The Java Language Specification - Java SE 25 Edition

James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, Gavin Bierman

4. Tipos, Valores y Variables

El lenguaje de programación Java es un lenguaje de *tipado estático*, lo que significa que cada variable y cada expresión tiene un tipo conocido en tiempo de compilación.

Java también es un lenguaje de *tipado fuerte*, ya que los tipos limitan los valores que una variable puede contener o que una expresión puede producir, limitan las operaciones admitidas sobre 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 Java se dividen en dos tipos: **tipos primitivos** y **tipos de referencia**. Los tipos *primitivos* son el tipo `booleano` y los tipos numéricos. Los tipos numéricos son los tipos enteros `byte`, `short`, `int`, `long` y `char`, y los tipos de punto flotante `float` y `double`. Los tipos de referencia son los tipos de clase, los tipos de interfaz y los tipos de array. También existe un tipo especial nulo (*null*). Un objeto es una instancia creada dinámicamente de un tipo de clase o un array creado dinámicamente. Los valores de un tipo de referencia son referencias a objetos. Todos los objetos, incluidos los arreglos, admiten los métodos de la clase `Object`. Los literales de cadena se representan mediante objetos `String`.

4.1. Tipos de datos y valores

En el lenguaje de programación Java existen dos tipos de datos: **tipos primitivos** y **tipos de referencia**. En consecuencia, existen dos tipos de valores de datos que pueden almacenarse en variables, pasarse como argumentos, devolverse por métodos y sobre los que se puede operar: **valores primitivos** y **valores de referencia**.

Type:
  PrimitiveType
  ReferenceType

También existe un *tipo nulo* especial, el tipo de la expresión `null`, que no tiene nombre.

Debido a que el tipo nulo no tiene nombre, es imposible declarar una variable de este tipo o convertirla a él.

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

La referencia nula siempre se puede asignar o convertir a cualquier tipo de referencia.

> En la práctica, el programador puede ignorar el tipo `null` y simplemente considerar que `null` es un literal especial que puede ser de cualquier tipo de referencia.

4.2. Tipos y valores primitivos

Un tipo primitivo está predefinido por el lenguaje de programación Java y se nombra mediante su palabra clave reservada:

PrimitiveType:
  {Annotation} NumericType
  {Annotation} `boolean`

NumericType:
  IntegralType
  FloatingPointType

IntegralType:
  (one of)
  `byte short int long char`

FloatingPointType:
  (one of)
  `float double`

Los valores primitivos no comparten estado con otros valores primitivos.

Los *tipos numéricos* son los enteros y los de punto flotante.

Los *tipos enteros* son `byte`, `short`, `int` y `long`, cuyos valores son enteros con signo de 8, 16, 32 y 64 bits en complemento a dos, respectivamente; y `char`, cuyos valores son enteros sin signo de 16 bits que representan unidades de código UTF-16.

Los tipos de *punto flotante* son `float`, cuyos valores corresponden exactamente a los números de punto flotante de 32 bits del estándar IEEE 754 (binary32); y `double`, cuyos valores corresponden exactamente a los números de punto flotante de 64 bits del estándar IEEE 754 (binary64).

El tipo `boolean` tiene exactamente dos valores: `true` y `false`.

4.2.1. Tipos y valores enteros

Los valores de los tipos enteros son números enteros en los siguientes rangos:

- Para `byte`, de -128 a 127, ambos inclusive
- Para `short`, de -32768 a 32767, ambos inclusive
- Para `int`, de -2147483648 a 2147483647, ambos inclusive
- Para `long`, de -9223372036854775808 a 9223372036854775807, ambos inclusive
- Para `char`, de `\u0000` a `\uffff`, ambos inclusive, es decir, de 0 a 65535

4.2.2. Operaciones con enteros

El lenguaje de programación Java proporciona varios operadores que actúan sobre valores enteros:

- Los operadores de comparación, que dan como resultado un valor `boolean`:
  - Los operadores de comparación numérica <, <=, >y >=
  - Los operadores de igualdad numérica == y !=
- Los operadores numéricos, que dan como resultado un valor `int` o `long`:
  - Los operadores unarios de suma y resta + y -
  - Los operadores multiplicativos *, / y%
  - Los operadores aditivos + y -
  - El operador de incremento ++, tanto prefijo como sufijo
  - El operador de decremento –, tanto prefijo como sufijo
  - Los operadores de desplazamiento con signo y sin signo ´, ˇ y ˇ >
  - El operador de complemento bit a bit
  - Los operadores bit a bit enteros &, ~ y ||
- El operador condicional `?`
- El operador de conversión, que permite convertir un valor entero a un valor de cualquier tipo numérico especificado.
- El operador de concatenación de cadenas +, que, al recibir un operando de tipo `String` y un operando entero, convierte el operando entero a `String` (la forma decimal de un operando `byte`, `short`, `int` o `long`, o el carácter de un operando `char`) y, a continuación, genera una nueva cadena que resulta de la concatenación de ambas.

Otros constructores, métodos y constantes útiles están predefinidos en las clases `Byte`, `Short`, `Integer`, `Long` y `Character`.

Si un operador entero, distinto de un operador de desplazamiento, tiene al menos un operando de tipo `long`, la operación se realiza con precisión de 64 bits y el resultado del operador numérico es de tipo `long`. Si el otro operando no es de tipo `long`, se amplía a `long` mediante promoción numérica.

En caso contrario, la operación se realiza con precisión de 32 bits y el resultado del operador numérico es de tipo `int`. Si alguno de los operandos no es de tipo `int`, se amplía a `int` mediante promoción numérica.

Los operadores enteros no indican desbordamiento ni subdesbordamiento.

Cualquier valor de cualquier tipo entero puede convertirse a o desde cualquier tipo numérico. No existen conversiones entre tipos enteros y el tipo `boolean`.

> Consulte la sección 4.2.5 para ver un método para convertir expresiones enteras a booleano.

Un operador de enteros puede lanzar una excepción (ğ11 (Excepciones)) por los siguientes motivos:

- Cualquier operador de enteros puede lanzar una `NullPointerException` si se requiere la conversión de desempaquetado de una referencia nula.
- El operador de división de enteros `/` y el operador de resto de enteros `%` pueden lanzar una `ArithmeticException` si el operando derecho es cero.
- Los operadores de incremento y decremento ++ y – pueden lanzar un `OutOfMemoryError` si se requiere la conversión de empaquetado y no hay suficiente memoria disponible para realizarla.

Ejemplo 4.2.2-1. Operaciones con enteros

```
class Test {
    public static void main(String[] args) {
        int i = 1000000;
        System.out.println(i * i);
        long l = i;
        System.out.println(l * l);
        System.out.println(20296 / (l - i));
    }
}
```

Este programa produce la siguiente salida:

```
-727379968
1000000000000
```

Luego se produce una excepción `ArithmeticException` en la división por `l - i`, ya que `l - i` es cero. La primera multiplicación se realiza con precisión de 32 bits, mientras que la segunda es una multiplicación de tipo

long. El valor `-727379968` es el valor decimal de los 32 bits menos significativos del resultado matemático, `1000000000000`, un valor demasiado grande para el tipo `int`.

4.2.3 Tipos y valores de punto flotante

Los tipos de punto flotante son `float` y `double`, que se asocian conceptualmente con los formatos de punto flotante binary32 de 32 bits y binary64 de 64 bits para valores y operaciones IEEE 754, según se especifica en el estándar IEEE 754.

> En Java SE 15 y versiones posteriores, el lenguaje de programación Java utiliza la versión 2019 del estándar IEEE 754. Antes de Java SE 15, el lenguaje de programación Java utilizaba la versión de 1985 del estándar IEEE 754, donde el formato binario32 se conocía como formato simple y el formato binario64 como formato doble.

El estándar IEEE 754 incluye no solo números positivos y negativos con signo y magnitud, sino también ceros positivos y negativos, *infinitos* positivos y negativos, y valores especiales que *no son números* (en adelante, `NaN`, *Not-a-Number*). Un valor NaN se utiliza para representar el resultado de ciertas operaciones no válidas, como la división de cero entre cero. Las constantes NaN de tipo `float` y `double` están predefinidas como `Float.NaN` y `Double.NaN`.

Los valores finitos no nulos de un tipo de punto flotante se pueden expresar en la forma $s \cdot m \cdot 2^{(e-N+1)}$, donde:

- *s* es +1 o -1,
- *m* es un entero positivo menor que 2N,
- *e* es un entero entre $E_{min} = -(2^{K-1} - 2)$ y $E_{max} = 2^{K-1} - 1$, ambos inclusive, y
- *N* y *K* son parámetros que dependen del tipo.

Algunos valores se pueden representar de esta forma de más de una manera. Por ejemplo, suponiendo que un valor `v` de un tipo de punto flotante se pueda representar de esta forma usando ciertos valores para *s*, *m* y *e*, entonces si *m* fuera par y e menor que 2^{K-1}, se podría dividir *m* por la mitad y aumentar *e* en 1 para obtener una segunda representación para el mismo valor *v*.

Una representación de esta forma se denomina normalizada si $m \geq 2^{N-1}$; de lo contrario, se dice que la representación es *subnormal*. Si un valor de tipo punto flotante no se puede representar de tal manera que $m \geq 2^{N-1}$, entonces

se dice que el valor es un valor subnormal, porque su magnitud es inferior a la magnitud del valor normalizado más pequeño.

Las restricciones sobre los parámetros *N* y *K* (y sobre los parámetros derivados E_{min} y E_{max}) para los tipos `float` y `double` se resumen en la Tabla 4.2.3-A.

Tabla 4.2.3-A. Parámetros de punto flotante

Parámetro	float	double
N	24	53
K	8	11
E_{max}	+127	+1023
E_{min}	-126	-1022

Excepto para NaN, los valores de punto flotante están *ordenados*. De menor a mayor, son: infinito negativo, valores negativos finitos distintos de cero, cero negativo y positivo, valores positivos finitos distintos de cero e infinito positivo.

El estándar IEEE 754 permite múltiples valores NaN distintos para cada uno de sus formatos de punto flotante binary32 y binary64. Sin embargo, la plataforma Java SE generalmente trata los valores NaN de un tipo de punto flotante dado como si estuvieran comprimidos en un único valor canónico, por lo que esta especificación normalmente se refiere a un NaN arbitrario como si fuera un valor canónico.

> Según IEEE 754, una operación de punto flotante con argumentos que no son NaN puede generar un resultado NaN. IEEE 754 especifica un conjunto de patrones de bits NaN, pero no impone qué patrón de bits NaN en particular se utiliza para representar un resultado NaN; esto queda a criterio de la arquitectura del hardware. Un programador puede crear NaN con diferentes patrones de bits para codificar, por ejemplo, información de diagnóstico retrospectiva. Estos valores NaN se pueden generar con los métodos `Float.intBitsToFloat` y `Double.longBitsToDouble` para `float` y `double`, respectivamente. Por el contrario, para inspeccionar los patrones de bits de los valores NaN, se pueden usar los métodos *Float.floatToRawIntBits* y `Double.doubleToRawLongBits` para `float` y `double`, respectivamente.

El cero positivo y el cero negativo se comparan de forma equivalente, por lo que el resultado de la expresión `0.0 == -0.0` es verdadero y el resultado de `0.0 >-0.0` es falso. Otras operaciones pueden distinguir entre cero positivo y negativo; por ejemplo, `1.0/0.0` tiene el valor de infinito positivo, mientras que el valor de `1.0/-0.0` es infinito negativo.

NaN *no tiene orden*, por lo que:

- Los operadores de comparación numérica <, <=, >y >= devuelven `false` si uno o ambos operandos son NaN.

  En particular, ($x < y$) == !($x >= y$) será `false` si `x` o `y` son NaN.

- El operador de igualdad $==$ devuelve `false` si alguno de los operandos es NaN.

- El operador de desigualdad $!=$ devuelve `true` si alguno de los operandos es NaN.

  En particular, $x != x$ es `true` si y solo si `x` es NaN.

4.2.4 Operaciones de punto flotante

El lenguaje de programación Java proporciona varios operadores que actúan sobre valores de punto flotante:

- Los operadores de comparación, que dan como resultado un valor `boolean`:
  - Los operadores de comparación numérica <, <=, >y >=
  - Los operadores de igualdad numérica $==$ y $!=$
- Los operadores numéricos, que dan como resultado un valor de tipo `float` o `double`:
  - Los operadores unarios de suma y resta + y -
  - Los operadores multiplicativos *, / y%
  - Los operadores aditivos + y -
  - El operador de incremento $++$, tanto prefijo como sufijo
  - El operador de decremento $--$, tanto prefijo como sufijo
- El operador condicional ? : y el operador posfijo
- El operador de conversión, que permite convertir un valor de punto flotante a un valor de cualquier tipo numérico especificado.
- El operador de concatenación de cadenas +, que, al recibir un operando de tipo `String` y un operando de punto flotante, convierte este último a un `String` que representa su valor en forma decimal (sin pérdida de información) y, a continuación, genera una nueva `String` mediante la concatenación de ambas.

Otros constructores, métodos y constantes útiles están predefinidos en las clases `Float`, `Double` y `Math`.

Si al menos uno de los operandos de un operador binario es de tipo punto flotante, la operación se considera de punto flotante, incluso si el otro operando es entero.

Si al menos uno de los operandos de un operador numérico es de tipo `double`, la operación se realiza mediante aritmética de punto flotante de 64 bits, y el resultado es un valor de tipo `double`. Si el otro operando no es de tipo `double`, se amplía previamente a este tipo mediante promoción numérica.

En caso contrario, si al menos uno de los operandos es de tipo `float`, la operación se realiza mediante aritmética de punto flotante de 32 bits, y el resultado es un valor de tipo `float`. Si el otro operando no es de tipo `float`, se amplía previamente a este tipo mediante promoción numérica.

La aritmética de punto flotante se realiza de acuerdo con las reglas del estándar IEEE 754, incluyendo el desbordamiento y el subdesbordamiento, con la excepción del operador de resto %.

Cualquier valor de tipo de punto flotante se puede convertir a cualquier tipo numérico o viceversa. No existen conversiones entre tipos de punto flotante y el tipo `boolean`.

> Consulte la sección 4.2.5 para ver un método para convertir expresiones de punto flotante a `boolean`.

Un operador de punto flotante puede generar una excepción (sección 11 (Excepciones)) por los siguientes motivos:

- Cualquier operador de punto flotante puede generar una `NullPointerException` si se requiere la conversión de desempaquetado (sección 5.1.8) de una referencia nula.
- Los operadores de incremento y decremento ++ (secciones 15.14.2 y 15.15.1) y – (secciones 15.14.3 y 15.15.2) pueden generar un `OutOfMemoryError` si se requiere la conversión de empaquetado (sección 5.1.7) y no hay suficiente memoria disponible para realizarla.

Ejemplo 4.2.4-1. Operaciones de punto flotante.

```
class Test {
    public static void main(String[] args) {
        // An example of overflow:
        double d = 1e308;
        System.out.print("overflow produces infinity: ");
        System.out.println(d + "*10==" + d*10);
        // An example of gradual underflow:
        d = 1e-305 * Math.PI;
        System.out.print("gradual underflow: " + d + "\n   ");
        for (int i = 0; i < 4; i++)
            System.out.print(" " + (d /= 100000));
        System.out.println();
        // An example of NaN:
        System.out.print("0.0/0.0 is Not-a-Number: ");
        d = 0.0/0.0;
        System.out.println(d);
        // An example of inexact results and rounding:
        System.out.print("inexact results with float:");
        for (int i = 0; i < 100; i++) {
            float z = 1.0f / i;
            if (z * i != 1.0f)
                System.out.print(" " + i);
        }
        System.out.println();
        // Another example of inexact results and rounding:
        System.out.print("inexact results with double:");
        for (int i = 0; i < 100; i++) {
            double z = 1.0 / i;
            if (z * i != 1.0)
                System.out.print(" " + i);
        }
        System.out.println();
        // An example of cast to integer rounding:
        System.out.print("cast to int rounds toward 0: ");
        d = 12345.6;
        System.out.println((int)d + " " + (int)(-d));
    }
}
```

Este programa produce la siguiente salida:

```
overflow produces infinity: 1.0E308*10==Infinity
gradual underflow: 3.141592653589793E-305
    3.1415926535898E-310 3.141592653E-315 3.142E-320 0.0
0.0/0.0 is Not-a-Number: NaN
inexact results with float: 0 41 47 55 61 82 83 94 97
inexact results with double: 0 49 98
cast to int rounds toward 0: 12345 -12345
```

Este ejemplo demuestra, entre otras cosas, que un desbordamiento negativo gradual puede resultar en una pérdida gradual de precisión.

Cuando `i` es `0`, los resultados implican una división por cero, por lo que `z` se convierte en infinito positivo, y `z * 0` es NaN, que no es igual a `1.0`.

4.2.5 El tipo `boolean` y los valores booleanos

El tipo `boolean` representa una cantidad lógica con dos valores posibles, indicados por los literales `true` y `false`.

Los operadores booleanos son:

- Los operadores relacionales == y !=
- El operador de complemento lógico !

- Los operadores lógicos &, ~ y |
- Los operadores condicionales AND && y OR ||||
- El operador condicional `?`
- El operador de concatenación de cadenas `+`, que, al recibir un operando `String` y un operando `boolean`, convierte el operando `boolean` a una cadena (`true` o `false`) y genera una nueva `String` resultante de la concatenación de ambas.

Las expresiones booleanas determinan el flujo de control en varios tipos de sentencias:

- La sentencia `if`
- La sentencia `while`
- La sentencia `do`
- La sentencia `for`

Una expresión `boolean` también determina qué subexpresión se evalúa en el operador condicional `? :`.

Solo se pueden usar expresiones `boolean` en las sentencias de control de flujo y como primer operando del operador condicional `? :`.

Una expresión entera o de punto flotante `x` se puede convertir a un valor `boolean`, siguiendo la convención del lenguaje C de que cualquier valor distinto de cero es verdadero, mediante la expresión `x!=0`.

Una referencia a un objeto `obj` se puede convertir a un valor `boolean`, siguiendo la convención del lenguaje C de que cualquier referencia distinta de `null` es `true`, mediante la expresión `obj!=null`.

Un valor `boolean` se puede convertir a un `String` mediante conversión de cadena.

Un valor `boolean` se puede convertir a los tipos `boolean`, `Boolean` u `Object`. No se permiten otras conversiones al tipo `boolean`.