Elementos básicos del lenguaje Java

Ricardo Pérez López

IES Doñana, curso 2020/2021

Generado el 5 de enero de 2021 a las 19:16:00

Índice general

1. Tipos y valores en Java

1.1. Introducción

El lenguaje de programación Java es un **lenguaje de tipado estático**, lo que significa que cada variable y cada expresión tiene un tipo que se conoce en el momento de la compilación.

El lenguaje de programación Java también es un **lenguaje fuertemente tipado**, porque los tipos limitan los valores que una variable puede contener o que una expresión puede producir, limitan las operaciones admitidas en esos valores y determinan el significado de las operaciones.

El tipado estático fuerte ayuda a **detectar errores en tiempo de compilación**.

Los **tipos** del lenguaje de programación Java se dividen en **dos categorías**:

- Tipos **primitivos**.
- Tipos **referencia**.

Consecuentemente, en Java hay dos categorías de **valores**:

- Valores primitivos.
- Valores referencia.

Los **tipos primitivos** son:

- El tipo **booleano** (`boolean`).
- Los tipos **numéricos**, los cuales a su vez son:
  - Los tipos **integrales**: `byte`, `short`, `int`, `long` y `char`.
  - Los tipos **de coma flotante**: `float` y `double`.

Los **tipos referencia** son:

- Tipos de **clase**.
- Tipos de **interfaz**.
- Tipos de ***array***.

Además, hay un tipo especial que representa el valor **nulo** (**`null`**).

En Java, un objeto es una de estas dos cosas:

- O bien es una instancia creada dinámicamente de un tipo de clase.
- O bien es un *array* creado dinámicamente.

Los valores de un tipo referencia son referencias a objetos.

Todos los objetos, incluidos los *arrays*, admiten los métodos de la clase `Object`.

Las cadenas literales representan objetos de la clase `String`.

1.2. Tipos primitivos

Los **tipos primitivos** están predefinidos en Java y se identifican mediante su nombre, el cual es una palabra clave reservada en el lenguaje.

Un valor primitivo es un valor de un tipo primitivo.

Los valores primitivos **no son objetos** y no comparten estado con otros valores primitivos.

En consecuencia, los valores primitivos no se almacenan en el montículo y, por tanto, las variables que contienen valores primitivos no guardan una referencia al valor, sino que almacenan el valor mismo.

Los tipos primitivos son los **booleanos**, los **integrales** y los tipos de **coma flotante**.

1.2.1. Booleanos

El tipo booleano (`boolean`) contiene dos valores, representados por los literales booleanos **`true`** (verdadero) y **`false`** (falso).

Un literal booleano siempre es de tipo `boolean`.

Sus operaciones son:

Igualdad:	`==`, `!=`
Complemento lógico (*not*):	`!`
And, *or* y *xor* estrictos:	`&`, `|`, `^`
And y *or* perezosos:	`&&`, `||`
Condicional ternario:	`? :`

```
jshell> true && false
$1 ==> false

jshell> false == false
$2 ==> true

jshell> true ^ false
$3 ==> true
```

```
jshell> !true
$4 ==> false

jshell> true ? 1 : 2
$5 ==> 1

jshell> false ? 1 : 2
$6 ==> 2
```

1.2.2. Integrales

Los **tipos integrales** son:

- **Enteros** (`byte`, `short`, `int` y `long`): sus valores son **números enteros con signo** en complemento a dos.
- **Caracteres** (`char`): sus valores son **enteros sin signo** que representan caracteres Unicode almacenados en forma de *code units* de UTF-16.

Sus tamaños y rangos de valores son:

Tipo	Tamaño	Rango
`byte`	8 bits	`-128` a `127` inclusive
`short`	16 bits	`32768` a `32767` inclusive
`int`	32 bits	`2147483648` a `2147483647` inclusive
`long`	64 bits	`9223372036854775808` a `9223372036854775807` inclusive
`char`	16 bits	`'\u0000'` a `'\uffff'` inclusive, es decir, de `0` a `65535`

Los literales que representan números enteros pueden ser de tipo `int` o de tipo `long`.

Un literal entero será de tipo `long` si lleva un sufijo `l` o `L`; en caso contrario, será de tipo `int`.

Se pueden usar caracteres de subrayado (_) como separadores entre los dígitos del número entero.

Los literales de tipos enteros se pueden expresar en:

- **Decimal:** no puede empezar por `0`, salvo que sea el propio número `0`.
- **Hexadecimal:** debe empezar por `0x` o `0X`.
- **Octal:** debe empezar por `0`.
- **Binario:** debe empezar por `0b` o `0B`.

Ejemplos de literales de tipo `int`:

```
0
2
0372
0xDada_Cafe
1996
0x00_FF__00_FF
```

Ejemplos de literales de tipo `long`:

```
0l
0777L
0x100000000L
2_147_483_648L
0xC0B0L
```

Un literal de tipo `char` representa un carácter o secuencia de escape.

Se escriben encerrados entre comillas simples (también llamadas *apóstrofes*).

Los literales de tipo `char` sólo pueden representar *code units* de Unicode y, por tanto, sus valores deben estar comprendidos entre `'\u0000'` y `'\uffff'`.

Ejemplos de literales de tipo `char`:

```
'a'
'%'
'\t'
'\\'
'\''
'\u03a9'
'\uFFFF'
'\177'
'™'
```

En Java, los **caracteres** y las **cadenas** son **tipos distintos**.

1.2.2.1. Operadores integrales

Java proporciona una serie de operadores que actúan sobre valores integrales.

Los **operadores de comparación** dan como resultado un valor de tipo `boolean`:

Comparación numérica:	`<`, `<=`, `>`, `>=`
Igualdad numérica:	`==`, `!=`

```
jshell> 2 <= 3
$1 ==> true

jshell> 4 != 4
$2 ==> false
```

Los **operadores numéricos** dan como resultado un valor de tipo `int` o `long`:

Signo más y menos (unarios):	`+`, `-`
Multiplicativos:	`*`, `/`, `%`
Suma y resta:	`+`, `-`
Preincremento y postincremento:	`++`
Predecremento y postdecremento:	`--`
Desplazamiento con y sin signo:	`<<`, `>>`, `>>>`
Complemento a nivel de bits:	`~`

And, *or* y *xor* a nivel de bits:	`&`, `|`, `^`

Si un operador integral (que no sea el desplazamiento) tiene al menos un operando de tipo `long`, la operación se llevará a cabo en precisión de 64 bits y el resultado de la operación numérica será de tipo `long`.

Si el otro operando no es `long`, se convertirá primero a `long`.

En caso contrario, la operación se llevará a cabo usando precisión de 32 bits, y el resultado de la operación numérica será de tipo `int`.

Si alguno de los operandos no es `int` (por ejemplo, `short` o `byte`), se convertirá primero a `int`.

Ciertas operaciones pueden lanzar excepciones. Por ejemplo, el operador de división entera (`/`) y el resto de la división entera (`%`) The integer lanzan una excepción `ArithmeticException` si el operando derecho es cero.

```
jshell> 2 + 3
$1 ==> 5              // Devuelve un int

jshell> 2 + 3L
$2 ==> 5              // Devuelve un long

jshell> 8 >> 1
$3 ==> 4

jshell> -8 >> 1
$4 ==> -4

jshell> -8 >>> 1
$5 ==> 2147483644

jshell> 2 == 3 ? 5 : 6L
$6 ==> 6
```

1.2.3. De coma flotante

Los **tipos de coma flotante** son valores que representan **números reales** almacenados en el formato de coma flotante **IEEE-754**.

Existen dos tipos de coma flotante:

- **`float`:** sus valores son números de coma flotante de 32 bits (simple precisión).
- **`double`:** sus valores son números de coma flotante de 64 bits (doble precisión).

Un literal de coma flotante tiene las siguientes partes en este orden (que algunas son opcionales según el caso):

1. Una parte entera.
2. Un punto (`.`).
3. Una parte fraccionaria.

4. Un exponente.
5. Un sufijo de tipo.

Los literales de coma flotante se pueden expresar en decimal o hexadecimal (usando el prefijo `0x` o `0X`).

Todas las partes numéricas del literal (la entera, la fraccionaria y el exponente) deben ser decimales o hexadecimales, sin mezclar algunas de un tipo y otras de otro.

Se permiten caracteres de subrayado (_) para separar los dígitos de la parte entera, la parte fraccionaria o el exponente.

El exponente, si aparece, se indica mediante el carácter `e` o `E` (si el número es decimal) o el carácter `p` o `P` (si es hexadecimal), seguido por un número entero con signo.

Un literal de coma flotante será de tipo `float` si lleva un sufijo `f` o `F`; si no lleva ningún sufijo (o si lleva opcionalmente el sufijo `d` o `D`), será de tipo `double`.

El literal positivo finito de tipo `float` más grande es `3.4028235e38f`.

El literal positivo finito de tipo `float` más pequeño distinto de cero es `1.40e-45f`.

El literal positivo finito de tipo `double` más grande es `1.79769313486231570e308`.

El literal positivo finito de tipo `double` más pequeño distinto de cero es `4.9e-324`.

Ejemplos de literales de tipo `float`:

```
1e1f
2.f
.3f
0f
3.14f
6.022137e+23f
```

Ejemplos de literales de tipo `double`:

```
1e1
2.
.3
0.0
3.14
1e-9d
1e137
```

El estándar IEEE-754 incluye números positivos y negativos formados por un signo y una magnitud.

También incluye:

- Ceros positivo y negativos:

  `+0`

  `-0`

- Infinitos positivos y negativos:

  `Float.POSITIVE_INFINITY`

  `Float.NEGATIVE_INFINITY`

  `Double.POSITIVE_INFINITY`

  `Double.NEGATIVE_INFINITY`

- Valores especiales *Not-a-Number* (o *NaN*), usados para representar ciertas operaciones no válidas como dividir entre cero:

  `Float.NaN`

  `Double.NaN`

1.2.3.1. Operadores de coma flotante

Los operadores que actúan sobre valores de coma flotante son los siguientes:

Los **operadores de comparación** dan como resultado un valor de tipo `boolean`:

Comparación numérica:	`<`, `<=`, `>`, `>=`
Igualdad numérica:	`==`, `!=`

```
jshell> 2.0 <= 3.0
$1 ==> true

jshell> 4.0 != 4.0
$2 ==> false
```

Los **operadores numéricos** dan como resultado un valor de tipo `float` o `double`:

Signo más y menos (unarios):	`+`, `-`
Multiplicativos:	`*`, `/`, `%`
Suma y resta:	`+`, `-`
Preincremento y postincremento:	`++`
Predecremento y postdecremento:	`--`

Si al menos uno de los operandos de un operador binario es de un número de coma flotante, la operación se realizará en coma flotante, aunque el otro operando sea un integral.

Si al menos uno de los operandos de un operador numérico es de tipo `double`, la operación se llevará a cabo en aritmética de coma flotante de 64 bits y el resultado de la operación numérica será de tipo `double`.

Si el otro operando no es `double`, se convertirá primero a `double`.

En caso contrario, la operación se llevará a cabo usando aritmética de coma flotante 32 bits, y el resultado de la operación numérica será de tipo `float`.

Si el otro operando no es `float`, se convertirá primero a `float`.

```
jshell> 4 / 3
$1 ==> 1

jshell> 4 / 3.0
$2 ==> 1.3333333333333333

jshell> 4 / 3.0f
$3 ==> 1.3333334

jshell> 4 / 0.0
$4 ==> Infinity

jshell> 4.0 * Double.NaN
$5 ==> NaN
```

Una operación de coma flotante que produce *overflow* devuelve un infinito con signo.

Una operación de coma flotante que produce *underflow* devuelve un valor desnormalizado o un cero con signo.

Una operación de coma flotante que no tiene un resultado matemáticamente definido devuelve `NaN`.

Cualquier operación numérica que tenga un `NaN` como operando devuelve `NaN` como resultado.

1.2.4. Subtipado

Se dice que un tipo S es **supertipo directo** de un tipo T cuando esos dos tipos están relacionados según unas reglas que veremos luego. En tal caso, se escribe:

$$S >_1 T$$

Se dice que un tipo S es **supertipo** de un tipo T cuando S se puede obtener de T mediante clausura reflexiva y transitiva sobre la relación de *supertipo directo*. En tal caso, se escribe:

$$S :> T$$

S es un **supertipo propio** de T si $S :> T$ y $S \neq T$. En tal caso, se escribe:

$$S > T$$

Los **subtipos** de un tipo T son todos aquellos tipos U tales que T es un supertipo de U, más el tipo nulo. Cuando S es un subtipo de T se escribe:

$$S <: T$$

S es un **subtipo propio** de T si $S <: T$ y $S \neq T$. En tal caso, se escribe:

$$S < T$$

S es un **subtipo directo** de T si $T >_1 S$. En tal caso, se escribe:

$$S <_1 T$$

Las relaciones de **subtipo** y **supertipo** son muy importantes porque un valor de un tipo se puede convertir en un valor de un supertipo suyo sin perder información (es lo que se denomina **ampliación** o *widening*).

1.2.4.1. Subtipado entre tipos primitivos

Las siguientes reglas definen la relación de **subtipo directo** entre los tipos primitivos de Java:

- `float` $<_1$ `double`
- `long` $<_1$ `float`
- `int` $<_1$ `long`
- `char` $<_1$ `int`
- `short` $<_1$ `int`
- `byte` $<_1$ `short`

1.2.5. Conversiones entre datos primitivos

Es posible convertir valores de un tipo a otro, siempre y cuando se cumplan ciertas condiciones y teniendo en cuenta que, en determinadas ocasiones, puede haber pérdida de información.

Por ejemplo, no es posible convertir directamente valores numéricos en booleanos o viceversa.

Pero sí es posible convertir valores numéricos a otro tipo numérico, aunque es posible que se pueda perder información, según sea el caso.

Por ejemplo, convertir un número de coma flotante en un entero supondrá siempre la pérdida de la parte fraccionaria del número.

Igualmente, es posible que haya pérdida de información al convertir un número de más bits en otro de menos bits.

1.2.5.1. *Casting*

El ***casting*** o *moldeado* de tipos es una operación de conversión entre tipos.

En el caso de tipos primitivos, el *casting* se usa para:

- Convertir, en tiempo de ejecución, un valor de un tipo numérico a un valor similar de otro tipo numérico.
- Garantizar, en tiempo de compilación, que el tipo de una expresión es `boolean`.

El *casting* se escribe anteponiendo a una expresión, y entre paréntesis, el nombre del tipo al que se quiere convertir el valor de esa expresión.

Por ejemplo, si queremos convertir a `short` el valor de la expresión `4 + 3`, hacemos:

```
(short) (4 + 3)
```

Los paréntesis alrededor de la expresión `4 + 3` son necesarios para asegurarnos de que el *casting* afecta a toda la expresión y no sólo al `4`.

1.2.5.2. De ampliación (*widening*)

Existen 19 conversiones de ampliación o *widening* sobre tipos primitivos:

- De `byte` a `short`, `int`, `long`, `float` o `double`.
- De `short` a `int`, `long`, `float` o `double`.
- De `char` a `int`, `long`, `float` o `double`.
- De `int` a `long`, `float` o `double`.
- De `long` a `float` o `double`.
- De `float` a `double`.

Una conversión primitiva de ampliación nunca pierde información sobre la magnitud general de un valor numérico.

Una conversión primitiva de ampliación de un tipo integral a otro tipo integral no pierde ninguna información en absoluto; el valor numérico se conserva exactamente.

En determinados casos, una conversión primitiva de ampliación de `float` a `double` puede perder información sobre la magnitud general del valor convertido.

Una conversión de ampliación de un valor `int` o `long` a `float`, o de un valor `long` a `double`, puede producir pérdida de precisión; es decir, el resultado puede perder algunos de los bits menos significativos del valor. En este caso, el valor de coma flotante resultante será una versión correctamente redondeada del valor entero.

Una conversión de ampliación de un valor entero con signo a un tipo integral simplemente extiende el signo de la representación del complemento a dos del valor entero para llenar el formato más amplio.

Una conversión de ampliación de `char` a un tipo integral rellena con ceros la representación del valor `char` para llenar el formato más amplio.

A pesar de que puede producirse una pérdida de precisión, una conversión primitiva de ampliación nunca da como resultado una excepción en tiempo de ejecución.

1.2.5.3. De restricción (*narrowing*)

1.2.6. Promociones numéricas

1.3. Tipos por referencia

1.3.1. Nulo

Existe un tipo especial llamado **tipo *nulo***.

El tipo nulo es el tipo de la expresión `null`, la cual representa la **referencia nula**.

La referencia nula es el único valor posible de una expresión de tipo nulo.

El tipo nulo no tiene nombre y, por tanto, no se puede declarar una variable de tipo nulo o convertir un valor al tipo nulo.

La referencia nula siempre puede asignarse o convertirse a cualquier tipo referencia.

En la práctica, el programador puede ignorar el tipo nulo y suponer que `null` es simplemente un literal especial que pertenece a cualquier tipo referencia.

2. Variables en Java

2.1. Variables de tipos primitivos

2.2. Variables de tipos por referencia

2.3. Declaraciones de variables

3. Estructuras de control

3.1. Bloques

3.2. `if`

3.3. `switch`

3.4. `while`

3.5. `for`

3.6. `do ... while`

4. Entrada/salida

4.1. Flujos `System.in`, `System.out` y `System.err`

4.2. `java.util.Scanner`