La librería Consola DAW

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

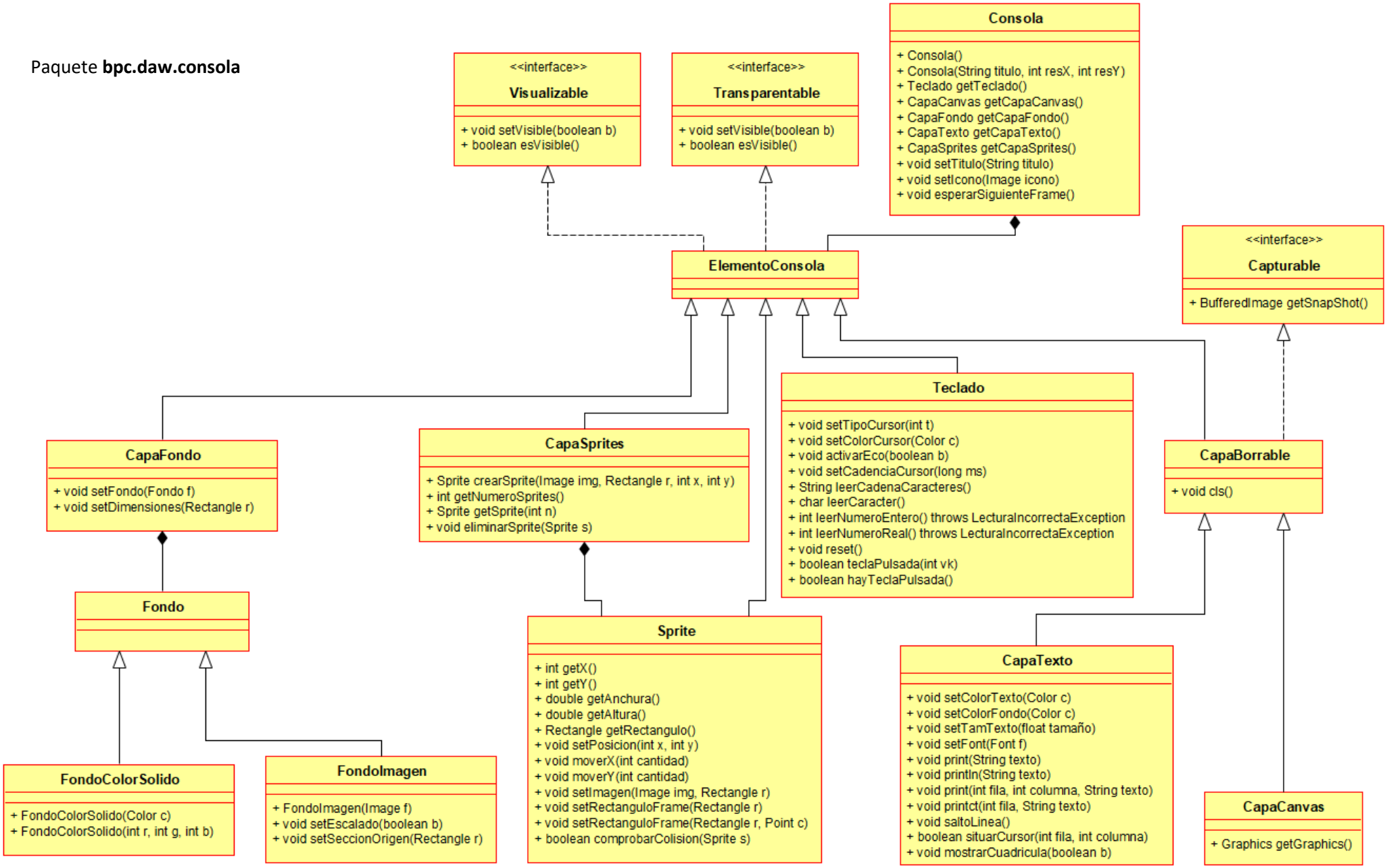
Esta librería convierte la interfaz de usuario de una aplicación en un sistema de programación a pantalla completa que ofrece mayores posibilidades que la terminal estándar de Java. Está basada en los modos clásicos de texto y gráficos de los PC y se programa de forma similar a los lenguajes como C, Pascal, etc.

Cuando un programa crea una "Consola DAW", la pantalla cambia y pasa a estar formada por cuatro capas gráficas que se superponen:

- **Capa de fondo:** Es el fondo de la pantalla, que puede ser de color fijo o una imagen.
- **Capa canvas:** Es una capa en la que se puede dibujar mediante Java 2D.
- **Capa de texto:** Contiene todo el texto que se muestra en la pantalla.
- **Capa de sprites:** En esta capa se pueden poner imágenes con capacidad de movimiento y detección simple de colisiones (sprites).

Además, la consola ofrece un teclado que permite la comprobación del estado de las teclas y la lectura de tipos de datos básicos. El diagrama de clases de esta librería está en la página siguiente:

Paquete **bpc.daw.consola**



1) Consola

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase transforma la interfaz de usuario del programa que la usa en una Consola DAW, y es el punto de entrada para acceder a las cuatro capas que la forman.

Consola
+ Consola() + Consola(String titulo, int ancho, int alto) + void setTitulo(String t) + void setIcono(Image ic) + CapaFondo getCapaFondo() + CapaTexto getCapaTexto() + CapaCanvas getCapaCanvas() + CapaSprites getCapaSprites() + Teclado getTeclado() + void esperarSiguienteFrame()

- El primer constructor crea una Consola DAW y la pone en marcha. La consola se cierra automáticamente cuando termina el programa principal, o al pulsar ALT+F4. En este último caso el programa principal finaliza también aunque no se haya terminado.
- El segundo constructor crea una Consola DAW en una ventana. Como parámetros se recibe el título de la ventana, su ancho y su alto medidos en píxeles. No es posible cambiar el tamaño de la ventana una vez creada.
- El método setTítulo permite indicar el texto que se mostrará en la ventana de la aplicación cuando es ejecutada por el sistema operativo.
- El método setIcono permite indicar la imagen que se mostrará como icono de la aplicación cuando es ejecutada por el sistema operativo.
- El método getCapaFondo devuelve la capa de fondo de la consola.
- El método getCapaTexto devuelve la capa de texto de la consola.
- El método getCapaCanvas devuelve la capa canvas de la consola.
- El método getCapaSprites devuelve la capa de sprites de la consola.
- El método getTeclado devuelve el teclado asociado a la consola.
- El método esperarSiguienteFrame sincroniza el programa con el momento en el que es seguro dibujar sin que se produzca parpadeo.

2) ElementoConsola

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase representa un componente cualquiera de la consola (capa, sprite, etc) y posee los métodos comunes a todos ellos.

ElementoConsola

3) Visualizable

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase representa un componente que puede mostrarse u ocultarse en la pantalla.

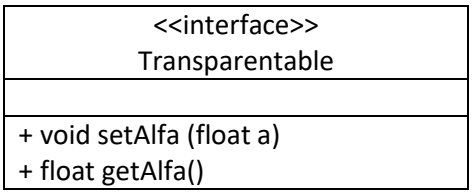
<<interface>> Visualizable
+ void setVisible (boolean b) + boolean esVisible()

- El método setVisible permite mostrar u ocultar el componente. Si se pasa **true** el componente se muestra, y si se pasa **false** se oculta.
- El método esVisible devuelve true si el componente representado por el objeto se está viendo en la consola.

4) Transparentable

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase representa un componente que tiene un grado de transparencia llamado “canal alfa”. El canal alfa es un número entre 0 y 1 que varía entre 0 (completamente transparente) a 1 (completamente visible).



- El método setAlfa permite cambiar el grado de transparencia (o **canal alfa**) del componente. Debe ser un valor con decimales comprendido entre 0.0 y 1.0.
- El método getAlfa devuelve el valor de transparencia del componente, expresado como un número con decimales comprendido entre 0.0 y 1.0.

5) Teclado

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase representa el teclado de la consola DAW y permite comprobar el estado de las teclas (saber si una tecla concreta está pulsada en un momento), leer datos de tipo básico y personalizar la apariencia del cursor que aparece cuando se leen datos por teclado.

Teclado
+ boolean teclaPulsada (int vk) + boolean hayTeclaPulsada() + String leerCadenaCaracteres() + char leerCaracter() + int leerNúmeroEntero() throws Exception + double leerNúmeroReal() throws Exception + void activarEco (boolean b) + void setCadenciaCursor (long ms) + void setTipoCursor (int tipo) + void setColorCursor (Color c) + void reset()

- teclaPulsada nos devuelve true si la tecla cuyo código pasamos como parámetro está pulsada en el momento en que llamamos al método. Los códigos son constantes definidas en la clase **KeyEvent** (paquete java.awt). Hay un código para cada tecla.
- hayTeclaPulsada devuelve true si hay alguna tecla pulsada en ese momento.
- leerCadenaCaracteres lee un String por teclado.
- leerCaracter lee un char por teclado.
- leerNúmeroEntero lee un int por teclado. Si se escribe algo que no es de ese tipo de dato se lanza una excepción.
- leerNúmeroReal lee un double por teclado. Si se escribe algo que no es de ese tipo de dato se lanza una excepción.
- activarEco activa o desactiva el eco del teclado. Cuando el eco está activado, las teclas que pulse el usuario para introducir datos se verán en la pantalla.
- setCadenciaCursor define el número de milisegundos entre cada parpadeo del cursor.
- setTipoCursor permite cambiar el tipo de cursor, según las siguientes constantes:

Teclado.CURSOR_NULO	No se muestra ningún cursor
Teclado.CURSOR_SOLIDO	El cursor es un bloque rectangular de color sólido
Teclado.CURSOR_HUECO	El cursor es un rectángulo con borde sin relleno
Teclado.CURSOR_LINEA	El cursor es una línea horizontal situada inferiormente

- setColorCursor permite cambiar el color del cursor.
- reset pone el estado de todas las teclas como "no pulsadas" aunque lo estén. Se usa cuando se pulsa una tecla y en el siguiente frame la pulsación previa origina un efecto indeseado.

6) CapaFondo

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase representa el color o imagen de fondo que se muestra en la consola. Es la primera capa que se dibuja, por lo que todas las demás (canvas, texto, sprites) se ven encima de ella. Por defecto esta capa tiene un fondo de tipo color sólido negro.

CapaFondo
+ void setFondo (Fondo f) + public void setDimensiones (Rectangle r)

- El método setFondo permite cambiar el fondo de la consola. Debe pasarse un objeto de la clase Fondo en cualquiera de sus dos subclases: FondoImagen o FondoColorSólido.
- El método setDimensiones permite definir el rectángulo de la pantalla donde se dibujará el fondo. Por defecto dicho rectángulo es la pantalla completa. Si se indica un rectángulo diferente, el fondo se dibujará exclusivamente dentro de esa zona, por lo que fuera de ella la consola puede mostrar gráficos extraños si no se maneja con cuidado.

7) FondoColorSólido

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase representa un fondo de la consola compuesto por un único color sólido.

FondoColorSólido
+ FondoColorSólido (int r, int g, int b) + FondoColorSólido (Color c)

- El primer constructor crea un fondo formado por un color cuyos componentes RGB (rojo, verde, azul) se pasan como parámetros. Cada uno de ellos debe ser un valor comprendido entre 0 y 255.
- El segundo constructor crea un fondo formado por un objeto color que se pasa como parámetro. La clase Color se encuentra en el paquete java.awt

8) FondoImagen

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase representa un fondo de la consola formado por una imagen.

FondoImagen
+ FondoImagen (Image i) + void setEscalado (boolean r) + void setSecciónOrigen (Rectangle r)

- El constructor crea un fondo cuya imagen se pasa como parámetro.
- El método setEscalado hace que la imagen se ajuste al tamaño de la capa de fondo. Si se pasa false, no se producirá escalado y se mostrará solo la parte de la imagen que entre dentro de la capa.
- El método setSecciónOrigen se utiliza cuando no queremos mostrar la imagen de fondo completamente, sino una parte de ella. Usando este método indicamos el rectángulo de la imagen que queremos mostrar. Es interesante destacar que si el rectángulo que se pasa como parámetro es una variable y ésta cambia, la imagen de fondo de la consola se actualiza automáticamente al nuevo valor del rectángulo.

9) CapaBorrable

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase es una capa cuyo contenido puede borrarse.

CapaBorrable
+ cls()

- El método cls (*abreviatura de Clear Screen, una orden clásica en programación*) borra el contenido de la capa y la deja vacía. Tras llamarse a este método, el contenido que tuviese la capa se perderá.

10) Capturable

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta interfaz representa un componente de la consola del que podemos capturar su imagen.

Capturable
+ BufferedImage getSnapShot()

- El método getSnapShot devuelve una imagen con lo que se muestra en el objeto sobre el que se ha llamado al método.

11) CapaTexto

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta clase es la capa que contiene todo el texto mostrado en la consola. Está formada por una cuadrícula en la que se muestra una letra en cada casilla.

CapaTexto
+ void setColorTexto (Color c) + void setColorFondo (Color c) + void setTamTexto (float t) + void setFont (Font f) + void print(String s) + void print(int f, int c, String s) + void println(String s) + void printct(String s) + void saltoLinea() + void situarCursor(int f, int c) + boolean setDimensiones (float t, int f, int c, Point p) + void mostrarCuadrícula(boolean b)

- setColorTexto cambia el color del texto para todo lo que se escriba a continuación.
- setColorFondo cambia el color del fondo para todo lo que se escriba a continuación.7
- setTamTexto borra la pantalla y cambia la cuadrícula de la capa para mostrar texto más grande. Al llamar a este método cambia el tamaño de las letras.
- setFont cambia el tipo de letra. Se recomienda usar fuentes de tipo monospace (todos los caracteres son de igual tamaño). Para otros tipos de letra se recomienda escribir el texto usando la capa Canvas.
- print imprime un mensaje en la posición en la que se encuentre el cursor.
- El segundo método print imprime un mensaje en la fila y columna indicadas.
- println imprime un mensaje en la posición del cursor y realiza un salto de línea.
- El método saltoLínea realiza un salto de línea desde la posición del cursor.
- situarCursor hace que el cursor de escritura de texto pase a la fila y columna indicada como parámetro.
- setDimensiones permite limitar la capa de texto a una zona determinada de la pantalla. El primer parámetro es el tamaño de las letras, el segundo el número de filas de texto deseadas, el tercero es el número de columnas y el cuarto el punto de la pantalla donde se quiere situar la capa de texto.
- mostrarCuadrícula hace que muestre u oculte una cuadrícula que ayuda a ver dónde se colocan las letras en el texto.

12) CapaCanvas

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

En muchos lenguajes de programación un Canvas es una zona donde puede dibujarse de forma libre: figuras geométricas, imágenes, etc. La consola DAW proporciona la clase CapaCanvas, que es una capa en la que podemos dibujar usando la librería Java 2D.

CapaCanvas
+ Graphics getGraphics()

- El método getGraphics nos proporciona un objeto Graphics (puede ser convertido en Graphics2D mediante casting) para dibujar en la capa mediante Java 2D.

13) CapaSprites

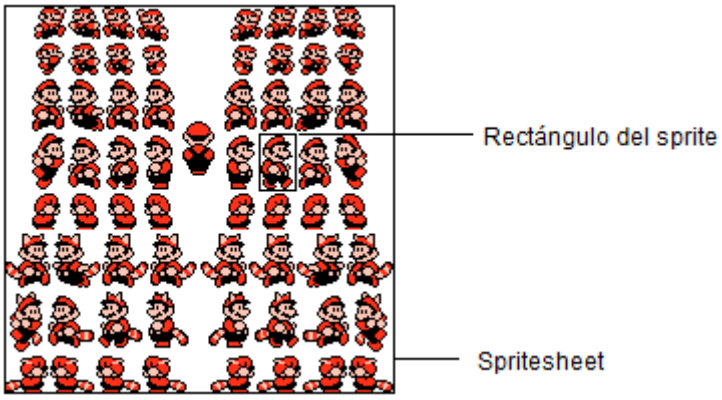
- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Esta capa está formada por todos los sprites que se muestran en la consola y proporciona métodos para crearlos y borrarlos.

CapaSprites
+ Sprite crearSprite (Image i, Rectangle r, int x, int y) + int getNúmeroSprites() + Sprite getSprite (int n) + void eliminarSprite (Sprite s)

- El método crearSprite crea un nuevo sprite y lo añade a la capa de sprites. Para ello recibe una imagen, la zona rectangular de la imagen que define la forma inicial del sprite y las coordenadas de la pantalla donde se situará.

  Cuando se trabaja con sprites lo habitual es tener una imagen (llamada **Spritesheet**) con todas las animaciones de un personaje, y definir un rectángulo que será el que mostrará el estado actual del sprite. Si el rectángulo cambia, el sprite cambia automáticamente a la zona definida en el nuevo rectángulo.



- El método getNúmeroSprites devuelve el número total de sprites que hay en la consola.
- El método getSprite devuelve el n-ésimo sprite registrado en la consola, o null si no existe dicho sprite.
- El método eliminarSprite borra un sprite de la consola.

14) Sprite

- **Librería:** *Consola DAW*
- **Archivo:** *ConsolaDAW.jar*
- **Desarrollador:** *Juan Diego*
- **Paquete:** *bpc.daw.consola*

Un sprite es una imagen rectangular que puede moverse por la pantalla y puede "chocar" con otros sprites. Los sprites se crean y destruyen por medio de la capa de sprites.

Sprite
+ void setPosición (int x, int y) + void moverX (int c) + void moverY (int c) + void setImagen (Image img) + void setRectángulo (Rectangle r) + boolean comprobarColisión (Sprite s)

- El método setPosición cambia la posición del sprite al punto de coordenadas (x,y)
- El método moverX desplaza en la dirección del eje X al sprite en la cantidad de píxeles indicada. Si el parámetro es positivo se mueve hacia la derecha, y si es negativo, hacia la izquierda.
- El método moverY desplaza en la dirección del eje Y al sprite en la cantidad de píxeles indicada. Si el parámetro es positivo se mueve hacia abajo, y si es negativo, hacia arriba.
- El método setImagen cambia la "spritesheet" asociada al sprite.
- El método setRectángulo cambia el rectángulo que define la porción de la "spritesheet" que se mostrará en el sprite.
- El método comprobarColisión devuelve true si el sprite está chocando con el que se pasa como parámetro. Se considera que hay un choque si los rectángulos que encierran ambos sprites tienen intersección (método **Bounding Box**).