



# Objetos y clases

Los objetivos de esta unidad de trabajo son dos:

- introducir los conceptos de clase y objeto a través de un ejemplo
- aprender a trabajar con el entorno de desarrollo BlueJ

## 2.1.- Objetos y clases

### Concepto

Los objetos modelan objetos del dominio del problema

Cuando escribimos un programa en un lenguaje orientado a objetos estamos creando (en el ordenador) un modelo de alguna parte del mundo real. Las partes que constituyen el modelo son objetos que aparecen en el dominio del problema. Estos objetos deberán ser representados en el modelo que se construye de tal manera que pueda ser ejecutado en el ordenador.

Una **clase** es una abstracción que describe a un tipo particular de objetos. Por ejemplo, si queremos modelar una simulación de tráfico, trabajaremos con entidades tales como “coches”. ¿Qué es un coche en este contexto? ¿Una clase o un objeto?

Algunas cuestiones nos pueden ayudar a responder a esta pregunta: ¿De qué color es el coche? ¿Cuál es su velocidad? Estas cuestiones sólo podrán ser contestadas cuando hablemos de un coche particular. En nuestro contexto, “coche” se refiere a la clase coche, estamos hablando de coches en general, no de un coche en particular. Si decimos, “mi coche rojo alcanza una velocidad de 13 Km/h” estamos hablando de un coche en particular, el mío.

Una **clase** describe un **tipo de objeto**.

Un **objeto** representa instancias individuales de una clase. Habitualmente nos referiremos a un objeto particular como una **instancia** (una **instancia de una clase**). Instancia es sinónimo de objeto (“mi coche rojo es una instancia de la clase coche”).

### Concepto

Los objetos se crean a partir de las clases. La **clase** describe la categoría del objeto. El **objeto** es la representación detallada y particular de algo de la realidad

## 2.2.- Creando objetos

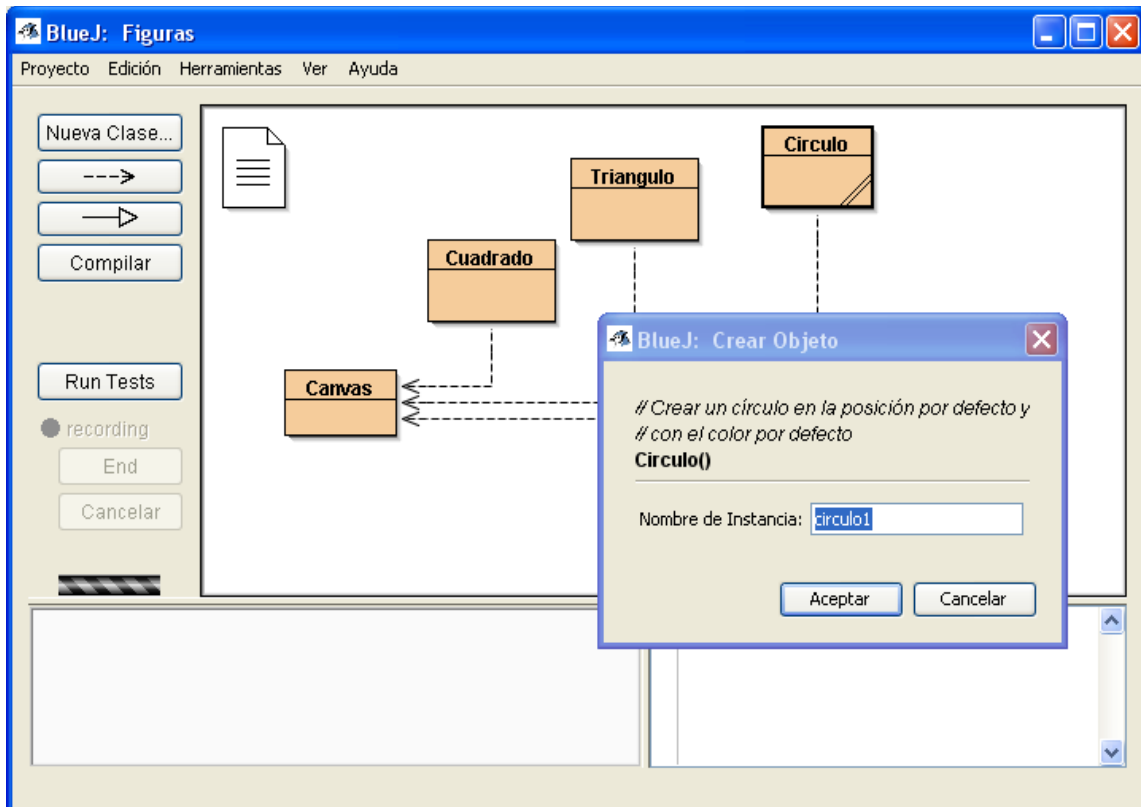
Trabajaremos en este tema con el *proyecto Figuras*.

Este proyecto consta de 4 clases: Canvas (el lienzo), Circulo, Cuadrado y Triangulo.

Para crear un objeto de la clase Circulo hay que hacer:

- click en botón derecho en la clase Circulo y elegir new Circulo()
- introducir el nombre de la instancia

Así creamos un objeto *circulo1* de la clase Circulo. Este objeto ha quedado en el **Object Bench** (*depósito o banco de objetos*).



Proyecto Figuras

**Convención** – Los nombres de las clases en Java empiezan en mayúsculas (Circulo) y los nombres de los objetos en minúsculas (*circulo1*).

**Ejer. 2.1.** Crea otro círculo. Crea un cuadrado. Crea un triángulo.

## 2.3.- Llamando a métodos

### Concepto

A través de los *métodos* accedemos y modificamos el estado de un objeto (sus atributos)

Veamos ahora cómo podemos comunicarnos con los objetos invocando a sus **métodos**.

Los objetos hacen algo si llamamos / invocamos a sus métodos. Los métodos son los que definen el **comportamiento** de un objeto.

**Ejer.2.2.** Click en el botón derecho del objeto *circulo1* e invoca:

- hacerVisible()
- moverDerecha() varias veces
- moverAbajo() varias veces
- hacerInvisible()
- hacerVisible()

Cuando invocamos un método sobre un objeto estamos enviando un **mensaje** al objeto.

Un programa orientado a objetos es un conjunto de objetos que interactúan entre sí a través de mensajes. Algunos mensajes se los envía un objeto a sí mismo. Otros mensajes los envía un objeto a otro objeto.

**Concepto**  
Un *parámetro* proporciona información a un método.

## 2.4.- Parámetros

Los métodos pueden tener **parámetros**. Los parámetros proporcionan información adicional al método para que efectúe su tarea.

### Ejer.2.3.

- Invoca al método `moverHorizontal()`. Este método requiere información adicional, la distancia:

```
void moverHorizontal(int distancia)
```



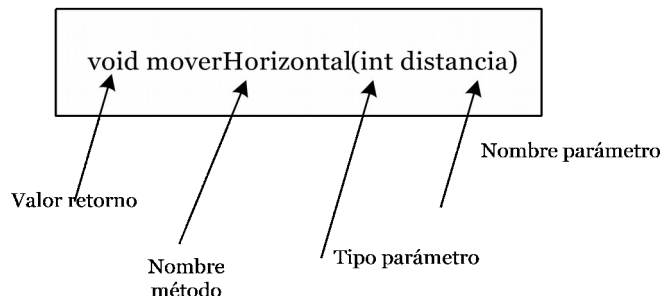
- Invoca `moverVertical()`. Invoca `moverDespacioVertical()`.

Cuando un método necesita un parámetro indica qué tipo de parámetro requiere:

```
void moverHorizontal(int distancia)
```

Esta línea es la **signatura** del método que:

- proporciona información sobre el método (*nombre* – será un nombre descriptivo – y *valor de retorno*)
- indica los *parámetros* que necesita: por cada parámetro se especifica su *tipo* y su *nombre*



**Concepto**  
El encabezado de un método es su *signatura*

Si un método no tiene parámetros al nombre del método le siguen dos paréntesis vacíos:

```
void hacerVisible()
```

```
void moverAbajo()
```

Otro ejemplo de método (con dos parámetros que se separan por ,):

```
void cambiarTamano(int nuevoAlto, int nuevoAncho)
```

*Convención* – Los nombres de los métodos se escriben en minúscula. Si es una palabra compuesta, la primera palabra se escribe en minúscula y las restantes comienzan en mayúscula.

```
void moverEspacioVertical(int distancia)
```

## 2.5.- Tipos de datos

Los parámetros tienen tipos. **Un tipo de datos** indica qué valores puede tomar el parámetro.

### Ejemplos

**Concepto**

Un *tipo de datos* define un conjunto de valores y las operaciones sobre esos valores

*int distancia* - indica que el parámetro es de tipo entero y tomará como valores los n<sup>os</sup> enteros.

El método `void cambiarColor(String color)` tiene un parámetro de tipo `String`. Este tipo indica que es una cadena de caracteres y cuando se escriba lo pondremos entre “ ”.

### Ejer.2.4.

- Llama al método `cambiarColor()` y cambia el color a “red”. Prueba con otros colores, “green”, “yellow”.
- Prueba con un color desconocido (“rojo”).
- Invoca el mismo método sin especificar ningún parámetro. ¿Qué ocurre?
- Prueba ahora a llamar al método pero especificando el color sin “”. ¿Qué obtienes?

Java soporta varios tipos de datos que veremos al final del tema.

## 2.6.- Múltiples instancias

Dada una clase podemos crear varios objetos (varias instancias) a partir de ella.

**Concepto**

*Múltiples instancias*: se pueden crear muchos objetos similares a partir de una clase

En nuestro ejemplo podemos crear varios círculos a partir de la clase `Circulo`. Cada uno de estos objetos tendrá su propio color, tamaño, posición. Podemos cambiar el atributo (característica) de un objeto círculo (su tamaño, por ejemplo) invocando al método `cambiarTamano()` pero sólo afectará a ese círculo, no a los demás.

### Ejer.2.5.

- Crea varios círculos
- Hazlos visibles
- Muévelos alrededor de la pantalla
- Haz un círculo más grande y amarillo
- Ahora haz otro mas pequeño y verde
- Prueba con otras figuras. Cambia sus posiciones, tamaño y colores

## 2.7.- Estado de un objeto

Todo objeto tiene un **estado**.

### Concepto

El estado de un objeto es el conjunto de valores de sus atributos en un instante de tiempo dado

El estado de un objeto está definido por los valores de sus **atributos (campos)**. En nuestro ejemplo, si tomamos un objeto *círculo* sus atributos (los que definen su estado) son:

*diámetro*  
*xPosicion*  
*yPosicion*  
*color*  
*esVisible*

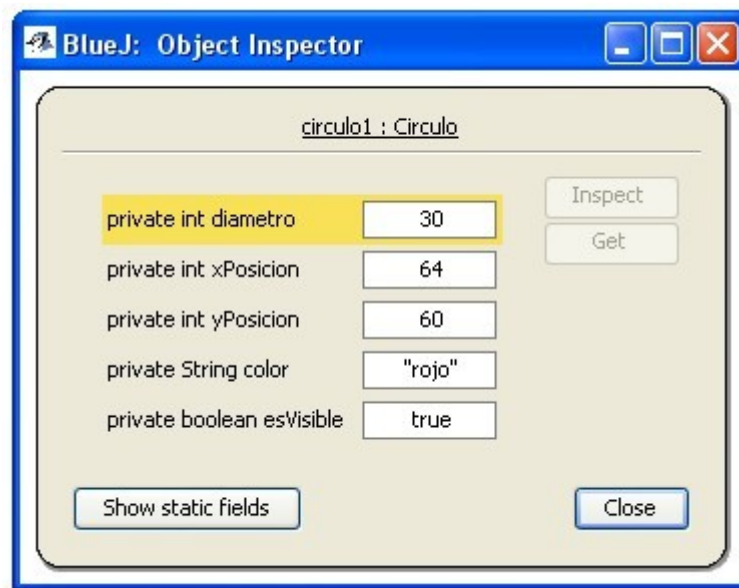
Los atributos también tienen un tipo que indica los valores que pueden tomar.

Algunos métodos modifican el estado de un objeto, otros consultan su estado.

### Ejemplos

`void moverIzquierda()` - modifica el atributo *xPosicion*  
`boolean esVisible()` - consulta el estado del atributo *esVisible*

En BlueJ podemos ver el estado de un objeto con la función *Inspect*.



### Ejer.2.6.

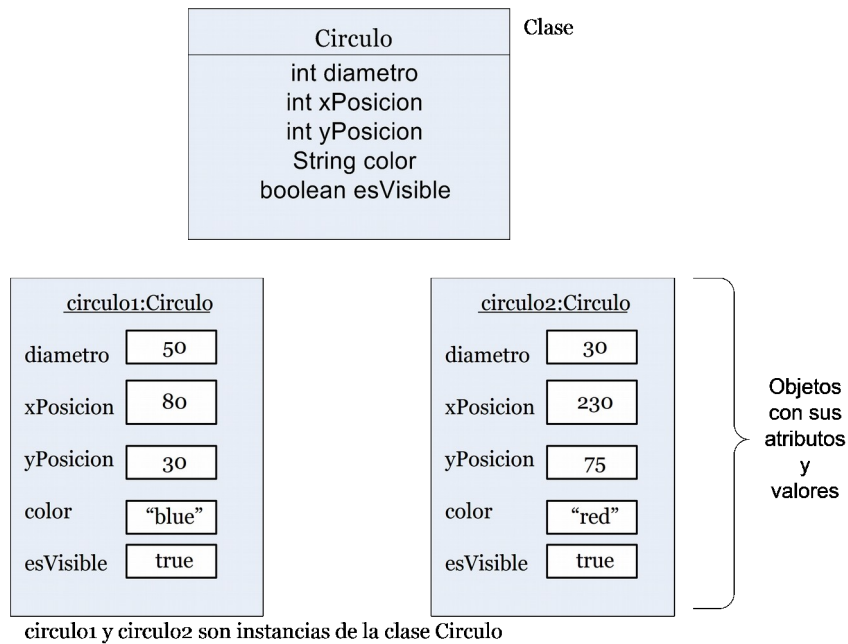
- Crea un círculo e inspecciona su estado
- Cambia el estado del círculo con algún método mientras el inspector de objetos está abierto. Observa cómo han cambiado sus atributos

## 2.8.- Qué hay en un objeto

Objetos de la misma clase tienen todos los mismos atributos. El nombre, tipo y nº de los campos es el mismo mientras que el valor actual de cada atributo en cada objeto puede variar.

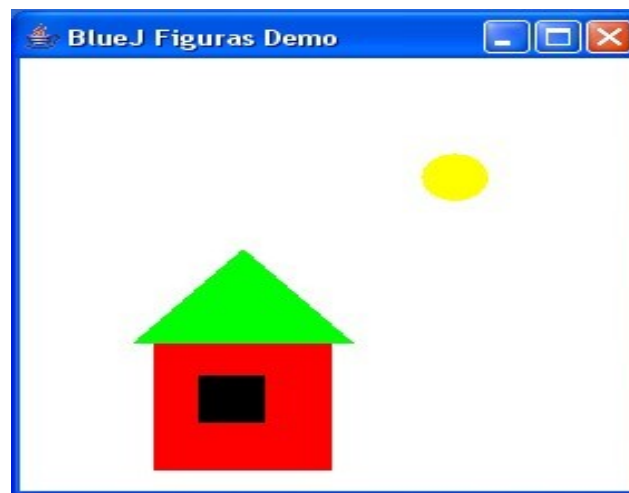
Objetos de diferentes clases pueden tener diferentes atributos. Un círculo tiene un campo *diámetro* mientras que un triángulo tienen un campo *ancho* y otro *altura*. El nº, tipo y nombre de los atributos se definen en la clase. Cuando se crea un objeto de la clase *Circulo*

automáticamente tienen los atributos definidos en esa clase. Los valores de esos atributos se almacenan en el objeto.



Los métodos se definen también en la clase. Todos los objetos de una misma clase tienen los mismos métodos (el mismo comportamiento). Sin embargo, los métodos se invocan sobre los objetos.

**Ejer.2.7.** A partir del proyecto Figuras crea una imagen similar a la que se muestra en la figura siguiente. Anota los pasos que sigues para conseguirlo. *Pistas* – Necesitarás un círculo amarillo, 2 cuadrados (uno rojo y uno negro) y un triángulo verde.



## 2.9.- Interacción de objetos

Los objetos pueden crear otros objetos y pueden interactuar entre ellos a través de llamadas a métodos. A esto se le llama **paso de mensajes** entre objetos.

Un programa Java es un conjunto de objetos. El usuario de un programa inicia el programa (normalmente creando un primer objeto) y todos los demás objetos se crean (directa o indirectamente) a partir de éste.

**Ejer.2.8.**

- Abre el proyecto Dibujo
- Crea una instancia de la clase Dibujo
- Invoca al método dibujar()
- Llama a los métodos setNegroBlanco() y setColor()
- ¿Cómo crees que la clase Dibujo efectúa el dibujo?

## 2.10.- Código fuente

Cada clase tiene un código fuente asociado escrito en el lenguaje Java. A través de ese código se describe la estructura de la clase:

- *atributos* (campos) que poseerán los objetos
- *métodos* (comportamiento) que se podrán invocar sobre los objetos de esa clase (mensajes que se podrán enviar a esos objetos)

Gran parte de nuestro trabajo a lo largo del curso será aprender a escribir las clases y para ello deberemos conocer el lenguaje Java.

**Ejer.2.9.** Desde BlueJ se puede ver el código fuente de una clase seleccionando *Abrir Editor* desde el menú contextual o haciendo doble click sobre la clase.  
Examina el código fuente de la clase Dibujo y de la clase Circulo.

A través del editor escribiremos el código fuente de una clase. Una vez escrito habrá que compilarla (botón *Compilar*). Si hacemos cambios al código fuente habremos de compilarla otra vez.

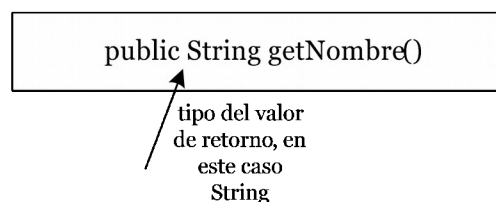
**Ejer.2.10.**

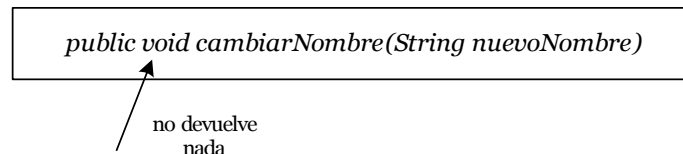
- En el código fuente de la clase Dibujo localiza la parte que hace el dibujo. Cámbialo para que el sol sea azul y no amarillo.
- Añade un segundo sol al dibujo. Tendrás que añadir un nuevo atributo a la clase:  
`private Circulo sol2;`
- Escribe después el código apropiado para crear el segundo sol.

## 2.11.- Valores de retorno

Un método puede devolver un valor (**valor de retorno**), un resultado.

La signatura de un método nos dice si devuelve o no un valor:





La palabra *void* indica que un método no devuelve nada, no retorna ningún resultado.

Los métodos que devuelven valores nos permiten consultar el estado de un objeto.

#### Ejer.2.11.

- Abre el proyecto LabClase. Este proyecto tiene 2 clases: Laboratorio y Estudiante. El proyecto está diseñado para llevar el control de estudiantes inscritos en un laboratorio y escribir la lista de todos ellos.
- Crea varios objetos de la clase Estudiante.
- Llama al método `getNombre()` de cada estudiante. ¿Qué ocurre?
- Crea un objeto de la clase Laboratorio. Indica el máximo nº de estudiantes que tendrá el laboratorio
- Envía el mensaje `numeroEstudiantes()` al objeto de la clase Laboratorio que acabas de crear. ¿Qué devuelve?

## 2.12.- Objetos como parámetros

Los objetos pueden ser pasados como parámetros a métodos de otros objetos.

```
void matricularEstudiante(Estudiante nuevoEstudiante)
```

El parámetro tiene como tipo la clase Estudiante.

#### Ejer.2.12.

- Asegúrate de que hay varios objetos de la clase Estudiante creados en el Object Bench
- Llama al método `matricularEstudiante()` de la clase Laboratorio y proporciona como parámetro alguno de los estudiantes creados (para pasar como parámetro un objeto de la clase Estudiante pon el cursor en el cuadro de diálogo y haz click en el objeto estudiante que quieras pasar)
- Haz lo mismo con dos o tres estudiantes más (matricúlalos en el laboratorio)
- Llama al método `escribirLista()` de la clase Laboratorio. Verás la lista de estudiantes matriculados en la *ventana de Terminal*
- Con el inspector de objetos examina los atributos del objeto de la clase Laboratorio

#### Ejer.2.13.

- Abre el proyecto Coche. Crea un objeto de nombre *miCoche* de color azul, marca Ford Focus y del año 2004.
- ¿De qué tipo es el atributo *color*? ¿Y el atributo *año*?
- Llama al método `getAño()`. ¿Qué devuelve? ¿De qué tipo es el valor que devuelve?
- Llama al método `setColor()` y cambia el color del coche.
- Llama al método `print()`.
- Crea otro objeto *tuCoche* proporcionando como parámetros los valores de atributos de tu coche favorito.
- Llama al método `igualQue()` del objeto *miCoche* y pasa como parámetro el objeto *tuCoche*. ¿Qué valor devuelve el método? ¿De qué tipo es este valor?



- Accede al código de la clase `Coche` y añade un nuevo atributo, *numeroPuertas*, de tipo entero.
- Añade un nuevo método `getNumeroPuertas()` (puedes hacerlo a través de la opción de menú BlueJ *Editar/Insertar Método*) que devuelve el valor de este atributo (hazlo viendo cómo está codificado, por ej, el método `getColor()`)
- Modifica el constructor añadiendo un nuevo parámetro, las puertas que tiene el coche, `int numPuertas`, e incluye el código necesario en el constructor.
- Modifica adecuadamente el método `igualQue()` para que dos coches sean iguales además si coincide su nº de puertas.

**Ejer.2.14.** Los siguientes valores, ¿a qué tipo de datos corresponden?

"Hola"   101   -4   false   "33"   -4.7

**Ejer.2.15.**

- Escribe la signatura de un método que se llama `calcularMedia()` que tiene dos parámetros (a y b ) de tipo *int* y devuelve un resultado también de tipo *int*.
- Escribe la signatura de un método `setNombre()` que tiene un parámetro de tipo *String* (*nuevoNombre*) y no devuelve nada.

**Ejer.2.16.** El ordenador que estás utilizando, ¿es un objeto o una clase?

## 2.13.- Los tipos de datos en Java

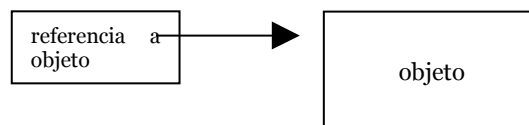
Ya hemos visto como los atributos de una clase, los parámetros y el valor de retorno de un método (si lo hay) tienen un determinado tipo de datos. El tipo de datos determina los valores que pueden tomar. En general, veremos que cualquier variable (no solo los atributos y parámetros) posee un determinado tipo de datos.

Java clasifica los tipos de datos en:

- a) tipos **primitivos** – son los tipos *simples*. El atributo o parámetro (la variable, en general) guarda directamente el valor de ese tipo.

```
private int altura;      178      altura
private boolean esVisible;  true      esVisible
```

- b) tipos **referencia** – almacenan una referencia (*puntero*) a un objeto.



```
private Estudiante estudiante1;
private Circulo sol;
private String nombreEstudiante;
```

De momento veremos únicamente los tipos primitivos.

### 2.13.1.- Los tipos primitivos en Java

Las siguientes tablas muestran todos los tipos primitivos del lenguaje Java. Java tiene seis tipos numéricos: cuatro para enteros y dos para números reales. Proporciona además un tipo para representar un carácter simple y otro tipo para representar valores lógicos.

Nombre del tipo	Precisión	Rango	Ejemplos
<b>Tipos numéricos enteros</b>			
<b>byte</b>	8 bits	-128 a 127	24 -2 123
<b>short</b>	16 bits	-32768 a 32767	1234 -23456
<b>int</b>	32 bits	$-2^{31}$ a $2^{31} - 1$	-2003 5409
<b>long</b>	64 bits	$-2^{63}$ a $2^{63} - 1$	4233781L 55L

**Precisión** – el nº de bits que utiliza la máquina virtual de Java para guardar el valor

**Rango** – conjunto de valores que comprende el tipo (valores que se pueden almacenar)

Nombre del tipo	Precisión	Rango	Ejemplos
<b>Tipos numéricos reales</b>			
<b>float</b>	32 bits	-3.4E38 a 3.4E38	43.889F
<b>double</b>	64 bits	-1.7E308 a 1.7E308	45.63 2.4e5 45.64

Nombre del tipo	Precisión	Rango	Ejemplos
<b>Otros tipos</b>			
<b>char</b>	16 bits	Un carácter Unicode (UTF-16)	'm' '?' '\u00F6'
<b>boolean</b>	Java no lo especifica	Valor booleano <i>true</i> o <i>false</i>	true false

- Un nº sin punto decimal se interpreta como un *int* (entero). Si queremos que sea un entero largo (*long*) hay que añadir una L. **Ej.** 46 46L
- Un nº con punto decimal se interpreta como un *double*. Si se especifica una F al final será *float*. **Ej.** 46.89 46.89F
- Un nº real también puede representarse en notación exponencial. **Ej.** 24.5e-3
- Java considera el código Unicode (codificación UTF-16) para representar los caracteres. Es un código de 16 bits (cada carácter es representado por 4 dígitos hexadecimales, '\u0041' es la letra 'A') que permite representar más de 65000 caracteres de cualquier idioma del mundo. Unicode incluye el código ASCII utilizado por la mayoría de los ordenadores para el intercambio de información.
- Un carácter se escribe entre comillas simples o como un valor Unicode de 4 dígitos hexadecimales precedido de \u. **Ej.** 'A' , 'a', '?' '\u00F6'
- Algunos caracteres se pueden especificar mediante *secuencias de escape*, comenzando por la barra invertida \, y tienen un significado especial.

Carácter	Significado
\n	Salto de línea
\t	Tabulador
\"	Comillas dobles
'	Comillas simples
\\	Barra invertida

Los valores booleanos son *true* y *false*.

## 2.13.2.- Los operadores y las expresiones en Java

Sobre cada uno de los tipos anteriores se pueden utilizar un conjunto de operadores para formar expresiones.

Una **expresión** se construye agrupando operadores y operandos. Las expresiones se *evalúan* y producen un resultado de un determinado tipo.

### 2.13.2.1- Las expresiones aritméticas

Se construyen con *operadores aritméticos*. Los operandos son valores de un tipo primitivo numérico, entero o real. Cuando se evalúan producen como resultado un valor numérico.

<b>Operadores aritméticos</b>	$+$ $-$ $*$ $/$ $\%$ (módulo o resto de una división)
-------------------------------	---

El resultado de aplicar los operadores  $/$  y  $\%$  depende de si los operandos son enteros o reales. Si son enteros la división es entera, si son reales (o al menos uno de los operandos lo es) el resultado es real.

<b>Ejemplos</b>	$5 + 3 \Rightarrow 8$ $5 / 3 \Rightarrow 1$ $5.0 / 3 \Rightarrow 1.66666$	$5 / 2 \Rightarrow 2$ $7 \% 3 \Rightarrow 1$
-----------------	---	---

Cuando aparece más de un operador en una expresión se aplican las *reglas de precedencia* de operadores (la tabla se muestra en la siguiente pregunta). Para cambiar la prioridad de los operadores se utilizan los paréntesis. Si varios operadores tienen la misma prioridad la evaluación se realiza de izquierda a derecha (excepto en el caso de los operadores de asignación que se evalúan de derecha a izquierda)

<b>Ejemplos</b>	$51 * 3 - 53 \Rightarrow 100$ $154 - 2 * 27 \Rightarrow 100$ $(200 - 5) / 2 \Rightarrow 97$ $2 * (47 + 3) \Rightarrow 100$
-----------------	---


**Ejer.2.17.** Evalúa las siguientes expresiones aritméticas. Indica el tipo de resultado, entero o real.

$25 + 20 - 15$	
$20 * 10 + 15 * 10$	
$20 * 10 / 2 - 20 + 3 * 3$	
$15 / 10 * 2 + 3 / 4 * 8$	
$46 \% 9 + 4 * 4 - 2$	
$45 + 43 \% 5 * (23 + 3 \% 2)$	
$1.5 * 3$	

**Ejer.2.18.** Transforma las siguientes expresiones matemáticas en expresiones Java:

$2\pi\text{radio}$ ( $\pi = 3.1416$ )	
$2\pi\text{radio}^2$	
$a^2 + \frac{b^2}{c}$	
$\frac{4}{3(r+34)} - 9(a+bc) + 3 + \frac{d(2+a)}{a+bd}$	
$-b + \frac{\sqrt{b^2 - 4ac}}{2a}$	

### 2.13.2.2- Precedencia de los operadores

Operadores		<p>Mayor prioridad</p>  <p>Menor prioridad</p>
( )	Paréntesis	
++ --	Incremento / Decremento	
+ - !	Suma / Resta (Unario)	
* / %	Producto / División / Resto	
+ -	Suma / Resta	
< <= > >=	Comparación	
== !=	Igual / Distinto	
&&	boolean (y / and)	
	boolean (or / o)	
= += -= *= /= %=	Operadores de asignación	

### 2.13.2.3- Las expresiones booleanas

Producen, al evaluarlas, un resultado lógico (booleano), *true* (cierto) o *false* (falso).

Se construyen utilizando operadores relacionales y/o lógicos.

Los operadores relacionales usualmente se combinan con operandos y operadores aritméticos (también con operandos de tipo *char*).

<b>Operadores relacionales</b>	<code>==</code> <code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>&gt;=</code> <code>!=</code>
--------------------------------	---

Los operadores lógicos (o booleanos) combinan expresiones booleanas para producir un resultado booleano.

<b>Operadores booleanos</b>	<code>&amp;&amp;</code> <code>  </code> <code>!</code>
-----------------------------	--

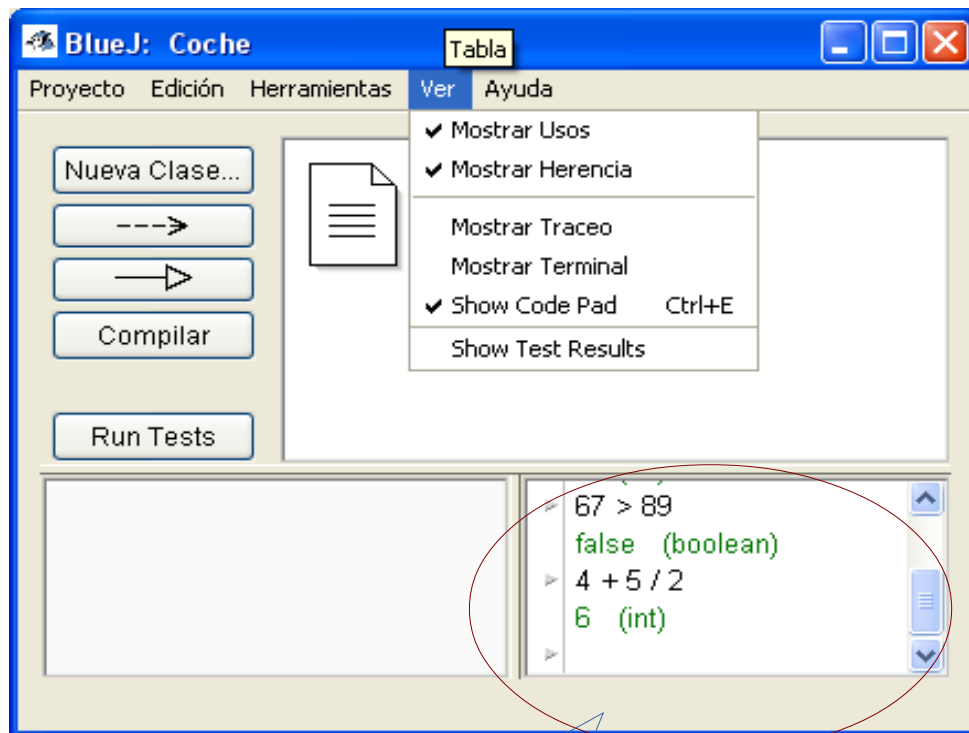
Se evalúan según las siguientes tablas de verdad:

<b>a</b>	<b>b</b>	<b>a &amp;&amp; b</b>	<b>a    b</b>	<b>!a</b>
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>
<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>false</i>
a,b: expresiones booleanas				

Java utiliza la evaluación en **cortocircuito** (*short circuiting*) en las expresiones que contienen operadores lógicos. Tan pronto como se conoce el resultado final de la expresión no se evalúan el resto de condiciones. Con el operador `&&` la evaluación se para con el primer *false*, en el caso del operador `||` con el primer *true*.

<code>sexo == 'M' &amp;&amp; edad &gt;= 65</code>	si <i>sexo</i> no es igual a 'M' la condición es <i>false</i> y el resultado final de la expresión será también <i>false</i> , no se evalúa la segunda condición
<code>notaTeoria &gt;= 5    notaPractica &gt; 7</code>	si <i>notaTeoria</i> es mayor o igual a 5 la condición es <i>true</i> y el resultado final de la expresión será también <i>true</i> , no se evalúa la segunda condición

### 2.13.3.- BlueJ y el CodePad



El CodePad permite introducir expresiones y sentencias para evaluarlas/ejecutarlas de forma inmediata

**Ejer.2.19.** Evalúa las siguientes expresiones lógicas, suponiendo x con valor 7:

$(\text{true}) \ \&\& \ (3 > 4)$	
$(\text{true}) \ \&\& \ (x > 4)$	
$x \ != \ 3$	
$(x > 0) \    \ (x < 0)$	
$25 > 20 \ \&\& \ 13 > 5$	
$10 + 4 < 15 - 3 \    \ 2 * 5 + 1 > 14 - 2 * 2$	
$4 * 2 \leq 8 \    \ 2 * 2 < 5 \ \&\& \ 4 > 3 + 1$	

**Ejer.2.20.** Define los siguientes atributos indicando su tipo:

- estaVacía*, que indica si una urna está o no vacía
- edad*, que indica la edad de una persona
- facturaLuz*, que denota el importe de la factura de la luz

- d) *estadoCivil* que indica el estado civil de una persona ('S' 'C' 'V' 'D')
- e) *nombreAsignatura* que indica el nombre de la asignatura que se está cursando
- f) *areaFigura* que indica el área de una determinada figura
- g) *estaAprobado* que indica si un alumno ha aprobado o no cierta asignatura

**Ejer.2.21.** Sean los siguientes atributos:

```
private int numero;
private int edad;
private int dia;
private int nota;
private char estadoCivil;
```

Construye las siguientes expresiones lógicas de tal manera que al evaluarlas sean ciertas si *numero*:

- a) es distinto de 0
- b) es igual a 0
- c) está comprendido entre 1 y 100
- d) está entre 1 y 100 o es negativo
- e) es par
- f) es múltiplo de 4
- g) es múltiplo de 4 pero no de 100

sea cierta :

- h) si una persona es adulta

sea cierta si *dia*:

- i) suponiendo que corresponde al mes de enero, es correcto
- j) suponiendo que corresponde al mes de abril, es incorrecto

sea cierta si una persona:

- k) ha aprobado un examen
- l) es soltera o casada