Tema 2: Los tipos de datos

Antonio J. Sierra

Índice

Introducción

- Java es un lenguaje **fuertemente tipado** (es parte de la seguridad y robustez).
- Cada **variable** tiene un **tipo**, cada **expresión** tiene un **tipo** y cada **tipo** está definido estrictamente.
- TODAS las asignaciones, bien sean explícitas o a través de paso de parámetros en llamadas a método, **se comprueba la compatibilidad de tipos** (Se comprueban las expresiones y parámetros asegurando la compatibilidad tipos).
- No hay conversión automática entre tipos en la que haya un conflicto.

Los tipos simples

- Los tipos son **byte, short, int long float, double, char y boolean. Cuatro grupos:**
  1. **Enteros**: En este grupo están los tipos **bytes, short, int** y **long**, que son para números con signo de valor completo.
  2. **Números en coma flotante**: Este grupo incluye los tipos **float**, y **double**, que representan números con precisión fraccionaria.
  3. **Caracteres**: corresponde al tipo **char**, que representa símbolos de un conjunto de caracteres como letras y números.
  4. **Boolean**. Es el tipo **boolean**, que es un tipo especial para valores lógicos

Enteros

- **bytes, short, int** y **long**.
- Todos estos tipos tienen signo.
- Java gestiona el significado del **bit más significativo** de forma diferente, añadiendo un nuevo operador llamado "*desplazamiento a la derecha sin signo".* Con este operador, desaparece la necesidad de tener un tipo entero sin signo.
- El ***tamaño*** de un entero se debería ver como el **comportamiento** que define para las variables y expresiones de ese tipo.

int

- 32 bits con signo
- El tipo entero más utilizado habitualmente:
  - para realizar el control de bucles
  - para indexar matrices
- Es un tipo cuyo rango comprende:

  –2.147.483.648 a 2.147.483.647
- Siempre que tenga una expresión con enteros que incluya **byte, short**, **int** y números literales, la expresión completa **promociona** a **int** antes de realizar el cálculo.

Java usa el complemento a2

- Los enteros *informáticos* se diferencian de los *matemáticos* en que tienen un valor mínimo y otro máximo. El entero más grande tiene un cero colocado en la posición que se encuentra situada a la izquierda del todo y el resto de posiciones son unos
  - 0111 1111 1111 1111 1111 1111 1111 1111, o **2,147,483,647** en decimal.

  Si le añadimos uno, obtendremos el

  1000 0000 0000 0000 0000 0000 0000 0000

  que es el número entero más pequeño en Java, el **-2,147,483,647.**

  0111 1111 1111 1111 1111 1111 1111 1111

byte

- El tipo entero más pequeño es el **byte**.
- 8 bits con signo
- Rango:

  desde –**128** (0111 1111) a **12**7(1000 0000).
- Los bytes son el mínimo común denominador existente en la transferencia de datos que se realiza entre ordenadores, y **Java los utiliza para las entradas como para las salidas**.
- Las variables del tipo **byte** son especialmente útiles cuando estamos trabajando con un flujo de datos recibido desde una red o un archivo.
- También pueden ser útiles cuando queremos trabajar con datos binarios que pueden no ser directamente compatibles con otros tipos de Java.
- Las variables de tipo byte se declaran utilizando la palabra clave **byte**. Por ejemplo, el código siguiente declara (define) dos variables **byte** llamadas **b** y **c**. **byte b,c;**
- **NO** se utilizan para los cálculos aritméticos. El compilador de Java no nos permite escribir:

  byte b3 = b1+b2;

short

- **short** es un tipo de 16 bits con signo. Su rango comprende desde –32768 (1000 0000 0000 0000) a 32767 (0111 1111 1111 1111).
- Probablemente es el tipo de Java menos utilizado: su byte más significativo primero (formato **big-endian**).
- No se puede utilizar un **short** en una operación aritmética.
- Al igual que pasaba con los **bytes**, si escribimos:
- **short s3 = 454 + 732;**

long

- 64 bits con signo
- Enteros con mayor rango:

[-9,223,372,036,854,775,808 9,223,372,036,854,775,807]

- Si se utilizan para hacer operaciones aritméticas.
- Para indicar que un número es **long**, bastará con utilizar una **L** (o **l**) como sufijo (por ejemplo, 2147483856L o –76l). Al igual que los enteros se pueden escribir en hexadecimal o octal (por ejemplo 0xCAFEBABEL o 0714L)
- **Nota**: La mayoría de los programadores prefieren la **L** para no confundir la letra **l** con un **1**.

Números en coma flotante

- Números con precisión fraccionaria:
  - **float** y
  - **double**

float

- Para **float**: el primer bit de **signo**, los 8 siguientes de **exponente**, los 23 siguientes de **mantisa**.
- La **mantisa** es un número entre 1 y 2; se parte de que el primer número es un uno seguido de la coma, por lo tanto tendrá una precisión de 24 bits.
- El **exponente** consta de 8 bits. Se considera que es un entero entre 0 y 255 carente de signo.
- 0 y 255 tienen un significado especial.
- Para calcular el exponente, se restar 127. Estarán comprendidos entre [-126 y +127], que corresponde con (1-127) y (254-127).

double

- Para **double**: **8 bytes**, de los cuales,
  - El primer **bit** se utiliza para el signo.
  - Los 11 siguientes para el exponente y
  - Los 53 restantes para la mantisa.
- Sumando en total salen 65 bits. Es porque el primer bit de la mantisa siempre es 1 y no se pone.
- Para calcular el exponente bastará con que restemos 1023.
- La precisión doble es más rápida que la simple. Las funciones matemáticas suelen devolver **double** (**sin(), cos() y sqrt()**).

Valores especiales

```
java.lang.Float

public static final float MAX_VALUE              3.4028234663852886E38f
public static final float MIN_VALUE              1.401298464324817E-45f
public static final float NaN                    0f/0f
public static final float NEGATIVE_INFINITY      -1f/0f
public static final float POSITIVE_INFINITY      1f/0f



java.lang.Double

public static final double MAX_VALUE             1.7976931348623157E308d
public static final double MIN_VALUE             4.9E-324d
public static final double NaN                   0d/0d
public static final double NEGATIVE_INFINITY     -1d/0d
public static final double POSITIVE_INFINITY     1d/0d
```

Resumen de los tipos numéricos

Tipo	Formato	Rango	Descripción
int	32 bits complemento a 2	[-2.147.483.648, 2.147.483.647]	Entero
byte	8 bits complemento a 2	[-128, 127]	Entero de un byte
short	16 bits	[-32768, 32767]	Big-endian
long	64 bits	[-9,223,372,036,854,775,808 y 9,223,372,036,854,775,807].	Enteros largos

Tipo	Formato	Rango
double	64 bits (8 bytes)	[-1.7976931348623157E308 a -4.9E-324] negativos [4.9E-324 a 1.7976931348623157E308] positivos
float	32 bits (4 bytes)	[-3.4028234663852886E38 a -1.401298464324817E-45] negativos [1.401298464324817E-45 a 3.4028234663852886E38] positivos

Caracteres

- Los **char** tiene una longitud de dos bytes (ocupan lo mismo que un **short**, no son lo mismo).
  - Los **short** tienen signo y los **char** carecen de él.
  - El primer bit en un **char** no tiene nada que ver con el signo.
- Así, mientras que si interpetamos
- 1000 0000 0000 0001 como un **short** tendremos -32,768, como un **char** tendremos el 32,769.
- Rango de **`char`** entre el 0 y el 65,535.

Ejemplos de char

char c = 114;

char d = 45000;

Las próximas dos líneas hacen que la compilación provoque un **ERROR :**

char e = -123;

char f = 65536;

Unicode y char

- **ASCII** (American Standard Code for Information Interchange):Contiene, las minúsculas a-z, las mayúsculas A-Z, los dígitos 0-9, varios signos de puntuación y numerosos caracteres de control. Numerados del 0 al 127.
- ISO Latin-1: 8 bits.
- Unicode: rango mayor.

UTF-8 (I)

- Cualquier **byte** que comience con un bit 0 es un carácter de 1 byte de ASCII.
- Cualquier **byte** que comience por 110 es un carácter de 2 byte.
- Cualquiera que comience por 1110 es un carácter de 3-byte.
- Por último, cualquier **byte** que comience por 10 será el segundo de un carácter multi-byte

UTF-8 (y II)

- **Caracteres entre el 1 y el 127 (\u0001 y \u007F)**, menos el carácter nulo, en un byte.
- **Caracteres entre el 127 y el 28,927 (\u0080 a \u07FF),,** tiene 11bit de datos, en dos bytes:

  1 1 0 x x x x x 1 0 x x x x x x

  6-10 bits 0-5 bits
- **Carácter nulo** dos bytes: 1100 0000 1000 0000
- Los caracteres entre el \u0800 y el \uFFFF tienen 16 bits de datos en tres bytes:

  1 1 1 0 x x x x 1 0 x xx x x x 1 0 x x x x x x

  12-15 bits 6-11 bits 0-5 bits

boolean

- Para valores lógicos: sólo puede tomar uno entre dos posibles valores, **true** (verdadero) o **false** (falso).
- Es el tipo que devuelven todos los operadores de comparación, como **a<b**.
- Es el tipo que se requiere en las expresiones condicionales que controlan las sentencias de control de flujo como **if** y **for**.
- En el código fuente en Java no se escriben 0 ó 1. Tampoco como "verdadero " o "falso".

Literales

- Un **valor constante** en Java se crea utilizando una representación **literal** de él.
- **Enteros**.
- **Números en coma flotante**.
- **Boolean**.
- **Caracteres y tipo cadena**.

Literales enteros

- Los enteros (**int**) son el tipo más utilizado habitualmente en los programas. es un valor de **32 bits**. Ejemplo: **1, 2, 3** y **12** (base 10) .
- Hay otras bases:
  - **OCTAL** (precedidos de cero: 09 da error)
  - **HEXADECIMAL** (va precedido de **0x** o bien **0X**).
- Se pueden asignar a una variable **byte** o **short**, no se genera ningún error si el valor literal está dentro del rango del tipo destino**.**
- Se puede asignar un literal **entero** a una variable **long**, de forma explícita con L (o l). Ejemplo, 0x7fff ffff ffff ffffL o 9223372036854775807L

Literales en coma flotante

- Valores decimales con una componente fraccional.
- **Notación estándar**: unos dígitos decimales, seguidos por un punto decimal (.), y a continuación unos cuantos dígitos fraccionales. Ejemplo: 2.0, 3.14159 y .6667, 3.0f
- **Notación científica:** un número en coma flotante con una notación estándar con una notación adicional que indica una multiplicación por diez elevado al exponente especificado. El exponente se indica mediante una **E** o **e** seguido de un número decimal, que puede ser positivo o negativo. Ejemplo. 6.022E23, 314E-05, 2e+10f.

Literales booleanos

- Los literales booleanos:
  - Un valor booleano sólo puede tener dos valores lógicos, **true** (verdadero)
  - **y false** (falso). Los valores **true** y **false**
- No se convierten en ninguna representación numérica. El literal **true** en Java no es igual a 1 ni el literal **false** es igual a 0.
- En Java, sólo se pueden asignar esto valores a variables declaradas como **boolean**, o ser utilizados en expresiones con operadores booleanos.

Literales tipo carácter

- Los caracteres en Java son valores de Unicode.
- 16 bits manipulables con operadores enteros, como la suma y la resta.
- Carácter literal con un par de comillas simples (' '). Por ejemplo los ASCII 'a','z'…
- Para los caracteres que no se puedan introducir directamente,
  - hay varias secuencias de escape
  - También hay un mecanismo para introducir directamente el valor de un carácter en octal ('\141') o hexadecimal ('\u0061').

Secuencias de escape

\ddd	Carácter octal (ddd)
\uxxxx	Carácter UNICODE hexadecimal (xxxx)
\'	Comilla simple
\"	Comilla doble
\r	Retorno de carro.
\n	Nueva línea (o salto de línea)
\f	Alimentación de página
\t	Tabulador
\b	Retroceso

Variables

- Es la unidad básica de almacenamiento.
- Se define mediante la combinación de un identificador, un tipo y un inicializador opcional.
- Las variables tienen un ámbito, que define su visibilidad y su tiempo de vida.
- Todas las variables deben ser **declaradas** antes de que puedan ser utilizadas.

Tipo identificador [= valor][, identificador [=valor]…]

Conversión de tipos

- Se puede asignar un valor de un tipo a una variable de otro tipo. Si los dos tipos son **compatibles**, se realizará la conversión automáticamente. Por ejemplo, siempre es posible asignar un valor **int** a una variable **long**.
- No todos los tipos son compatibles y, por tanto, no todas las conversiones de tipo se realizarán **implícitamente**.
- Es posible realizar conversión entre tipos incompatibles.
- Para hacer esto se debe usar un **cast**, o conversión explícita entre tipos incompatibles.
- Hay dos tipo de conversiones de tipo:
  - **automáticas** y
  - las **explícitas.**

Conversión automática

- Cuando se asigna un tipo de dato a otro tipo de variable, se realiza una **conversión de tipo automática** si se producen la condiciones siguientes:
  - **Los dos tipos son compatibles.**
  - **El tipo destino es más grande que el tipo origen**.
- Cuando se cumplen estas dos condiciones se produce un **ensanchamiento** o **promoción**.
- El tipo **`boolean`** no es compatible con el resto.

Conversión de tipos incompatibles

- Se usa un **cast**. Un **cast** es simplemente una conversión de tipo explícita y tiene la siguiente forma:

  (tipo) valor;

- **tipo** indica el tipo al que se ha de convertir el valor especificado. Por ejemplo, el siguiente fragmento de código convierte un **int** a **byte**. Si el valor del entero es más grande que el rango de un **byte**, se reducirá al módulo (resto de la división entera) del rango del tipo **byte**.

```
int a;
byte b;
//…
b = (byte) a;
```

- Cuando se asigna un valor en **coma flotante** a **un tipo entero** se produce una conversión de tipo diferente **truncando** la componente fraccional. Se pierde la componente fraccional.
- Por ejemplo, si se asigna el valor 1.23 a un entero el valor resultante será simplemente 1. El 0.23 habrá sido truncado.
- Por supuesto, si el tamaño de la componente numérica es demasiado grande para caber en el tipo entero destino, entonces ese valor será reducido al módulo del rango del tipo destino.

Promoción de tipo automática en expresiones

- Reglas de promoción:
  - En primer lugar, como ya hemos descrito, todos los valores **byte** y **short** son promocionadas a **int**.
  - Además, si un operador es de tipo **long**, la expresión completa se promociona a **long**.
  - Si un operado es de tipo **float** la expresión completa se promociona a **float.**
  - Si alguno de los operandos es **double**, el resultado es **double**.