



4. Conversiones de tipo

La conversión de tipo (en inglés, *casting*) es un procedimiento para transformar una variable primitiva de un tipo a otro. Esta operación es muy usual cuando queremos convertir valores decimales a enteros o viceversa.

También se utiliza para transformar un objeto de una clase a otra distinta, siempre y cuando haya una relación de herencia entre ambas. Esto se verá en unidades posteriores.

Dentro la conversión de tipo de variables primitivas se distinguen dos clases:

- Casting implícito
- Casting explícito

Las conversiones de tipo se realizan para conseguir que el resultado de una expresión sea del tipo que nosotros deseamos. Esto dependerá de lo que necesitemos en cada momento.

4.1. Casting implícito o automático

Cuando a una variable de un tipo se le asigna un valor de otro tipo numérico con menos bits para su representación, se realiza una conversión automática. En ese caso, el valor se dice que es promocionado al tipo más grande (el de la variable) para poder hacer la asignación. En la tabla de tipos de datos básicos se muestra el tamaño –tanto en bits como en bytes– de todos estos tipos de datos. Este sería el orden del más pequeño al más grande:

byte -> short -> char -> int -> long -> float -> double

También se realizan conversiones automáticas en las operaciones aritméticas. Cuando estamos utilizando valores de distinto tipo, el valor de tipo más pequeño se promociona al valor de tipo más grande, ya que el tipo mayor siempre podrá representar cualquier valor del tipo menor (por ejemplo, en la conversión de números enteros de int a long o de números científicos de float a double).

En este caso no se necesita escribir código para que la conversión se lleve a cabo. Ocurre cuando se realiza lo que se llama una **conversión ancha** (en inglés *widening casting*), es decir, cuando se coloca un valor pequeño en un contenedor grande.

Ejemplo:

```
int numeroEntero = 100;
long numeroEnteroLargo = numeroEntero;
// Un int de 4 bytes "cabe" en un long de 8 bytes.
```

4.2. Casting explícito

Cuando hacemos una conversión de un tipo con más bits a un tipo con menos bits. En estos casos debemos indicar que queremos hacer la conversión de manera expresa, ya que se puede producir una pérdida de datos y hemos de ser conscientes de ello. Este tipo de conversiones se realiza con el operador `cast`.



El operador **cast** es un operador unario que se forma colocando delante del valor a convertir el tipo de dato entre paréntesis. Tiene la misma precedencia que el resto de operadores unarios y se asocia de izquierda a derecha. El formato general para indicar que queremos realizar la conversión es:

(tipo_dato_destino) valor_a_convertir

En el casting explícito sí es necesario escribir código. Ocurre cuando se realiza una **conversión estrecha** (en inglés, *narrowing casting*), es decir, cuando se coloca un valor grande en un contenedor pequeño. El orden del más grande al más pequeño sería este:

double -> float -> long -> int -> char -> short -> byte

Hay que decidir qué hacemos con lo que no cabe en el tipo de dato más pequeño.

```
int numeroEntero = 100;
    /* Casting explícito: de int a short que tiene menor rango de
valores */
    short numeroEnteroCorto = (short) numeroEntero ;
```

Debemos tener en cuenta que un valor numérico nunca puede ser asignado a una variable de un tipo menor en rango, salvo que se lleve a cabo una conversión explícita.

Ejemplo:

```
int numeroEntero;
byte numeroEnteroPequeño;

/* no se realiza conversión alguna, es una asignación directa de un valor
a un int */
numeroEntero = 12;
/* se permite porque 12 está dentro del rango permitido de valores para
el tipo byte */
numeroEnteroPequeño = 12;
/* ERROR, no está permitida esta asignación (incluso aunque 12 podría
almacenarse en un byte) ya que son tipos de datos diferentes */
numeroEnteroPequeño = numeroEntero;
/* Correcto, forzamos conversión explícita */
byte numeroEnteroPequeño = (byte) numeroEntero;
```

En el ejemplo anterior podemos ver un caso típico de error de tipos, ya que estamos intentando asignarle a `numeroEnteroPequeño` el valor de `numeroEntero`, siendo `numeroEnteroPequeño` de un tipo más pequeño. Lo correcto es promocionar al tipo de datos `byte`, y entonces asignarle su valor a la variable `numeroEnteroPequeño`.



		Tipo destino								
		boolean	char	byte	short	int	long	float	double	
Tipo origen	boolean	-	N	N	N	N	N	N	N	
	char	N	-	C	C	CI	CI	CI	CI	
	byte	N	C	-	CI	CI	CI	CI	CI	
	short	N	C	C	-	CI	CI	CI	CI	
	int	N	C	C	C	-	CI	CI*	CI	
	long	N	C	C	C	C	-	CI*	CI*	
	float	N	C	C	C	C	C	-	CI	
	double	N	C	C	C	C	C	C	-	

Tabla 2.1. Tabla de conversión de tipos de datos primitivos

N: conversión no permitida (un boolean no se puede convertir a ningún otro tipo y viceversa).

CI: conversión implícita o automática.

CI*: conversión implícita o automática. Puede haber pérdida de datos.

C: casting de tipos o conversión explícita.

4.3. Reglas de promoción de tipos de datos

Cuando en una expresión hay datos o variables de distinto tipo, el compilador realiza la promoción de unos tipos en otros para obtener como resultado el tipo final de la expresión. Esta promoción de tipos se hace siguiendo unas reglas básicas a partir de las cuales se realiza esta promoción de tipos. De manera sucinta son las siguientes:

- Si uno de los operandos es de tipo double, el otro se convierte a double.
- En cualquier otro caso:
 - Si uno de los operandos es float, el otro se convierte a float.
 - Si uno de los operandos es long, el otro se convierte a long.
 - Si no se cumple ninguna de las condiciones anteriores, entonces ambos operandos se convierten al tipo int.

4.4. Conversión de números en coma flotante (float, double) a enteros (int)

Cuando convertimos números en coma flotante a números enteros, la parte decimal se trunca (redondeo a cero). Si queremos hacer otro tipo de redondeo, podemos utilizar, entre otras, las siguientes funciones:

`Math.round(numeroCientifico)`: redondeo al número entero más cercano.

`Math.ceil(numeroCientifico)`: mínimo entero que sea mayor o igual al número decimal.

`Math.floor(numeroCientifico)` : entero mayor que sea inferior o igual a número decimal.



Veamos este ejemplo tanto con float como con double. Van a dar el mismo resultado:

```
double numeroCientificoLargo = 3.5;

int numeroEnteroX = Math.round(numeroCientificoLargo); // numeroEnteroX valdrá 4
int numeroEnteroY = Math.ceil(numeroCientificoLargo); // numeroEnteroY valdrá 4
int numeroEnteroZ = Math.floor(numeroCientificoLargo); // numeroEnteroZ valdrá 3
```

4.5. Conversiones entre caracteres (char) y enteros (int)

Como, en el caso del tipo char, lo que guarda en realidad es el código Unicode de un carácter, los caracteres pueden considerarse números enteros sin signo, ya que representan la posición numérica en la tabla de símbolos Unicode que ya hemos presentado en el módulo de Sistemas Informáticos.

Por ejemplo:

```
int numeroEntero;
char valorCaracter;

numeroEntero = (int) 'A'; // numeroEntero valdrá 65
valorCaracter = (char) 65; // valorCaracter valdrá 65 que
equivale al carácter 'A'
valorCaracter = (char) ((int) 'A' + 1);
// valorCaracter valdrá 66 que equivale al carácter 'B'
```

4.6. Conversiones de tipo con cadenas de caracteres (string)

Para convertir cadenas de texto a otros tipos de datos se utilizan métodos del estilo `parseATipoDeDatosDestino()`. Aquí podemos ver cómo sería para los 6 tipos de datos básicos numéricos. Por ejemplo, si tenemos un número 12 almacenado en una variable de tipo string llamada `cadenaConNumeros` y lo queremos convertir a cada uno de los diferentes tipos de datos numéricos, haríamos lo siguiente:

```
String cadenaConNumeros = "12";
byte numeroEnteroPequeño = Byte.parseByte(cadenaConNumeros);
short numeroEnteroCorto = Short.parseShort(cadenaConNumeros);
int numeroEntero = Integer.parseInt(cadenaConNumeros);
long numeroEnteroLargo = Long.parseLong(cadenaConNumeros);
float numeroCientifico = Float.parseFloat(cadenaConNumeros);
double numeroCientificoLargo = Double.parseDouble(cadenaConNumeros);
```

Aquí ya nos hacen falta las clases envoltorio (en inglés, *wrapper*) para convertir el valor de la variable en un objeto que, a su vez, pueda ejecutar sus métodos para realizar la conversión correspondiente. Igual que los tipos de datos, existen ocho envoltorios, uno por cada tipo. Estos son: Boolean, Byte, Short, Integer, Long, Character, Float y Double. Todos ellos presentan los métodos siguientes:

- `xxxValue()` //devuelve el valor en el tipo indicado en xxx
- `parseXxx()` // convierte el valor al tipo indicado en xxx y devuelve el primitivo nombrado.
- `valueOf()` // devuelve un nuevo objeto wrapped del mismo tipo que el invocado.
- `toString()` //devuelve una cadena con el valor del primitivo envuelto en el objeto.

