tipos de datos primitivos

Tipo de variable	Bytes que ocupa	Rango de valores
boolean	**2**	true, false
byte	**1**	**-128 a 127**
short	**2**	**-32.768 a 32.767**
int	**4**	**-2.147.483.648 a 2.147.483.649**
long	**8**	**$-9 \cdot 10^{18}$ a $9 \cdot 10^{18}$**
double	**8**	**$-1{,}79 \cdot 10^{308}$ a $1{,}79 \cdot 10^{308}$**
float	**4**	**$-3{,}4 \cdot 10^{38}$ a $3{,}4 \cdot 10^{38}$**
char	**2**	**Caracteres (en Unicode)**

enteros

Los tipos **byte, short, int** y **long** sirven para almacenar datos enteros. Los enteros son números sin decimales. Se pueden asignar enteros normales o enteros octales y hexadecimales. Los octales se indican anteponiendo un cero al número, los hexadecimales anteponiendo 0x.

```
int numero=16; //16 decimal
numero=020; //20 octal=16 decimal
numero=0x14; //10 hexadecimal=16 decimal
```

Normalmente un número literal se entiende que es de tipo **int** salvo si al final se le coloca la letra L; se entenderá entonces que es de tipo long.

No se acepta en general asignar variables de distinto tipo. Sí se pueden asignar valores de variables enteras a variables enteras de un tipo superior (por ejemplo asignar un valor **int** a una variable **long**). Pero al revés no se puede:

```
int i=12;
byte b=i;     //error de compilación
```

La solución es hacer un **cast**. Esta operación permite convertir valores de un tipo a otro. Se usa así:

```
int i=12;
byte b=(byte) i;   //No hay problema por el (cast)
```

conversión entre tipos *(casting)*

Hay veces en las que se deseará realizar algo como:

```
int a;byte b=12;
a=b;
```

La duda está en si esto se puede realizar. La respuesta es que sí. Sí porque un dato byte es más pequeño que uno int y Java le convertirá de forma implícita. Sin embargo en:

```
int a=1;
byte b;
b=a;
```

El compilador devolverá error aunque el número 1 sea válido para un dato byte. Para ello hay que hacer un *casting*. Eso significa poner el tipo deseado entre paréntesis delante de la expresión.

```
int a=1;
byte b;
b= (byte) a; //No da error
```

En el siguiente ejemplo:

```
byte n1=100, n2=100, n3;
n3= n1 * n2 /100;
```

Aunque el resultado es 100, y ese resultado es válido para un tipo byte; lo que ocurrirá en realidad es que ocurrirá un error. Eso es debido a que primero multiplica 100 * 100 y como eso da 10000, no tiene más remedio el compilador que pasarlo a entero y así quedará aunque se vuelva a dividir. La solución correcta sería:

```
n3 = (byte) (n1 * n2 / 100);
```

operadores aritméticos

Son:

operador	significado
+	Suma
-	Resta
*	Producto
/	División
%	Módulo (resto)

Hay que tener en cuenta que el resultado de estos operadores varía notablemente si usamos enteros o si usamos números de coma flotante.

Por ejemplo:

```
double resultado1, d1=14, d2=5;
int resultado2, i1=14, i2=5;

resultado1= d1 / d2;
resultado2= i1 / i2;
```

resultado1 valdrá 2.8 mientras que *resultado2* valdrá 2. Es más incluso:

```
double resultado;
int i1=7,i2=2;
resultado=i1/i2; //Resultado valdrá 3
resultado=(double)i1/(double)i2; //Resultado valdrá 3.5
```

El operador del módulo (%) para calcular el resto de una división entera. Ejemplo:

```
int resultado, i1=14, i2=5;

resultado = i1 % i2; //El resultado será 4
```