Análisis de proyecto

El proyecto el cual voy a analizar es: https://github.com/svanimpe/fx-game-loops

Paquete Game:

Este código define la clase "Ball", que representa una pelota en un juego utilizando la biblioteca JBox2D y JavaFX.

La clase "Ball" extiende la clase "Circle" de JavaFX, lo que significa que hereda las propiedades y métodos de la clase "Circle", que se utiliza para dibujar y mostrar la pelota en la pantalla.

La pelota tiene un cuerpo físico asociado a ella, representado por el objeto "body" de la clase "Body" de JBox2D. Este cuerpo físico es necesario para simular la física y el movimiento de la pelota en el mundo del juego.

En el constructor de la clase, se crea el cuerpo físico de la pelota utilizando la clase "BodyDef" de JBox2D. Se establece el tipo del cuerpo como dinámico, lo que significa que será afectado por las fuerzas y colisiones en el mundo del juego.

La posición inicial de la pelota se establece de forma aleatoria dentro de los límites del ancho y alto del juego. Se utiliza la clase "Random" para generar coordenadas aleatorias.

Se define una forma de colisión para la pelota utilizando la clase "CircleShape" de JBox2D. Se establece el radio de la forma en 0.1 unidades.

Se crea una definición de accesorio ("FixtureDef") que especifica las propiedades físicas de la pelota, como la densidad y la restitución (rebote).

Se crea el cuerpo físico de la pelota en el mundo del juego utilizando el método "createBody" de la clase "World" de JBox2D, pasando la definición del cuerpo y del accesorio.

Se configuran las propiedades visuales de la pelota, como el radio y el color, utilizando los métodos heredados de la clase "Circle" de JavaFX.

El método "updatePosition" actualiza la posición visual de la pelota en función de la posición actual del cuerpo físico en el mundo del juego. Esto se logra utilizando el método "getPosition" del cuerpo físico y traduciendo las coordenadas a las coordenadas de la pantalla.

El método "interpolatePosition" realiza una interpolación suave de la posición visual de la pelota. Toma un parámetro "alpha" que indica la proporción de la interpolación. Se utiliza una combinación lineal de la posición actual de la pelota y la nueva posición calculada utilizando el método "getPosition" del cuerpo físico.

Tambien se define la clase "BallPit", que representa un juego o animación donde se sueltan un número determinado de pelotas elásticas en un pozo.

La clase implementa la interfaz "Game", lo que significa que proporciona métodos para cargar y actualizar el estado del juego.

En el método "load", se crea el suelo y las paredes del pozo utilizando objetos de la biblioteca JBox2D. Se define un cuerpo físico para el suelo y dos cuerpos físicos para las paredes izquierda y derecha del pozo. Se utilizan formas poligonales ("PolygonShape") para definir la forma de los objetos. Estos cuerpos físicos se crean en el mundo del juego pasado como parámetro. Además, se crea un número determinado de pelotas ("Ball") y se agregan a la lista "balls". Cada pelota se agrega al panel ("pane") proporcionado y también se almacena en la lista.

En el método "updatePositions", se actualiza la posición de todas las pelotas en la lista. Esto se logra llamando al método "updatePosition" de cada objeto "Ball".

En el método "interpolatePositions", se realiza una interpolación suave de la posición de todas las pelotas en la lista. Se llama al método "interpolatePosition" de cada objeto "Ball", pasando el parámetro "alpha" que indica la proporción de interpolación.

El método "toString" simplemente devuelve una representación en forma de texto del objeto "BallPit".

En resumen, este código define la clase "BallPit" que representa un juego o animación de un pozo de pelotas elásticas. Permite cargar el juego, actualizar las posiciones de las pelotas y realizar una interpolación suave de las posiciones. Utiliza la biblioteca JBox2D para simular la física y utiliza JavaFX para mostrar las pelotas en un panel.

Por ultimo este código define una interfaz llamada "Game" que establece un conjunto de métodos que deben implementarse en una clase que represente un juego o una animación.

La interfaz "Game" tiene tres métodos:

1. "load": Este método carga el juego en el mundo proporcionado. Toma como parámetros el mundo físico ("World") en el que se ejecutará el juego, un panel ("Pane") en el que se colocarán los elementos visuales del juego, y el número de objetos ("nrOfObjects") que se utilizarán para controlar la complejidad del juego. Este método se encargará de configurar el entorno del juego, crear los objetos necesarios y agregarlos al mundo físico y al panel.

2. "updatePositions": Este método actualiza la posición de todos los nodos (elementos) en el juego en función de su estado físico. En el contexto de un motor de física como JBox2D, esto implica actualizar las posiciones de los cuerpos físicos en el mundo. Este método se

llama periódicamente para mantener actualizadas las posiciones de los elementos en el juego.

3. "interpolatePositions": Este método realiza una interpolación suave de la posición de cada nodo en el juego entre su valor actual y su siguiente valor, según lo indicado por su estado físico. El parámetro "alpha" representa la proporción de interpolación y se utiliza para calcular la posición intermedia. Este método se utiliza para lograr animaciones fluidas al suavizar los movimientos entre dos estados físicos consecutivos.

En resumen, la interfaz "Game" define la estructura básica y los métodos que deben estar presentes en una clase que represente un juego o una animación. Los métodos "load", "updatePositions" e "interpolatePositions" son responsables de cargar el juego, actualizar las posiciones y realizar interpolaciones suaves, respectivamente. Estos métodos deben implementarse en la clase que implemente esta interfaz para proporcionar la lógica específica del juego o la animación.

Paquete Loop:

Es el paquete donde se usa la mayoria del patron el cual escogí, que como lo dice el titulo es game-loop.

Al verlo un poco por encima, nos encontramos con que en la clase Gameloop.java
Este código define una clase abstracta llamada "GameLoop" que es una superclase para todas las subclases relacionadas con el bucle del juego en la biblioteca JavaFX.

La clase GameLoop hereda de la clase AnimationTimer de JavaFX, lo que implica que se utiliza para crear bucles de animación en aplicaciones gráficas.

El objetivo principal de esta clase abstracta es proporcionar una propiedad común llamada "maximumStep" (paso máximo) y los métodos para acceder y modificar dicho valor. La variable "maximumStep" se establece inicialmente en el valor máximo posible de un número de punto flotante (Float.MAX_VALUE), lo que indica que no hay un límite máximo establecido.

En resumen, esta clase GameLoop proporciona una base para implementar bucles de juego en JavaFX y permite establecer un límite máximo para el paso de tiempo en cada iteración del bucle.

Luego nos encontramos con la clase “VariableSteps.java” el cual es una subclase que extiende la clase "GameLoop" y proporciona una implementación específica del bucle de juego utilizando pasos de tiempo variables.

La clase "VariableSteps" tiene tres parámetros en su constructor:

- "updater": es un objeto Consumer<Float> que representa una función o método que se llamará para actualizar el estado del juego en cada iteración del bucle. Recibe como argumento la cantidad de tiempo transcurrido desde la última actualización.

- "renderer": es un objeto Runnable que representa una función o método que se llamará para renderizar o mostrar la representación visual del juego en cada iteración del bucle.
- "fpsReporter": es un objeto Consumer<Integer> que representa una función o método que se llamará para informar sobre los cuadros por segundo (FPS) del juego en intervalos regulares.

El método principal de esta clase es "handle", que es una implementación del método abstracto de la clase AnimationTimer. Este método se ejecuta en cada fotograma del juego y realiza las siguientes acciones:

- Calcula el tiempo transcurrido desde el último fotograma en segundos.
- Limita el tiempo transcurrido al valor máximo establecido por la superclase GameLoop.
- Llama al objeto "updater" pasándole el tiempo transcurrido limitado como argumento, lo que permite actualizar el estado del juego.
- Llama al objeto "renderer" para renderizar la visualización del juego.
- Realiza un seguimiento del tiempo transcurrido y el número de fotogramas desde la última actualización de los FPS.
- Si ha transcurrido al menos medio segundo, calcula los FPS promedio y llama al objeto "fpsReporter" para informar sobre los FPS.
-

Además, la clase también tiene los métodos "stop" para detener el bucle de juego y reiniciar las variables de seguimiento, y "toString" para proporcionar una representación en cadena de la clase.

En el código se implementa una subclase llamada "FixedSteps" que extiende la clase "GameLoop" y proporciona una implementación específica del bucle de juego utilizando pasos de tiempo fijos.

Al igual que la clase "VariableSteps", la clase "FixedSteps" tiene tres parámetros en su constructor (Los parametros hacen lo mismo que en “VariableSteps” con la unica difrencia de “updater”):

- "updater": Recibe como argumento el tamaño fijo del paso de tiempo.
- "renderer"
- "fpsReporter"

El código define una constante llamada "timeStep" que indica el tamaño fijo del paso de tiempo, establecido en 0.0166 segundos (aproximadamente 60 FPS).

El método principal de esta clase es "handle", que es una implementación del método abstracto de la clase AnimationTimer. Este método se ejecuta en cada fotograma del juego y realiza acciones similiares a las que se menciono en “VariableSteps”, pero adicionando las siguientes:

- Acumula el tiempo transcurrido en la variable "accumulatedTime".

- Utiliza un bucle while para ejecutar el objeto "updater" con el tamaño fijo del paso de tiempo (timeStep) tantas veces como sea necesario para "ponerse al día" con el tiempo acumulado.
-

Además, la clase también tiene los métodos "stop" para detener el bucle de juego y reiniciar las variables de seguimiento, y "toString" para proporcionar una representación en cadena de la clase.

Al igual que las clases anteriores, la clase "FixedStepsWithInterpolation" tiene varios parámetros en su constructor:

- "updater": Recibe como argumento el tamaño fijo del paso de tiempo.
- "renderer"
- “fpsReporter"
- "interpolater": un objeto Consumer<Float> que representa una función o método utilizado para realizar la interpolación entre los fotogramas del juego. Recibe como argumento un valor de interpolación que indica la proporción de tiempo transcurrido desde el último fotograma.

El código define una constante llamada "timeStep" que indica el tamaño fijo del paso de tiempo, establecido en 0.0166 segundos (aproximadamente 60 FPS).

El método principal de esta clase es "handle", que es una implementación del método abstracto de la clase AnimationTimer. Este método se ejecuta en cada fotograma del juego y realiza las siguientes acciones:

- Acumula el tiempo transcurrido en la variable "accumulatedTime".
- Comprueba si el tiempo acumulado es menor que el tamaño fijo del paso de tiempo. Si es así, significa que ha habido una brecha en la actualización de los fotogramas y se debe realizar una interpolación. Calcula el tiempo restante necesario para alcanzar el siguiente paso de tiempo y el factor de interpolación basado en ese tiempo restante. Luego, llama al objeto "interpolater" con el factor de interpolación y sale del método.
- Si el tiempo acumulado es igual o mayor que dos veces el tamaño del paso de tiempo, significa que ha habido una brecha tan grande que el juego debe ponerse al día ejecutando múltiples actualizaciones. Utiliza un bucle while para ejecutar el objeto "updater" con el tamaño fijo del paso de tiempo tantas veces como sea necesario para ponerse al día.
- Llama al objeto "renderer" para renderizar la visualización del juego.
- Ejecuta nuevamente el objeto "updater" con el tamaño fijo del paso de tiempo para mantener la coherencia.
- Calcula un factor de interpolación (alpha) basado en el tiempo acumulado y el tamaño del paso de tiempo. Llama al objeto "interpolater" con el factor de interpolación para realizar la interpolación entre los fotogramas.

Además, la clase también tiene los métodos "stop" para detener el bucle de juego y reiniciar las variables de seguimiento, y "toString" para proporcionar una representación en cadena de la clase.

En resumen, la clase "FixedStepsWithInterpolation" implementa un bucle de juego utilizando pasos de tiempo fijos junto con interpolación. Esto permite mantener una velocidad constante de actualización y renderizado del juego, incluso cuando hay brechas en el tiempo entre los fotogramas. La interpolación se utiliza para suavizar las transiciones entre los fotogramas y mejorar la fluidez de la animación.

Patrón “GameLoop”:

El patron "GameLoop" se utiliza en los códigos anteriores para implementar un bucle de juego eficiente. Un bucle de juego es una parte fundamental de cualquier juego o animación, ya que permite actualizar el estado del juego y renderizar los elementos visuales de manera continua.

El propósito de "GameLoop" es proporcionar una estructura para ejecutar el bucle de juego de manera controlada y consistente. Esta clase encapsula la lógica del bucle de juego, incluyendo el control del tiempo, la llamada a los métodos de actualización y renderizado, y la sincronización con la frecuencia de cuadros (frames per second - FPS) objetivo.

Usar "GameLoop" separada permite mantener una estructura limpia y modular en el código del juego. Proporciona una separación clara entre la lógica del juego y la implementación del bucle de juego, lo que facilita la organización y el mantenimiento del código.

Además, se puede implementar diferentes enfoques para el bucle de juego, como el uso de pasos de tiempo fijos o variables, la interpolación de posiciones, el manejo de la frecuencia de cuadros y otros aspectos relacionados con la gestión del tiempo en el juego. Esto brinda flexibilidad para adaptar el bucle de juego según las necesidades del juego específico y optimizar el rendimiento.

En resumen, el uso del patrón es proporciona una estructura y funcionalidad comunes para ejecutar el bucle de juego, lo que facilita el desarrollo del juego, la sincronización del tiempo y el logro de una actualización y renderizado suaves.