una letra minúscula. Por ejemplo: `n`, `x2`, `mes`, `clave`, `suma`, ó `nombre`.
- si el identificador es una palabra compuesta, las palabras restantes comienzan por una letra mayúscula. Por ejemplo: `esDivisible`.
- El carácter del subrayado puede emplearse en cualquier lugar del identificador de una variable pero suele emplearse para separar nombres en identificadores de constantes.

La declaración e inicialización de una variable de tipo primitivo puede realizarse de forma simultánea en la misma línea empleando el operador **asignación** `=`. Por ejemplo:

```
int n = 15;
```

Independientemente de haber inicializado o no, el valor asignado a la variable puede modificarse las veces que se quiera durante la ejecución del programa. También puede realizarse la declaración e inicialización de varias variables del mismo tipo primitivo en la misma línea separándolas por comas. Por ejemplo:

```
double x = 12.5, y = 25.0;
```

En el caso de que no se inicialice explícitamente la variable, ésta toma el valor `0` si es numérica, `false` si es booleana y `'\0'` si es de tipo carácter.

Como se verá más adelante, la declaración de un objeto o instancia equivalente se realiza empleando la palabra `new`. Por ejemplo:

```
Integer n = new Integer(15);
```

Esta declaración y el tipo de dato (clase) `Integer` se verá más adelante con detenimiento.

### 3.6. Declaración de variables final o constantes

Las variables finales en Java son similares a las constantes empleadas en otros lenguajes de programación. Una vez inicializada una variable final su valor no puede ser modificado. La declaración de variables *finales* o **constantes** se realiza empleando la palabra reservada `final` antes del identificador del tipo de dato. Por ejemplo:

```
final int MAXIMO = 15;
```

La asignación de valor se puede posponer en el código, aunque en ningún caso su valor puede modificarse una vez ha sido inicializada ya que se generaría un error. Ejemplo de inicialización posterior a la declaración de la constante:

```
final int MAXIMO;
...
MAXIMO = 15;
```

Al igual que ocurre con las variables, el identificador elegido para designar una constante debe respetar las normas de construcción de identificadores de Java. Por convención:

- los identificadores de las constantes se componen de letras mayúsculas. Por ejemplo: `MAXIMO`.
- el carácter de subrayado (`_`) es aceptable en cualquier lugar dentro de un identificador, pero se suele emplear sólo para separar palabras dentro de los identificadores de las constantes. Por ejemplo: `MAXIMO_VALOR`.

### 3.7. Conversiones entre tipos de dato

El proceso consiste en almacenar el valor de una variable de un determinado tipo primitivo en otra variable de distinto tipo. Suele ser una operación más o menos habitual en un programa y la mayoría de los lenguajes de programación facilitan algún mecanismo para llevarla a cabo. En cualquier caso, no todas las conversiones entre los distintos tipos de dato son posibles. Por ejemplo, en Java no es posible convertir valores booleanos a ningún otro tipo de dato y viceversa. Además, en caso de que la conversión sea posible es importante evitar la pérdida de información en el proceso. En general, existen dos categorías de conversiones:

- De ensanchamiento o **promoción**. Por ejemplo: pasar de un valor entero a un real.
- De estrechamiento o **contracción**. Por ejemplo: pasar de un valor real a un entero.

Las conversiones de promoción transforman un dato de un tipo a otro con el mismo o mayor espacio en memoria para almacenar información. En estos casos puede haber una cierta pérdida de precisión al convertir un valor entero a real al desechar algunos dígitos significativos. Las conversiones de promoción en Java se resumen en la Tabla 3.3.

Tabla 3.3. Conversiones de promoción entre tipos de dato en Java

Tipo de origen	Tipo de destino
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	double
float	float, double

double	-
--------	---

Las conversiones de contracción son más comprometidas ya que transforman un dato de un tipo a otro con menor espacio en memoria para almacenar información. En estos casos se corre el riesgo de perder o alterar sensiblemente la información. Las conversiones de contracción en Java se resumen en la Tabla 3.4.

Tabla 3.4. Conversiones de contracción entre tipos de dato en Java

Tipo de origen	Tipo de destino
byte	char
short	byte, char
char	byte, short
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Tanto las conversiones de promoción como las de contracción se realizan por medio de los siguientes mecanismos:

- a) Por **asignación**: cuando una variable de un determinado tipo se asigna a una variable de otro tipo. Sólo admite conversiones de **promoción**. Por ejemplo: si `n` es una variable de tipo `int` que vale 25 y `x` es una variable de tipo `double`, entonces se produce una conversión por asignación al ejecutarse la sentencia

```
x = n;
```

la variable `x` toma el valor 25.0 (valor en formato real). El valor de `n` no se modifica.

- b) Por **promoción aritmética**: como resultado de una operación aritmética. Como en el caso anterior, sólo admite conversiones de **promoción**. Por ejemplo, si `producto` y `factor1` son variables de tipo `double` y `factor2` es de tipo `int` entonces la ejecutarse la sentencia

```
producto = factor1 * factor2;
```

el valor de `factor2` se convierte internamente en un valor en formato real para realizar la operación aritmética que genera un resultado de tipo `double`. El valor almacenado en formato entero en la variable `factor2` no se modifica.

- c) Con **casting** o “**moldes**”: con operadores que producen la conversión entre tipos. Admite las conversiones de promoción y de contracción indicadas anteriormente. Por ejemplo: si se desea convertir un valor de tipo `double` a un valor de tipo `int` se utilizará el siguiente código

```
int n;
double x = 82.4;
```

```
n = (int) x;
```

la variable `n` toma el valor 82 (valor en formato entero). El valor de `x` no se modifica.

El código fuente del siguiente programa ilustra algunas de las conversiones que pueden realizarse entre datos de tipo numérico.

```
/**
 * Ejemplos de conversiones entre distintos tipos de datos numericos
 * A. Garcia-Beltran - noviembre, 2004
 */
public class Conversiones {
    public static void main (String [] args) {
        int a = 2;
        double b = 3.0;
        float c = (float) (20000*a/b + 5);
        System.out.println("Valor en formato float: " + c);
        System.out.println("Valor en formato double: " + (double) c);
        System.out.println("Valor en formato byte: " + (byte) c);
        System.out.println("Valor en formato short: " + (short) c);
        System.out.println("Valor en formato int: " + (int) c);
        System.out.println("Valor en formato long: " + (long) c);
    }
}
```

Ejemplo de compilación y ejecución del programa anterior y salida por pantalla correspondiente:

```
$>javac Conversiones.java
```

```
$>java Conversiones
Valor en formato float: 13338.333
Valor en formato double: 13338.3330078125
Valor en formato byte: 26
Valor en formato short: 13338
Valor en formato int: 13338
Valor en formato long: 13338
```