 **Komet Kevin**

**Proyecto Software**

Planificación de coste



Autor: Arturo García Enguita

Fecha: 1-4-19

***Ingeniería Informática | Curso 2018 - 2019.***



*Komet Kevin por Ramiro Woutersen Uriarte , Sergio Langarita Benitez, Arturo García Enguita, Denis Florin Cobeti se distribuye bajo una*[*licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional*](http://creativecommons.org/licenses/by-nc-nd/4.0/)*.*

**Tabla de contenidos**

[4.- Diseño UML 3](#__RefHeading___Toc757_1322100828)

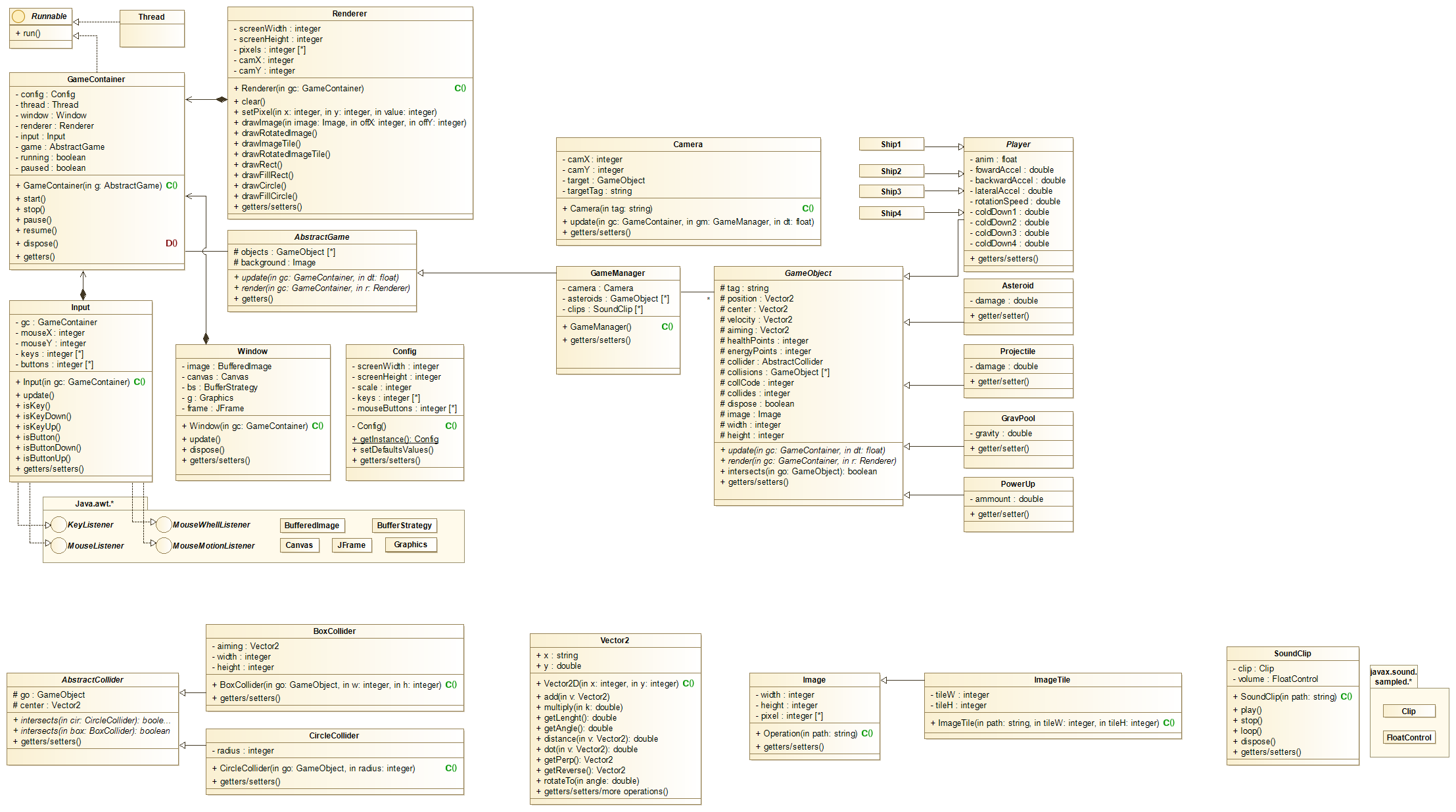
[4.2.- Diagramas de clases 3](#__RefHeading___Toc5030327)

[4.3.- Diagrama de actividad 6](#__RefHeading___Toc5030328)

## 4.- Diseño UML

## 4.2.- Diagramas de clases

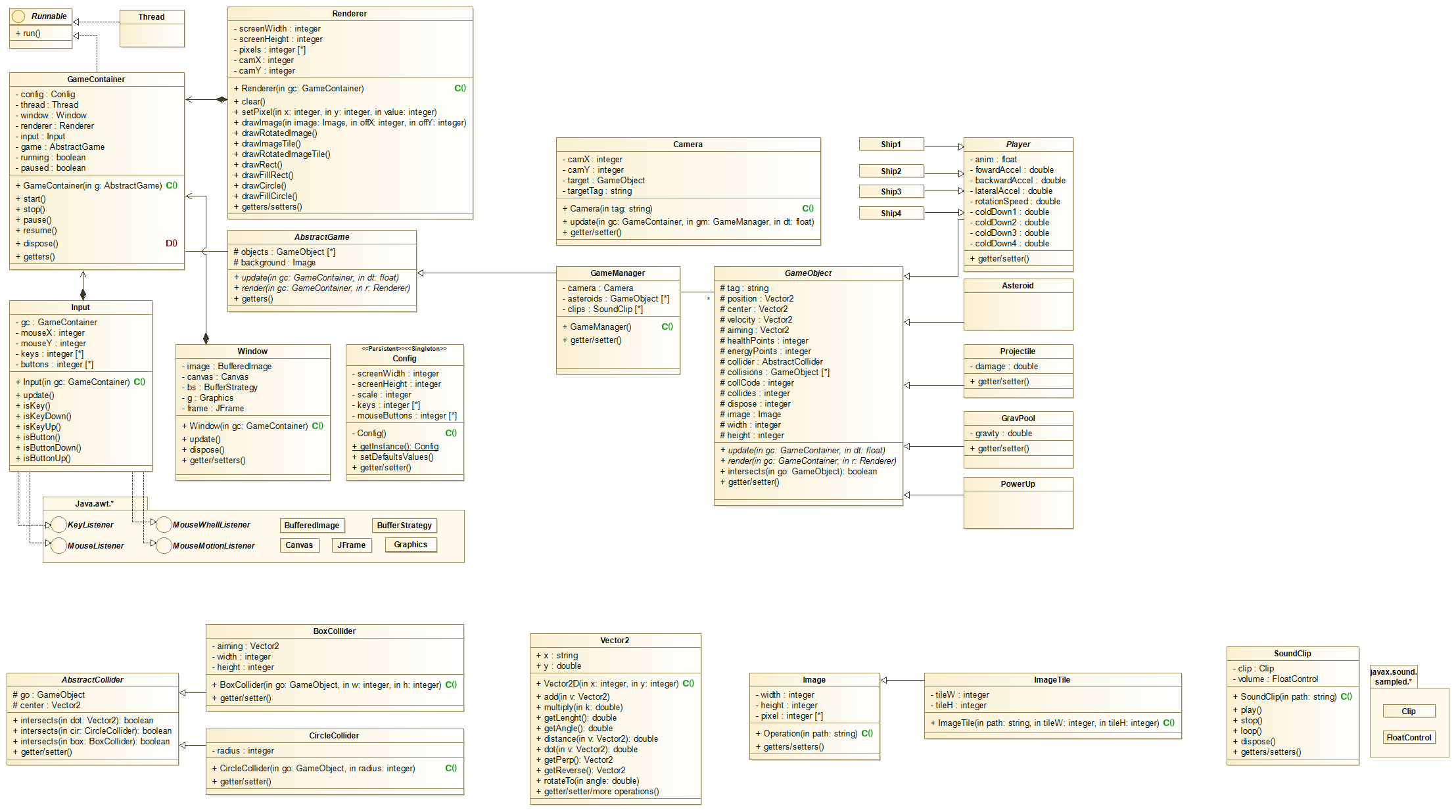
**Figura 4.2.1. - Clases esenciales del motor gráfico. Entrada / salida y renderizador**

****

GameContainer es la clase principal que aglutina todos los objetos del motor. En la clase Window se puede cargar el Canvas sobre el que dibujará el Renderer o JFrames para representar menús. Se pueden añadir más funciones draw de ser necesarias.

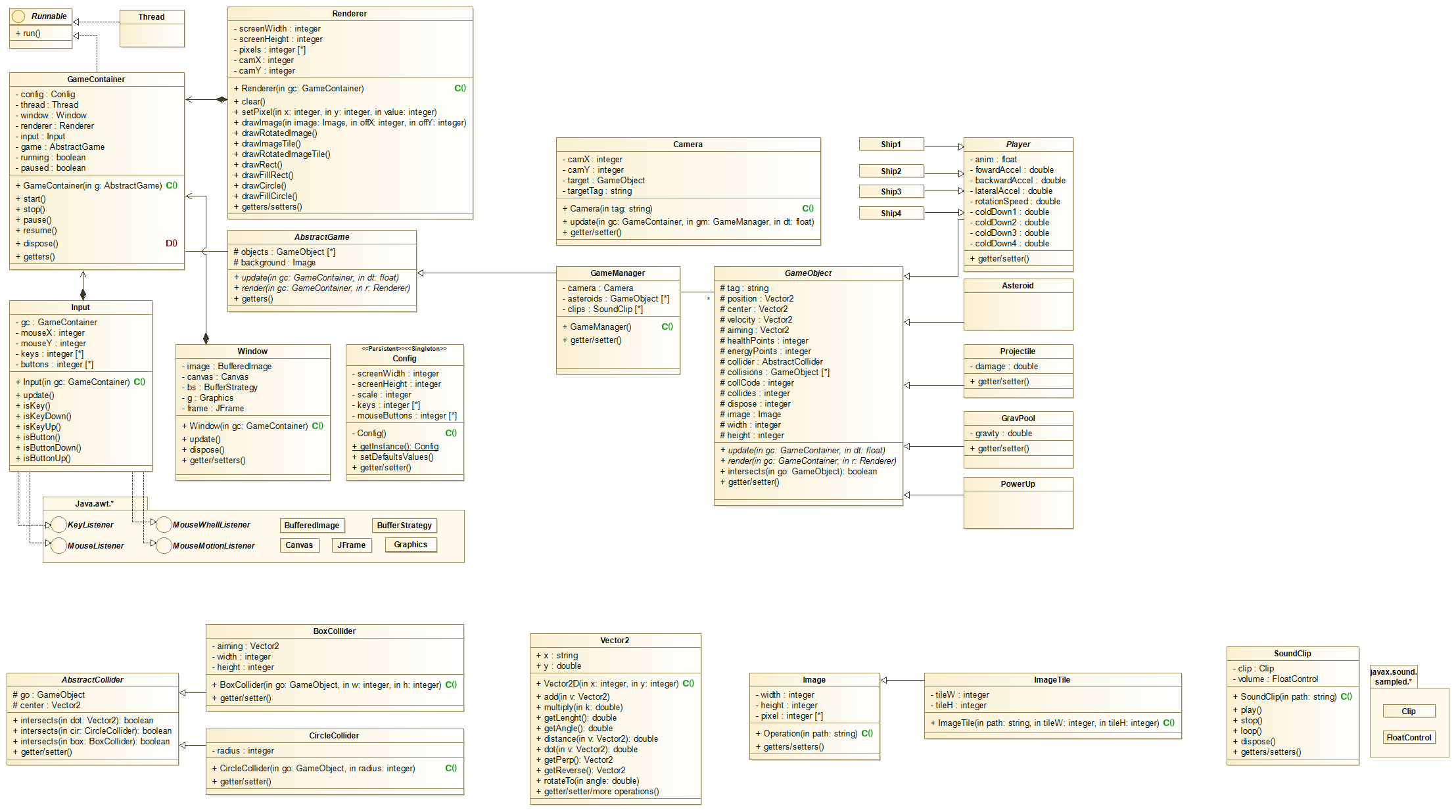
AbstractGame es el nexo de unión del código del juego con la engine.

**Figura 4.2.2. - Objetos que formatean PNGs para su uso en los métodos del renderizador.**

****

El renderizado será por software y trabajará con vectores de pixeles. ImageTile sirve para Sprites, básicamente es una matriz de imágenes para representar animaciones.

**Figura 4.2.3. - Objeto para la reproducción de audio de formato WAVs**

****

Este objeto no depende del motor y

puede ser utilizado desde cualquier

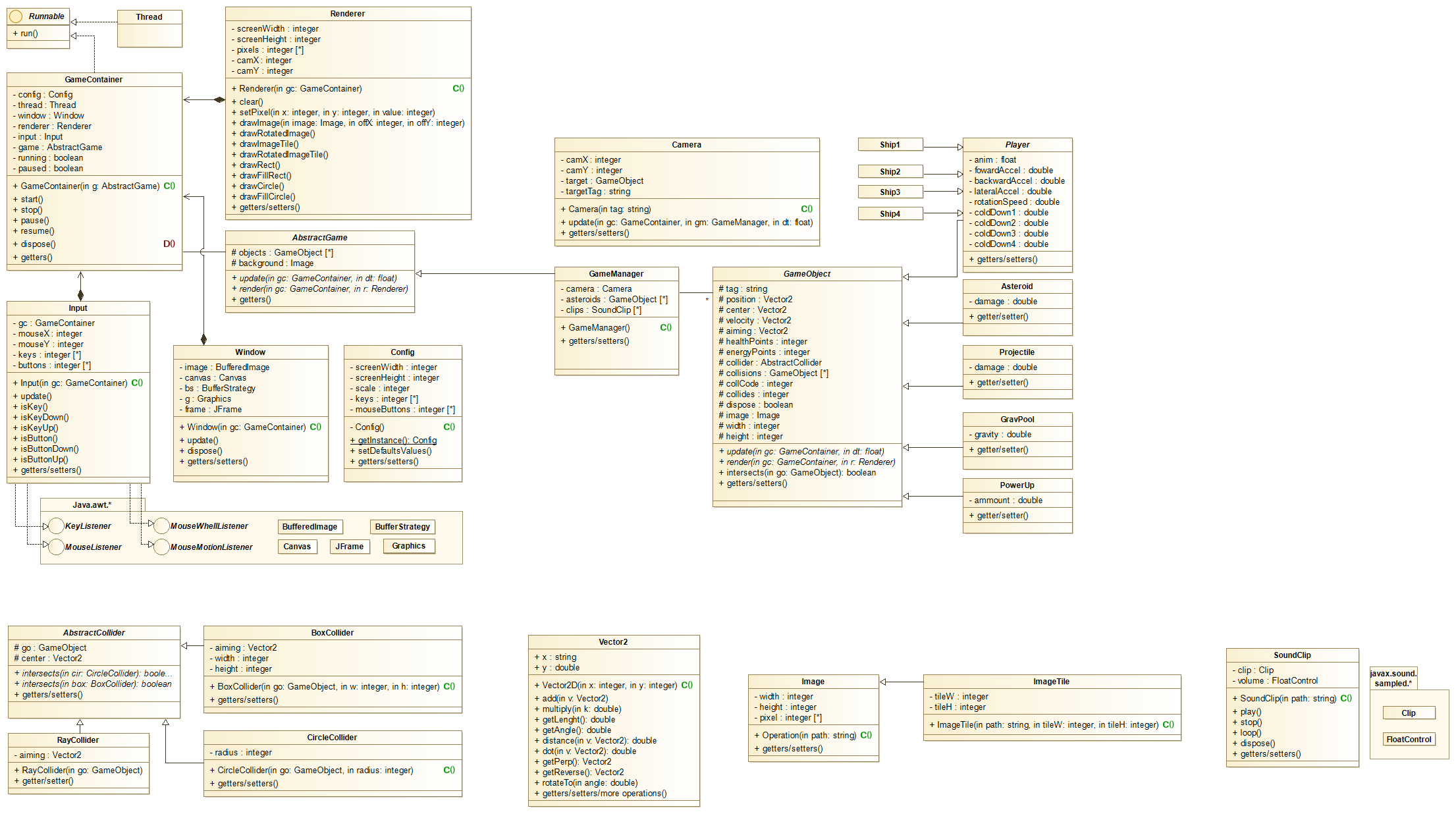
clase.

Puede reproducir en bucle para música

ambiental o cargarse en memoria para

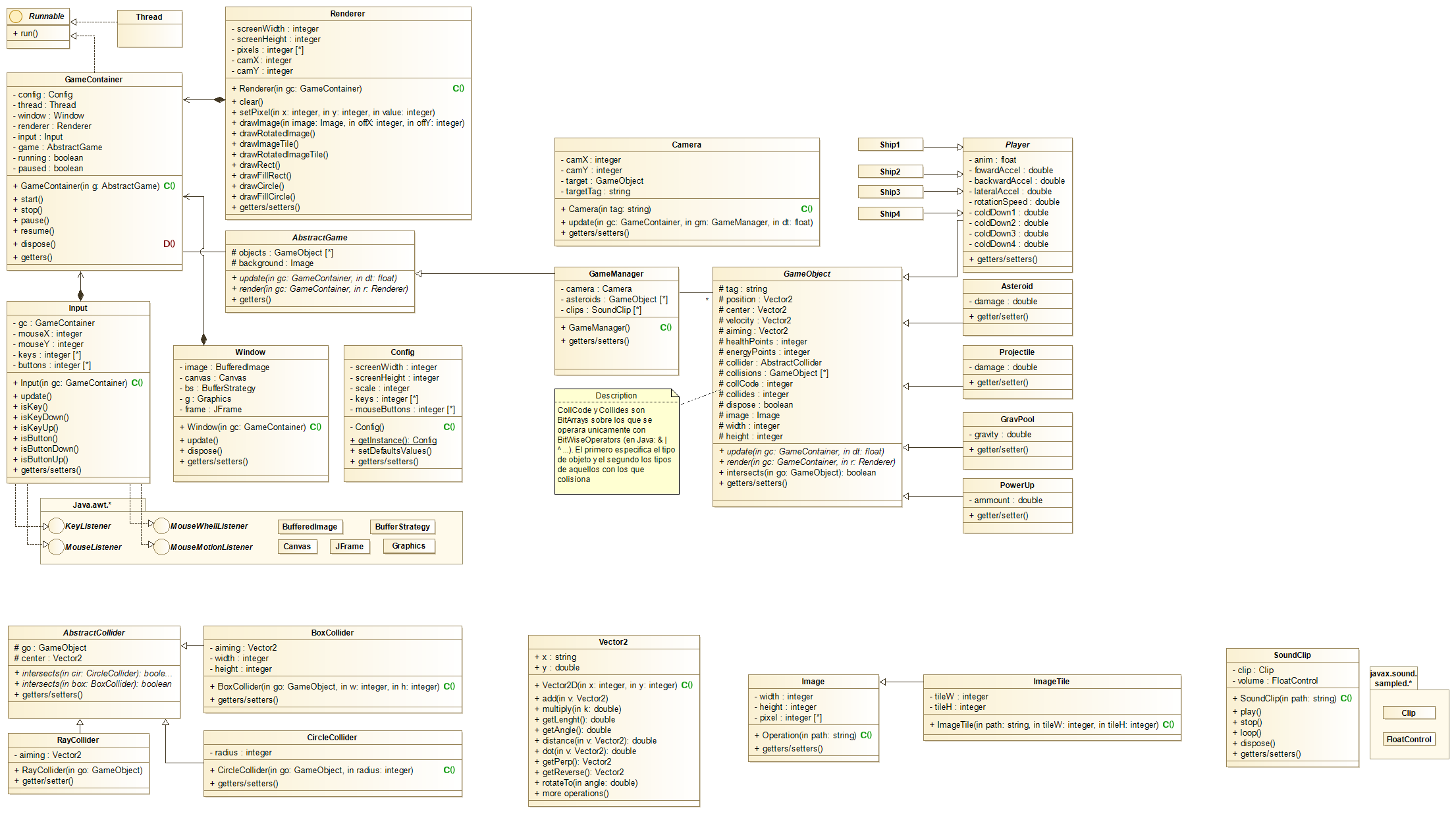
los efectos (disparos, explosiones...).

**Figura 4.2.4. - Colisionadores. Mecanismo lógico para detectar si dos formas intersectan**



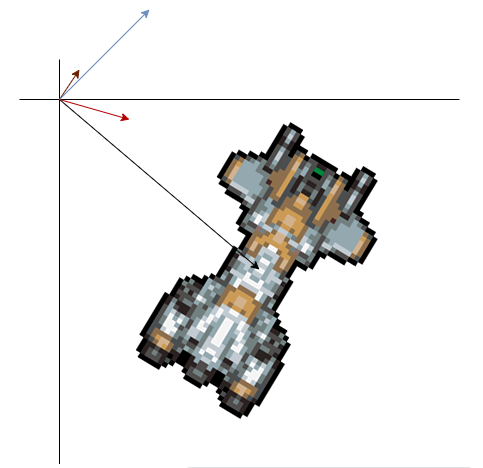
Todos los objetos del juego contaran con al menos uno, las formas que soportara este motor serán rayo (recta infinita), rectángulo y circulo (con radio 0 es un punto). Al Igual que con la clase de renderizado se pueden añadir más formas según las necesidades del juego. En un principio debería ser más que suficiente con esas 3.

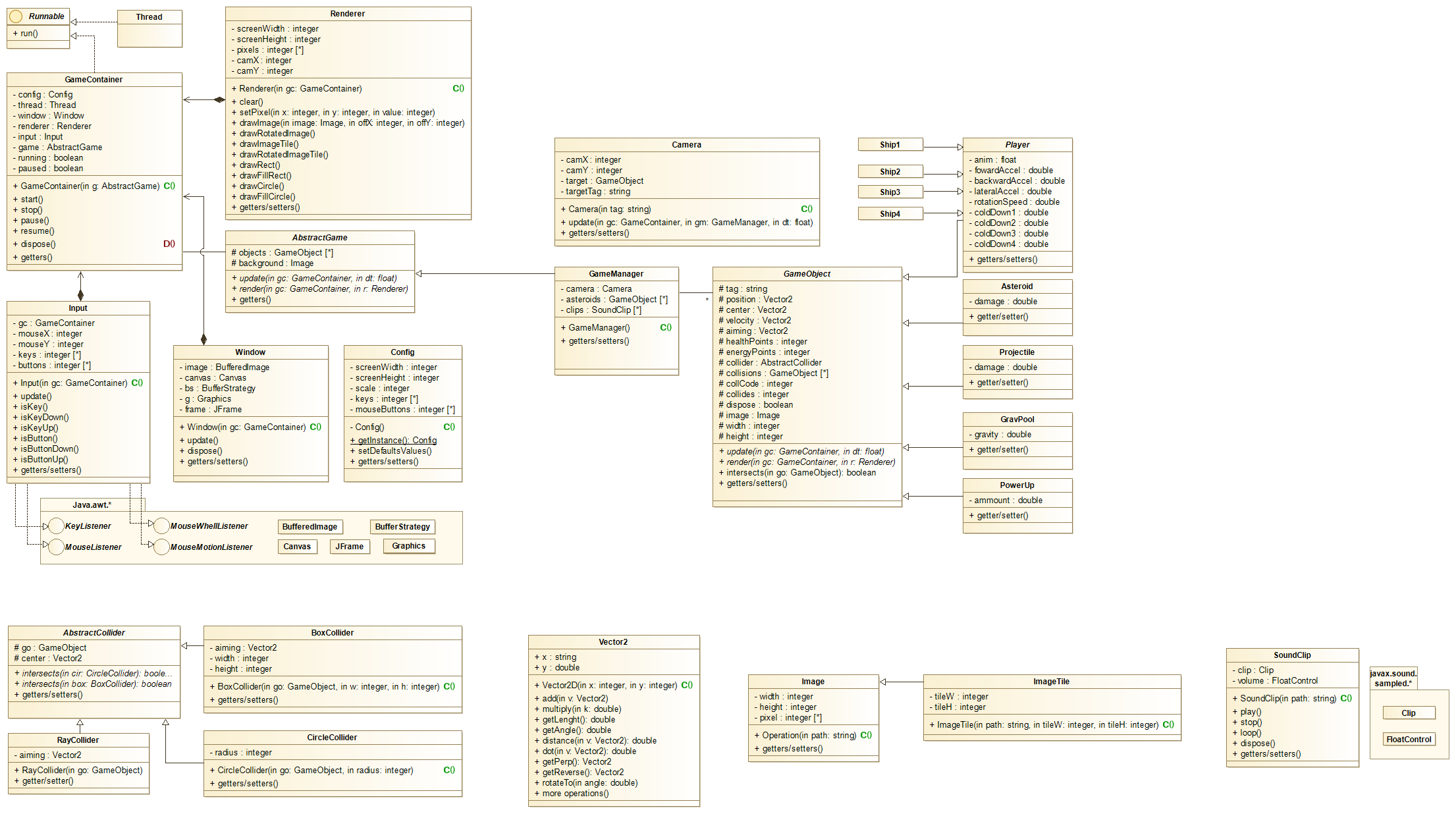
**Figura 4.2.5. - Clases que representan el código relativo a las mecánicas del juego.**



GameManager hereda de AbstractGame, es un lista de GameObjects y se encarga de avisar al motor sobre la actualización, colisiones y dibujado de los mismos.

**Figura 4.2.6. - Clase auxiliar de vectores bidimensionales. Facilita los cálculos en GameObjets**

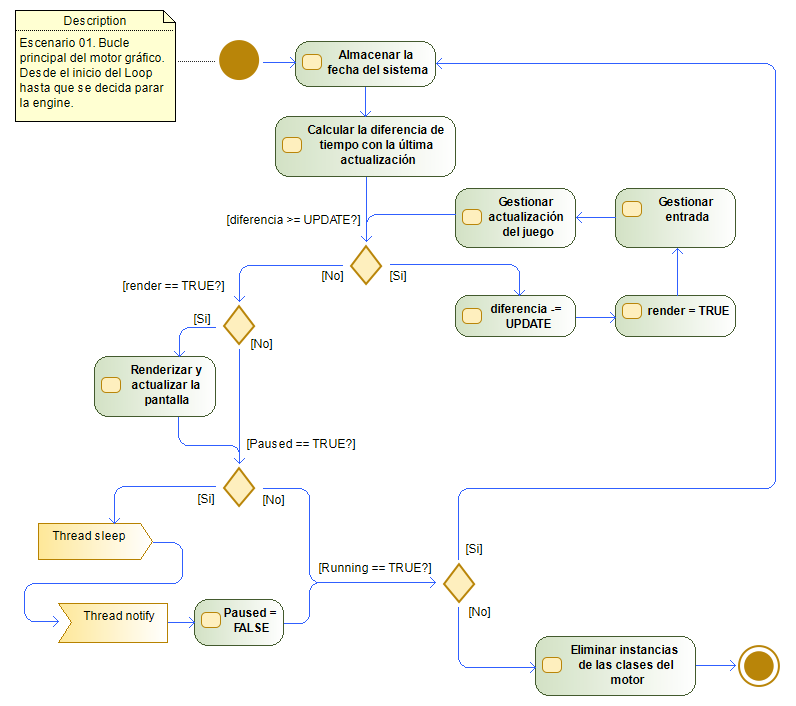




Cada GameObject posee 4 vectores: position (rojo), es una referencia para dibujar el objeto junto al vector normalizado aiming (marrón), ángulo. El vector center (negro) indica su centro y velocity (azul) se añadirá a position en la próxima simulación.

## 4.3.- Diagrama de actividad

**Figura 4.3.1. - Bucle del motor de la clase GameContainer (aka. Game Loop)**



Básicamente el motor esta en todo momento comprobando la fecha del sistema e iterando, tras un tiempo transcurrido establecido por la constante UPDATE, ejecuta el código que actualiza el mecanismo de entrada y realiza todos los cálculos pertinentes para la ejecución del juego. A continuación muestra los cambios en pantalla.

Este diseño permite que si un bucle tarda más de 2 \* UPDATE segundos en responder, se lleven a cabo más de una actualización del juego y una sola para la pantalla (el renderizador es el principal candidato a ser el cuello de botella del programa).

Por último, se comprobará si se desea pausar/reanudar o detener el motor.