Tema 2. Objetos y clases

Principales conceptos que se abordan en este tema

- Introducir los conceptos de ***clase*** y ***objeto*** a través de un ejemplo
- Aprender a trabajar con el entorno de desarrollo BlueJ

Número de sesiones

Se han estimado un total de 10 horas

2.1 El entorno BlueJ. Instalación y configuración

Para realizar la instalación del entorno BlueJ seguiremos los siguientes pasos básicos:

1. Descarga de la versión deseada desde la web oficial o desde los enlaces propuestos más abajo. En nuestro caso, por estar en fase de iniciación, podría ser suficiente descargar la versión de instalación que incluye en JDK.

2. Seleccionar la plataforma o sistema operativo, existen versiones para Windows, Ubuntu/Debian Linux MacOS, Raspian Linux y otros sitemas operativos. En todas las versiones disponibles se descarga un archivo ejecutable que se encarga de la instalación.

4. Comenzará la descarga del archivo de instalación ejecutable y una vez finalizada, lanzar éste, comenzando la instalación en nuestro equipo.

6. Posteriormente, establecemos el directorio donde se instalará BlueJ así como la carpeta que contiene el JDK que se utilizará por defecto.

7. Finalmente, la instalación se completa y dispondremos de este entorno totalmente operativo.

Para llevar a cabo las operaciones descritas en el paso 1, te ofrecemos la posibilidad de descargare software a través del siguiente enlace:

Descargar BlueJ

Tutorial de BlueJ

2.2 Objetos y clases

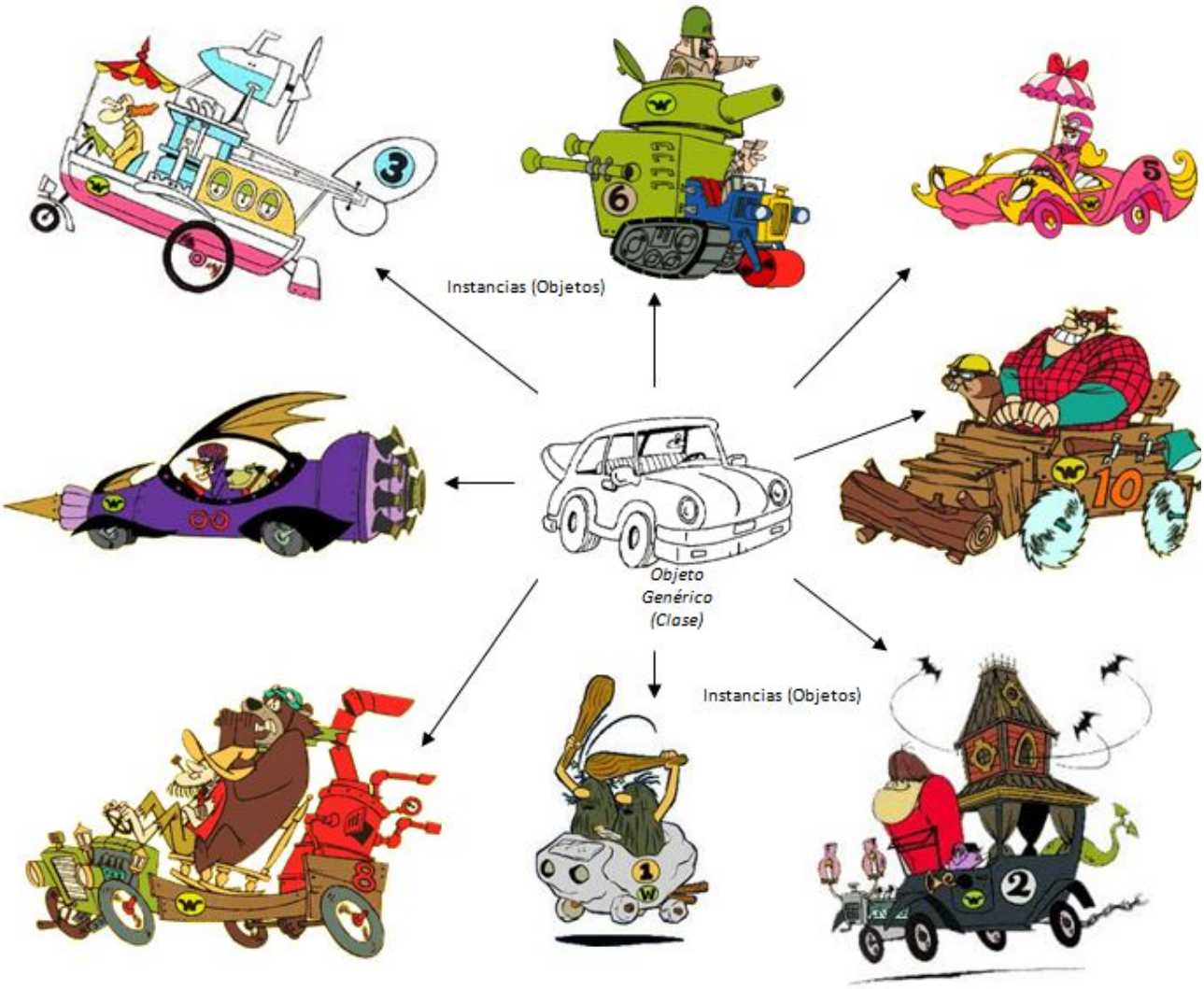
Cuando escribimos un programa en un lenguaje orientado a objetos estamos creando (en el ordenador) un modelo de alguna parte del mundo real. Las partes que constituyen el modelo son objetos que aparecen en el dominio del problema.

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”).



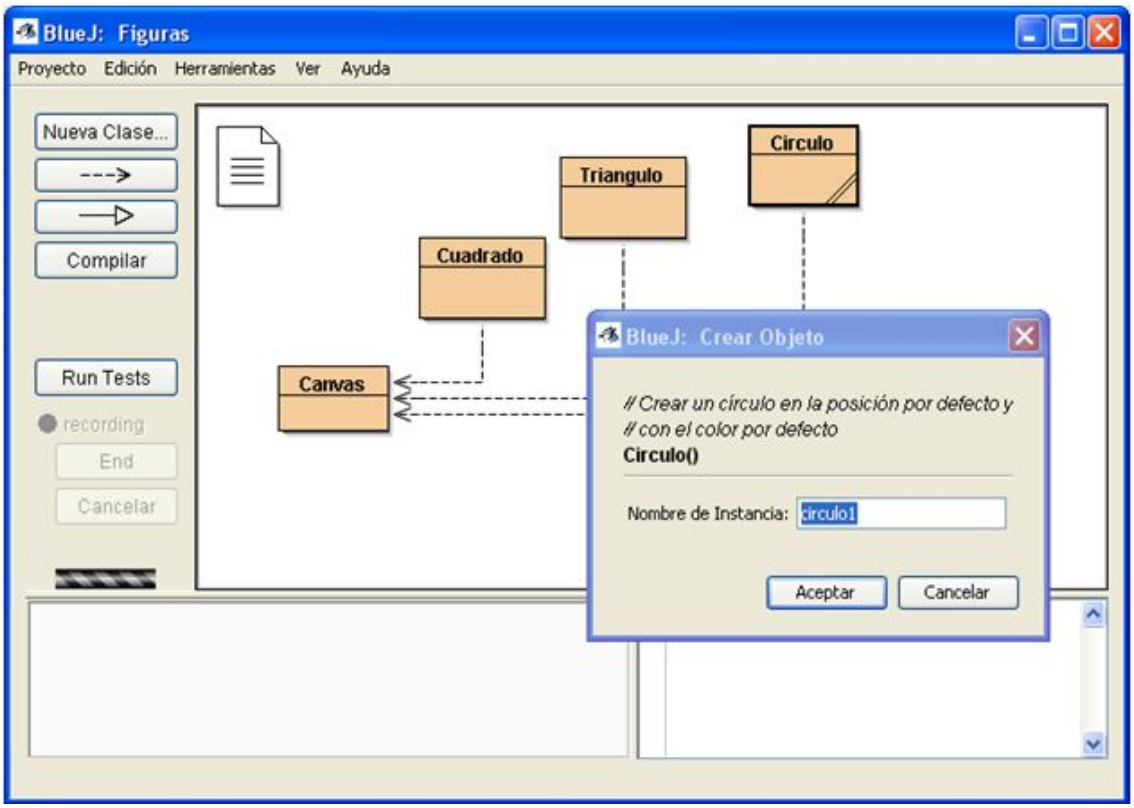
Caso práctico

Trabajaremos en este tema con el *proyecto Figuras.*

Inicie BlueJ y abra el proyecto *Figuras. (Enlace para descargar 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



Proyecto Figuras

Ejercicio 2.1

- Crea otro círculo.
- Crea un cuadrado.
- Crea un triángulo.

Debes conocer

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

2.3 Llamadas a métodos

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. A través de los métodos, accedemos y modificamos los datos del objeto.

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.

Ejercicio 2.2

Ejercicio (sobre el proyecto Figuras)

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

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

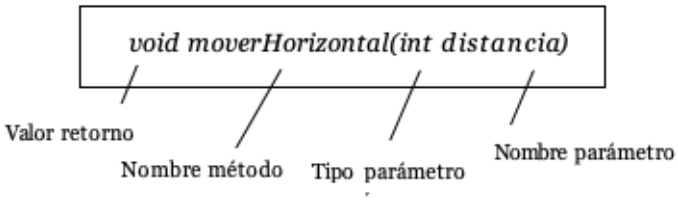
2.4 Parámetros

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

La **signatura** de un método define su entrada y su salida.

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*



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)
```

Debes conocer

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 moverDespacioVertical(int distancia)

Ejercicio 2.3

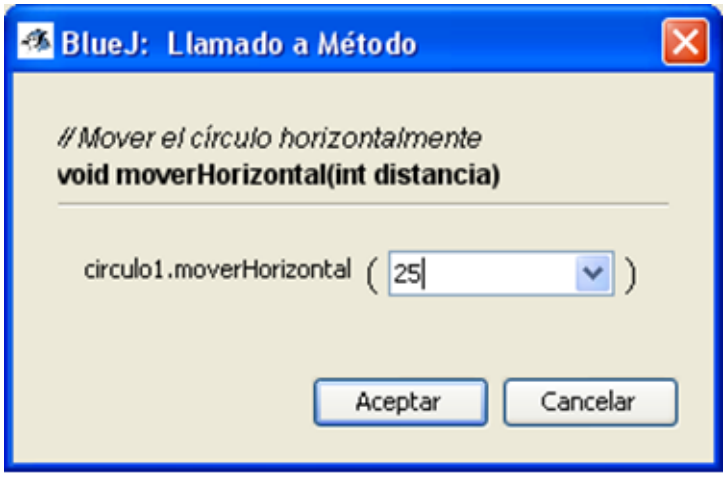
Ejercicio (sobre el proyecto Figuras)

Invoca al método moverHorizontal*()*.

Este método requiere información adicional, la distancia.

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

void moverHorizontal(int distancia)



- Invoca moverVertical*()*.
- Invoca moverDespacioVertical*()*.

2.5 Tipos de datos

Los atributos de una clase, los parámetros y el valor de retorno de un método (si lo hay) tienen un determinado tipo de datos. **Un tipo de datos** indica qué valores puede tomar el parámetro. En general, veremos que cualquier variable (no solo los atributos y parámetros) posee un determinado tipo de datos.

Ejemplos

int distancia - indica que el parámetro es de tipo entero y tomará como valores los nos 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 " ".

Java clasifica los tipos de datos en:

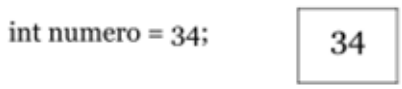
a) tipos **primitivos** – son los tipos *simples*. El atributo o parámetro (la variable, en general) guarda directamente el valor de ese tipo.

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

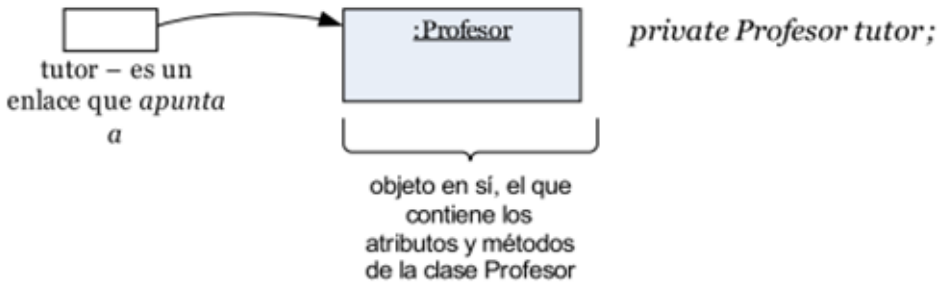
Aquellos definidos por las clases. Algunas clases son clases predefinidas de Java, como la clase String, otras las define el usuario.

Ambos tipos pueden utilizarse para indicar el tipo de una variable pero hay diferencias entre ellos:

- las variables de un tipo primitivo almacenan directamente el valor



- las variables de un tipo objeto (aquellas cuyo tipo es una clase) almacenan una *referencia* al objeto, no el objeto en sí.



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
Tipos numéricos enteros			
byte	8 bits	-128 a 127	24 -2 123
short	16 bits	-32768 a 32767	1234 -23456
int	32 bits	-2^{31} a $2^{31}-1$	-2003 5409
long	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
Tipos numéricos reales			
float	32 bits	-3.4E38 a 3.4E38	43.889F
double	64 bits	-1.7E308 a 1.7E308	45.63 2.4e5 45.64

Nombre del tipo	Precisión	Rango	Ejemplos
Otros tipos			
char	16 bits	Un carácter Unicode	'm' '?' '\u00F6'
boolean	8 bits	Valor booleano *true* o *false*	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 para representar los caracteres. Es un código de 16 bits (cada carácter es representado por 4 dígito 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*.

Ejercicio 2.4

Ejercicio (sobre el proyecto Figuras)

- 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?

2.6 Múltiples instancias

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

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.

Ejercicio 2.5

Ejercicio (sobre el proyecto Figuras)

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

2.7 Estado de un objeto

Todo objeto tiene un **estado**.

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:

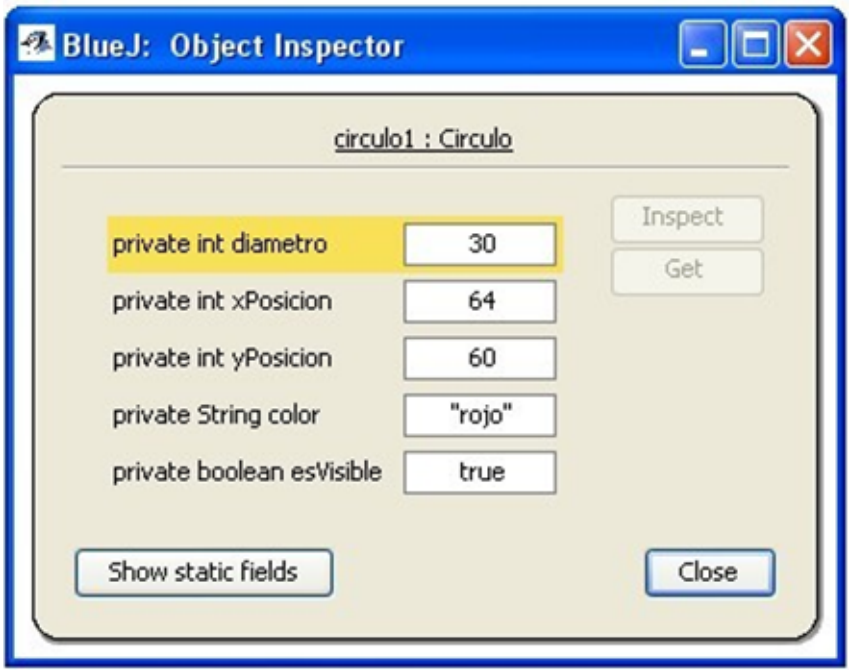
diámetro
xPosicion
yPosicion
color
esVisible

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*.



Ejercicio 2.6

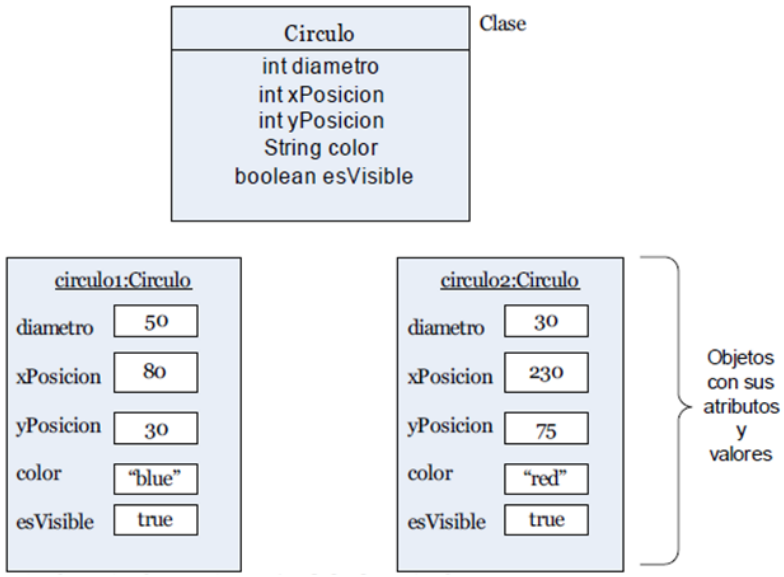
Ejercicio (sobre el proyecto Figuras)

- 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.



circulo1 y circulo2 son instancias de la clase Circulo

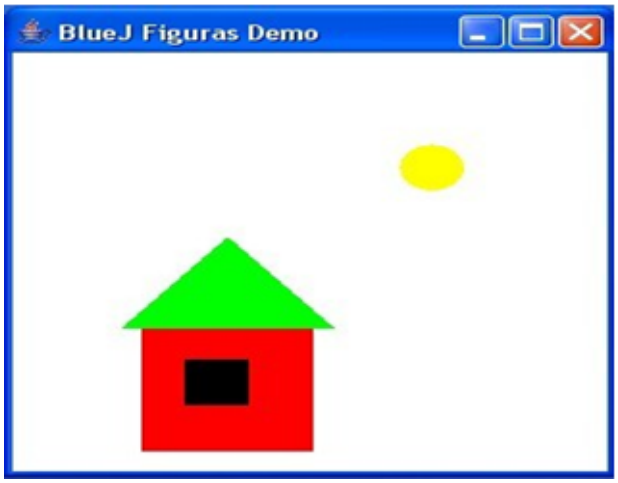
Los métodos se definen también en la clase. Todos los objetos de una misma clase tienen los mismos métodos (el mismo comportamiento). Sin embargo, los métodos se invocan sobre los objetos.

Ejercicio 2.7

Ejercicio (sobre el proyecto Figuras)

- 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.


BlueJ Figuras Demo

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 mensaje**s 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.

Ejercicio 2.8

Ejercicio (sobre el proyecto Figuras)

- Abre el proyecto Dibujo (el proyecto Dibujo es el proyecto Figuras + la clase Dibujo.java)
- 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.

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.

Ejercicio 2.9

Ejercicio (sobre el proyecto Dibujo) Enlace para descargar el proyecto Dibujo

- 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.

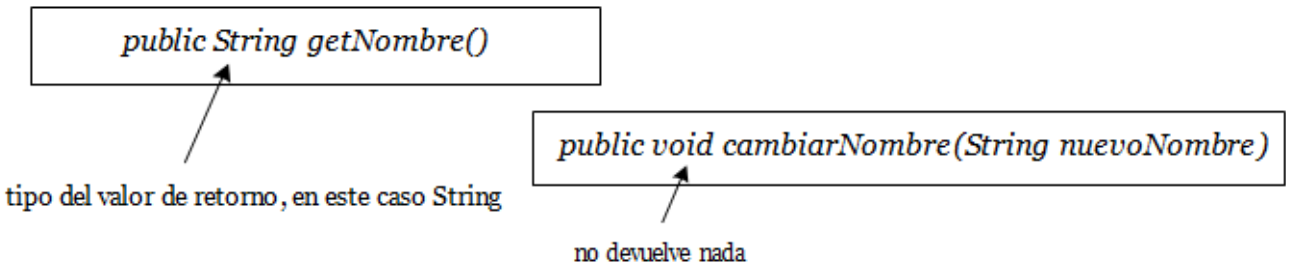
- En el código fuente de la clase Dibujo localiza la parte que hace el dibujo.
- Cámbialo para que el sol sea azul y no amarillo.
- Añade un segundo sol al dibujo. Tendrás que añadir un nuevo atributo a la clase:

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.

Ejercicio 2.10

Ejercicio (sobre el proyecto LabClase) Enlace para descargar el proyecto LabClase

- 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 tendrzá 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.

Ejercicio 2.11

Ejercicio (sobre el proyecto LabClase)

- 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 Termina*l
- Con el inspector de objetos examina los atributos del objeto de la clase Laboratorio

2.13 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.

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.

Operadores aritméticos	+ - * / % (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.

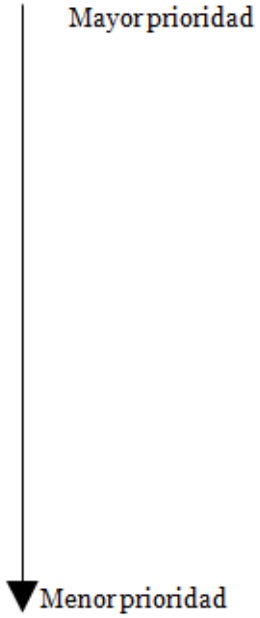
Ejemplos	5 + 3 → 8 5 / 3 → 1 5.0 / 3 → 1.66666	5 / 2 → 2 7 % 3 → 1

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)

Ejemplos	51 * 3 – 53 → 100 154 - 2 * 27 → 100 (200 - 5) / 2 → 97 2 * (47 + 3) → 100

Precedencia de los operadores

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



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*).

Operadores relacionales	== < <= > >= !=

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

Operadores booleanos	&& || !

Se evalúan según las siguientes tablas de verdad:

a	b	a && b	a || b	!a
false	*false*	*false*	*false*	*true*
false	*true*	*false*	*true*	*true*
true	*false*	*false*	*true*	*false*
true	*true*	*true*	*true*	*false*
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***.

sexo == 'M' && edad >= 65	si *sexo* no es igual a 'M' la condición es *false* y el resultado final de la expresión será también *false*, no se evalúa la segunda condición
notaTeoria >= 5 || notaPractica > 7	si *notaTeoria* es mayor o igual a 5 la condición es *true* y el resultado final de la expresión será también *true*, no se evalúa la segunda condición

Ejercicio 2.12

Evalúa las siguientes expresiones aritméticas:

- Indica el tipo de resultado, entero o real.

25 + 20 - 15 = ☐ Tipo de resultado: ☐

20 * 10 + 15 * 10 = ☐ Tipo de resultado: ☐

20 * 10 / 2 - 20 + 3 * 3 = ☐ Tipo de resultado: ☐

15 / 10 * 2 + 3 / 4 * 8 = ☐ Tipo de resultado: ☐

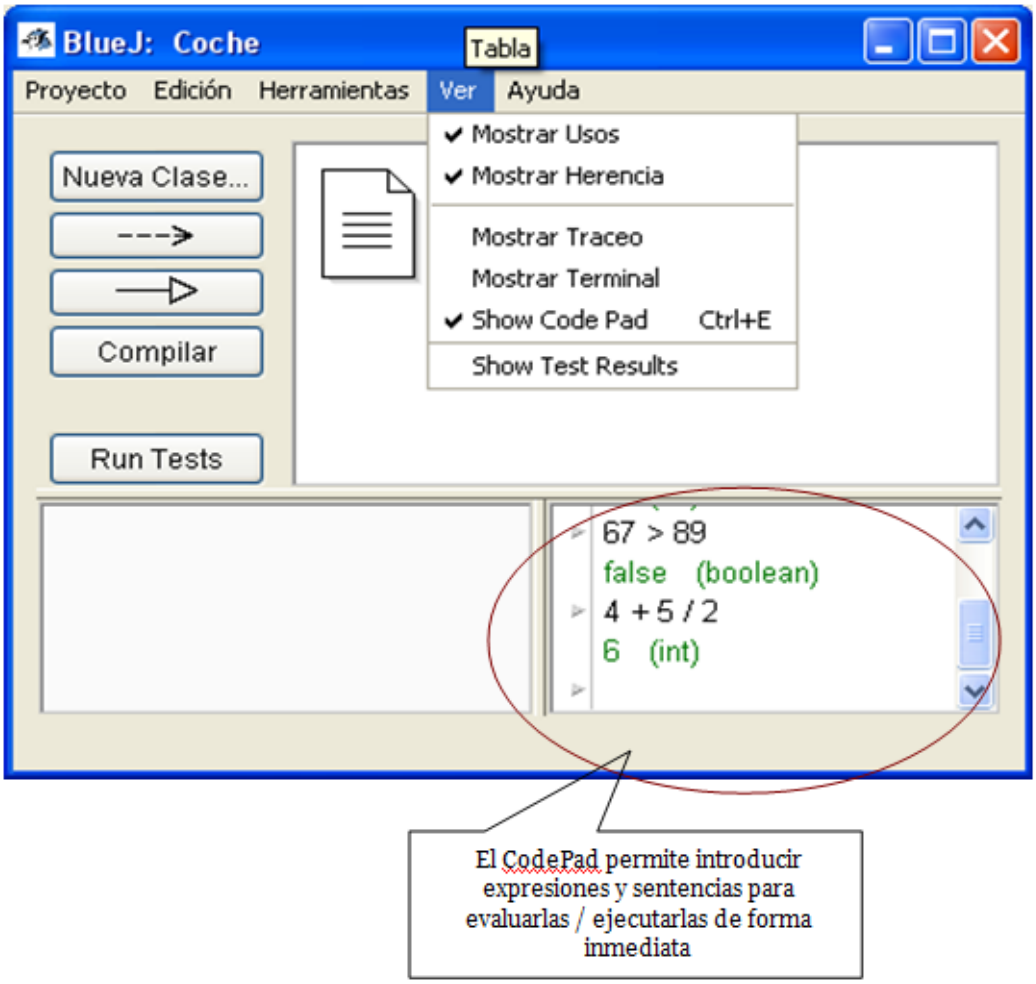
46 % 9 + 4 * 4 - 2 = ☐ Tipo de resultado: ☐

45 + 43 % 5 * (23 + 3 % 2) = ☐ Tipo de resultado: ☐

1.5 * 3= ☐ Tipo de resultado: ☐

Averiguar la puntuación | Mostrar/Eliminar las respuestas

2.14 BlueJ y el CodePad



Ejercicio 2.13

Evalúa las siguientes expresiones lógicas, suponiendo x con valor 7:

(true) && (3 > 4) => ____

(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 <= 8 || 2 * 2 < 5 && 4 > 3 + 1 => ____

Averiguar la puntuación | Mostrar/Eliminar las respuestas

Ejercicio 2.14

Define los siguientes atributos indicando su tipo:

Utiliza sólo una vez los siguientes tipos: double, String, char, boolean, byte, float

a) *estaVacia*, que indica si una urna está o no vacía -->

b) *edad*, que indica la edad de una persona -->

c) *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 -->

Averiguar la puntuación | Mostrar/Eliminar las respuestas

Ejercicio 2.15

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

sea cierta si una persona:

j) ha aprobado un examen

k) es soltera o casada

Averiguar la puntuación | Mostrar/Eliminar las respuestas