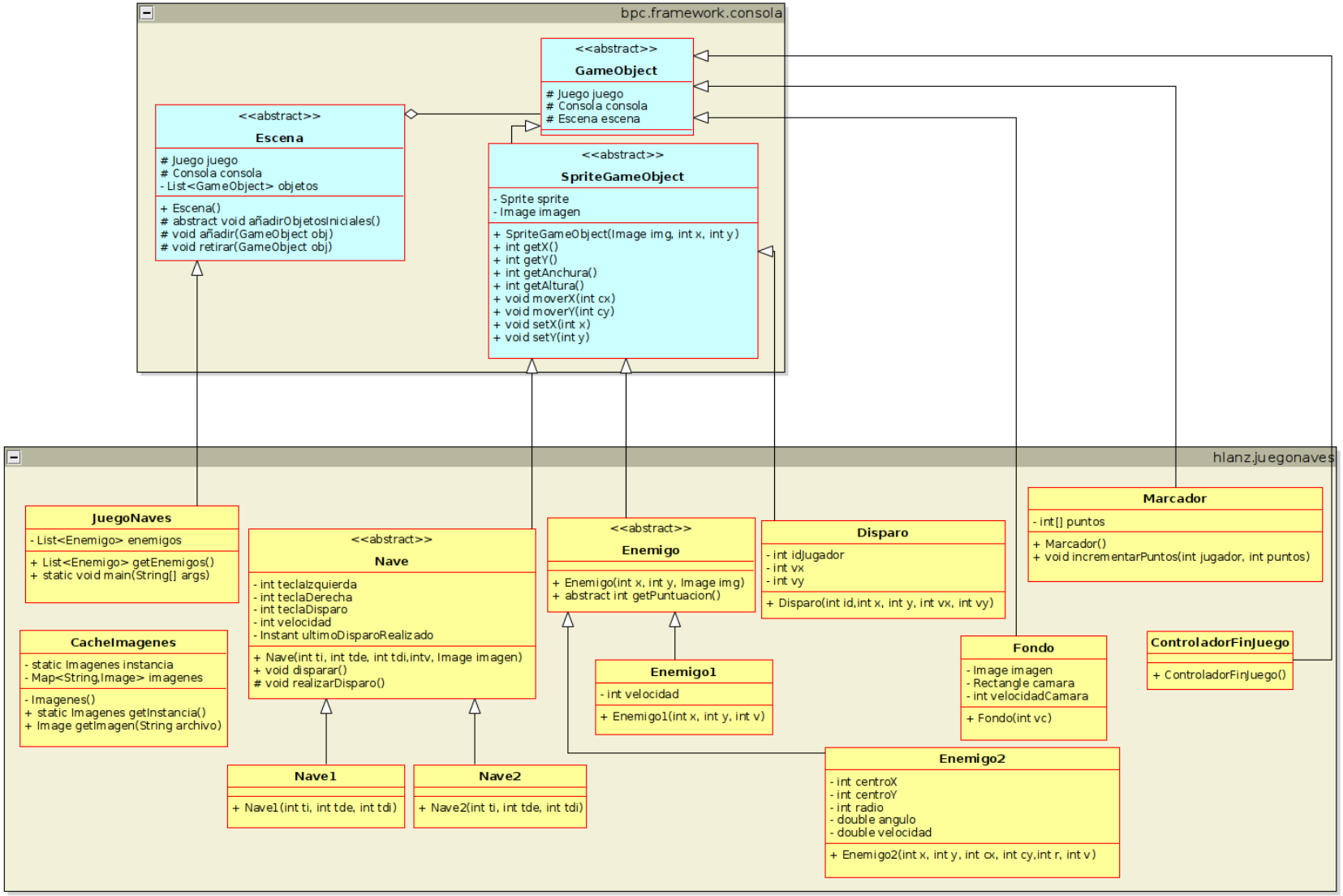
DIAGRAMA DE CLASES: JUEGO DE NAVES



CacheImagenes

Esta clase se encarga de almacenar las imágenes del juego, para que no sea necesario tener que cargarlas continuamente sino solo cuando son necesarias. La clase está diseñada de forma que solo haya un objeto posible de esta clase (**singleton**) y además las imágenes se cargarán en la memoria justo cuando se necesitan **(lazy loading**)

- instancia: Es el único objeto de esta clase que estará en la memoria RAM.
- imagenes: Es un Map que asocia el nombre de un archivo con su correspondiente imagen.

Los métodos son:

- Constructor: Crea un map vacío para almacenar las imágenes.

- getInstancia: Este método hace esto:

  - Si la propiedad **instancia** está vacía (**null**), se creará un objeto de la clase **Imagenes** y se guardará en dicha propiedad. Si la propiedad **instancia** no está vacía, no se hará nada.
  - En cualquier caso el método devuelve el valor de la propiedad **instancia**.

> ☞ ***¿Qué sentido tiene esto?****: Como el constructor es privado, la única forma de obtener el objeto* ***Imagenes*** *es mediante el método estático. Este método usa el constructor solo la primera vez que es llamado, devolviendo el objeto (la propiedad estática* ***instancia****) creado las siguientes veces. Con esto se consigue que en la memoria solo haya un objeto de la clase que estará disponible con el método* ***getInstancia*** *al resto de clases del juego.*

- getImagen: Este método realiza lo siguiente:

  - Si en el Map existe una imagen cuya ruta coincide con la que se pasa como parámetro, el método devuelve la imagen asociada a dicha ruta.
  - En caso contrario, se usará la clase **ImageIO** para cargar la imagen cuya ruta se pasa como parámetro, se almacenará en el Map (para futuros usos) y el método terminará devolviendo la imagen cargada.

Nave

Esta clase es la nave de un jugador. La nave se puede mover a izquierda y derecha y puede disparar una vez cada segundo.

- teclaIzquierda: Es una constante entera que guarda el código de la tecla izquierda del jugador, tal y como se define en la clase **KeyEvent**
- teclaDerecha: Es una constante entera que guarda el código de la tecla derecha del jugador, tal y como se define en la clase **KeyEvent**
- teclaDisparo: Es una constante entera que guarda el código de la tecla para disparar del jugador, tal y como se define en la clase **KeyEvent**
- velocidad: Es la velocidad a la que se desplaza la nave del jugador.
- ultimoDisparoRealizado: Es el instante en el que la nave ha realizado su último disparo.

Sus métodos son:

- constructor: Inicializa todas las propiedades.

- ejecutarFrame: Este metodo hace lo siguiente:
  - Si está pulsada la tecla izquierda, la nave se moverá hacia la izquierda la cantidad de píxeles indicada en el parámetro de velocidad. Si la nave se sale por el lado izquierdo de la pantalla, no se hará nada
  - Si está pulsada la tecla derecha, la nave se moverá hacia la derecha la cantidad de píxeles indicada en el parámetro de velocidad. Si la nave se sale por el lado derecho de la pantalla, no se hará nada
  - Si está pulsada la tecla de disparo se llamará al método **disparar**.

- disparar: Se obtendrá el instante actual y se comprobará si ha pasado más de un segundo desde el último disparo. En caso afirmativo, se actualizará **ultimoDisparoRecibido** para que sea igual al instante actual y después se llamará al método **realizarDisparo**. En caso contrario, no se hará nada.

- realizarDisparo: Es un método que será implementado en las clases hijas de forma que cada clase hija tenga su propio disparo.

Nave1

Esta clase es una nave de tipo 1

Sus métodos son:

- constructor: Inicializa todas las propiedades teniendo en cuenta que la velocidad de la nave es de 5 y que la imagen que usará es el archivo **nave1.png**. Se usará la **CacheImagenes** para obtener dicha imagen.

- realizarDisparo: se creará y añadirá a la escena del juego un objeto **Disparo** en las coordenadas (x,y) de la nave con una velocidad de 0 píxeles en el eje x y -10 píxeles en el eje Y (con esto conseguimos que el disparo se mueva hacia arriba). El disparo creado se añadirá a la escena.

Nave2

Esta clase es una nave de tipo 2

Sus métodos son:

- constructor: Inicializa todas las propiedades teniendo en cuenta que la velocidad de la nave es 3 y que la imagen que usará es el archivo **nave2.png**. Se usará la **CacheImagenes** para obtener dicha imagen.

- realizarDisparo: Si está pulsada la tecla de disparar se crearán y añadirán a la escena del juego dos objetos **Disparo** en las coordenadas (x,y) de la nave con estas velocidades:

    - El primer disparo, velocidad en ejeX -3, velocidad en ejeY -3
    - El segundo disparo, velocidad en ejeX 3, velocidad en ejeY -3

Disparo

Esta clase es un disparo emitido por un jugador

- idJugador: Es el número del jugador que ha emitido el disparo. Puede ser 0 (jugador 1) o 1 (jugador 2)
- vx: Es la velocidad horizontal del disparo
- vy: Es la velocidad vertical del disparo

Sus métodos son:

- constructor: Inicializa todas las propiedades teniendo en cuenta las propiedades recibidas y que la imagen que usará es el archivo **disparo.png**. Se usará la **CacheImagenes** para obtener dicha imagen.

- realizarFrame: Este método hace todo esto:

  - Desplaza el disparo horizontalmente la cantidad de píxeles indicada en la propiedad **vx**
  - Desplaza el disparo verticalmente la cantidad de píxeles indicada en la propiedad **vy**.
  - Si el disparo se sale de la pantalla se eliminará de la escena.
  - Se obtendrá la lista de enemigos que hay en la escena y se comprobará si el disparo colisiona con alguno de ellos. En caso de que sea así:
    - se retirarán de la escena el disparo y el enemigo
    - se incrementará al marcador del jugador que ha disparado con el valor de la puntuación del enemigo.

Enemigo

Esta clase es una nave enemiga genérica

Sus métodos son:

- constructor: Inicializa las propiedades
- getPuntuacion: Devuelve los puntos que gana un jugador cuando mata al enemigo. Esto dependerá de cada tipo de enemigo y por eso es abstracto.

Los métodos abstractos heredados seguirán siendo abstractos

Enemigo1

Esta clase es un enemigo que se mueve continuamente de izquierda a derecha.

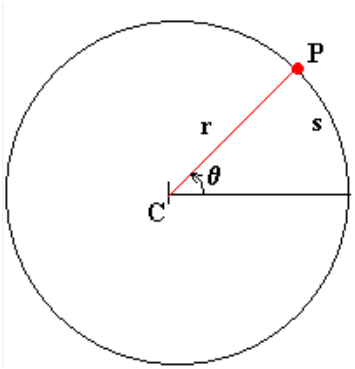
- velocidad: Es la velocidad horizontal del enemigo

Sus métodos son:

- constructor: Inicializa las propiedades, teniendo en cuenta que la imagen que usa el enemigo es **malo1.png**
- realizarFrame: Mueve al enemigo en el eje X la cantidad de píxeles indicada en su velocidad. Si se detecta que el enemigo se va a salir de la pantalla (por el lado izquierdo o el lado derecho) se cambiará el signo de la velocidad (o sea, si la velocidad era positiva se pondrá negativa y viceversa. Con esto conseguimos que cuando se vaya a salir se mueva para el otro lado).
- getPuntuacion: Devuelve el doble de la velocidad a la que se mueve el enemigo

Enemigo2

Esta clase es un enemigo que se mueve en una trayectoria circular. Mira esta imagen, en la que el punto rojo P es la posición del enemigo dentro del círculo.



- centroX: Es la coordenada x del centro del círculo que describe el enemigo. En la imagen sería la coordenada x del punto C
- centroY: Es la coordenada y del centro del círculo que describe el enemigo. En la imagen sería la coordenada y del punto C
- radio: Es el radio del círculo que describe el enemigo. En la imagen es la “r”
- angulo: Es el ángulo del círculo en el que está situado el enemigo. Su valor estará entre 0 y 2π. En la imagen es “θ”
- velocidad: Es el incremento de ángulo que el enemigo avanza en cada frame.

Sus métodos son:

- constructor: Inicializa las propiedades, teniendo en cuenta que el ángulo inicial es 0 y la imagen que usa el enemigo es **malo2.png**
- realizarFrame: Este método incrementa el valor de la propiedad **ángulo** la cantidad indicada en la propiedad **velocidad**. Si el nuevo valor del ángulo es superior o igual a 2π, se modificará y volverá a pondrá a **0**. A continuación, usa las fórmulas trigonométricas de seno y coseno para determinar las coordenadas (x,y) del enemigo y muevelo a dicha posición.
- getPuntuacion: Devuelve el triple de la velocidad a la que se mueve el enemigo.

Fondo

Esta clase es el fondo del juego. Como la imagen del fondo es muy grande, hay una cámara que sirve para ir desplazando el fondo continuamente hacia abajo.

- imagen: Es la imagen del fondo
- camara: Es un rectángulo con la parte del fondo que se verá en la pantalla. La cámara empieza arriba de la imagen y en cada frame la cámara se moverá un poco hacia abajo.
- velocidadCamara: Es la velocidad con la que la cámara se va desplazando hacia abajo.

Sus métodos son:

- constructor: inicializa las propiedades, teniendo en cuenta que la imagen que usa el fondo es el archivo **fondo.png**

- inicializar: Pone la cámara con un nuevo rectángulo cuya esquina superior izquierda es el punto (0,0) y el ancho y alto de la cámara son el alto y ancho de la pantalla.

- realizarFrame: Dibuja en la capa canvas de la Consola DAW la parte de la imagen de fondo que está definida en el rectángulo de la cámara. A continuación, mueve la cámara verticalmente la cantidad de píxeles indicada en la propiedad **velocidadCamara**. En caso de que el borde inferior de la cámara se salga de la imagen, se moverá la cámara al punto inicial (0,0).

- finalizar: No hace nada.

Marcador

Esta clase guarda las puntuaciones de los 2 jugadores del juego.

- puntos: Es un array de dos posiciones. La primera es la puntuación del jugador 1 y la segunda la puntuación del jugador 2.

Sus métodos son:

- constructor: crea el array de puntos para guardar los puntos de 2 jugadores.
- inicializar: pone a 0 los puntos de los dos jugadores.
- incrementarPuntos: Este método recibe el identificador de un jugador (0 = jugador 1, 1 = jugador 2) e incrementa la puntuación correspondiente la cantidad pasada como parámetro. Si la puntuación es negativa se lanzará una IllegalArgumentException con el mensaje “La puntuación no puede ser negativa”
- realizarFrame: Escribe en la Capa Canvas de la Consola DAW en la parte superior de la pantalla los puntos de los dos jugadores. Es importante por motivos de eficiencia NO usar la capa de texto, sino escribir directamente en la capa canvas.
- finalizar: No hace nada.

ControladorFinJuego

Esta clase comprueba si el juego debe terminar. El juego finaliza cuando todos los enemigos han sido destruidos y gana la partida el jugador que más puntos ha conseguido.

Sus métodos son:

- constructor: No hace nada
- ejecutarFrame: Detiene el juego si la lista de enemigos está vacía
- iniciar y finalizar → no harán nada

JuegoNaves

Esta clase es la escena del juego, y a la vez es el programa que la pone en marcha.

- enemigos: Es una lista donde se van a almacenar los enemigos que hay en el juego.

Métodos:

- getEnemigos: Devuelve la lista de enemigos.

- añadir: Se sobreescribirá este método para hacer lo siguiente:
  - Primero llama al método añadir de la clase padre, para que no se pierda lo que ya hace en la clase padre.
  - A continuación, si el GameObject que se recibe como parámetro es un objeto de la clase **Enemigo**, entonces se añadirá a la lista de enemigos.

- retirar: Se sobreescribirá este método para hacer lo siguiente:
  - Primero llama al método retirar de la clase padre, para que no se pierda lo que ya hace en la clase padre.
  - A continuación, si el GameObject que se recibe como parámetro es un objeto de la clase **Enemigo**, entonces se retirará de la lista de enemigos.

- añadirObjetosIniciales: Usa el método heredado **añadir** para añadir en este orden:
  - Un **ControladorFinJuego**
  - Un objeto **Fondo**
  - Dos objetos **Nave1** y **Nave2**. Puedes elegir libremente las teclas de cada jugador.
  - Añade de la forma que prefieras varios enemigos a la pantalla. Puedes ponerlos a mano, puedes hacer que se generen aleatoriamente, etc.
  - Por último, añade un objeto **Marcador**.

- main: Es el método que pone en marcha el juego. Creará un objeto de la clase **Juego,** e iniciará una escena **JuegoNaves** a resolución FullHD.