```java
package com.blindtigergames.werescrewed.entity.tween;

import aurelienribon.tweenengine.Timeline;
import aurelienribon.tweenengine.Tween;
import aurelienribon.tweenengine.TweenEquation;
import aurelienribon.tweenengine.TweenEquations;

import com.blindtigergames.werescrewed.entity.mover.TimelineTweenMover;
import com.blindtigergames.werescrewed.entity.platforms.Platform;

/**
 * Builds simple paths for platforms to move on. Use pixels for positions and
 * all positions are relative to the platform's spawning location. Support for
 * rotation depends on requests for it's use. Ask Stew if you really want
 * rotation support.
 *
 * @author stew
 *
 */
public class PathBuilder {
    private Timeline timeline;
    private Platform platformToMove;
    private TweenEquation easeFunction;
    private boolean repeatYoyo;
    private float timelineDelay;
    private float delay;
    private int loopCount;

    public PathBuilder( ) {
        reset( );
    }

    public void reset( ) {
        timeline = null;
        platformToMove = null;
        easeFunction = TweenEquations.easeNone;
        repeatYoyo = false;
        timelineDelay = 0f;
        delay = 0f;
        loopCount = Tween.INFINITY;
    }

    /**
     * start your path. you better use PathBuilder.platform() before you set a
     * target.
```



```java
     *
     * @return
     */
    public PathBuilder begin( ) {
        this.timeline = Timeline.createSequence( );
        return this;
    }

    /**
     * start your path using this and the path will apply to this platform
     *
     * @param platform
     *            platform to apply path to.
     * @return
     */
    public PathBuilder begin( Platform platform ) {
        this.platformToMove = platform;
        return this.begin( );
    }

    /**
     * set the target of the next target on the path
     *
     * @param platformToMove
     * @return
     */
    public PathBuilder platform( Platform platformToMove ) {
        this.platformToMove = platformToMove;
        return this;
    }

    /**
     * Set the ease of all subsequent targets on this path.
     *
     * @param easeFunction
     * @return
     */
    public PathBuilder ease( TweenEquation easeFunction ) {
        this.easeFunction = easeFunction;
        return this;
    }

    /**
     * set a new target on the path for the platform. Happens after the target
     * before and before the target after.
```

```
     *
     * @param xPixel
     *            PIXELS!!
     * @param yPixel
     *            PIXELS!!
     * @param time
     *            time to reach target from prev target (speed)
     * @return
     */
    public PathBuilder target( float xPixel, float yPixel, float time ) {
        timeline.push( Tween
                .to( platformToMove, PlatformAccessor.LOCAL_POS_XY, time )
                .delay( delay ).target( xPixel, yPixel )
                .ease( this.easeFunction ).start( ) );
        return this;
    }

    public PathBuilder repeatYoyo( boolean wantYoyoRepeat ) {
        this.repeatYoyo = wantYoyoRepeat;
        return this;
    }

    /**
     * the delay for each waypoint on the timeline 0 by default. applies to
     * every waypoint afterwards unless set back to 0.
     *
     * @param pathDelay
     * @return
     */
    public PathBuilder delay( float pathDelay ) {
        delay = pathDelay;
        return this;
    }

    /**
     * After each timeline loops, this delay will follow. 0 by default
     *
     * @param timelineDelay
     * @return
     */
    public PathBuilder timelineDelay( float timelineDelay ) {
        this.timelineDelay = timelineDelay;
        return this;
    }
```



```
    /**
     * set the number of loops of this timeline. infinity by default.
     *
     * @param loopCount
     * @return
     */
    public PathBuilder loops( int loopCount ) {
        this.loopCount = loopCount;
        return this;
    }

    /**
     * builds and returns the path you created. Pass this into a timeline mover.
     *
     * @return
     */
    public TimelineTweenMover build( ) {
        if ( repeatYoyo ) {
            timeline = timeline.repeatYoyo( loopCount, timelineDelay );
        } else {
            timeline = timeline.repeat( loopCount, timelineDelay );
        }

        return new TimelineTweenMover( timeline.start( ) );
    }

}
```