Java FX: Animation Timer

Learning Goal: Make your project dynamic by creating a game state that changes over time using a game loop

The Game Loop

- We are going to implement a game loop (an infinite loop that updates the game objects and renders the scene to the screen – usually 60 times per second)
- Using the AnimationTimer class is a bit tricky, but for our simple examples you can follow this tutorial.
- We are going to modify our "Hello, World" example to have the Earth orbiting around the Sun against a background of stars
- Important note: There are other ways to implement a game loop in JavaFX such as the Timeline class

Try this:

- Create a new class in the same overall project called TheGameLoop with the following import statements:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.Image;
import javafx.animation.AnimationTimer;
```

- And you should be used to this class statement and main method now:

```
// Animation of Earth rotating around the sun. (Hello, world!)
public class TheGameLoop extends Application
{
    public static void main(String[] args)
    {
        launch(args);
    }
```

- Add the following code which is very similar to our last programming practice. Remember to move sun.png and space.png into your src folder. They are underlined in yellow at this stage because we haven't actually displayed them anywhere yet.

```
@Override
public void start(Stage theStage)
{
    theStage.setTitle( "AnimationTimer Example" );

    Group root = new Group();
    Scene theScene = new Scene( root );
    theStage.setScene( theScene );

    Canvas canvas = new Canvas( 512, 512 );
    root.getChildren().add( canvas );

    GraphicsContext gc = canvas.getGraphicsContext2D();

    // we need 3 images for this class
    // remember to move them into the src folder!
    Image earth = new Image( "earth.png" );
    Image sun   = new Image( "sun.png" );
    Image space = new Image( "space.png" );



    theStage.show();
}
```

- Run at this point to make sure there are no issues with your code (don't forget the very last close bracket } for the class)

A) Now add the following lines right before `theStage.show();`

```
new AnimationTimer()
{
    double x = 0;
    double y = 0;
    public void handle(long currentNanoTime)
    {

        // Clear the canvas
        gc.clearRect(0, 0, 512,512);

        x = x+1;
        // background image clears canvas
        gc.drawImage( space, 0, 0 );  //draw the background
        gc.drawImage( earth, x, y );  // draw the earth in a new location each tick
        gc.drawImage( sun, 196, 196 );
    }
}.start();
```

- Run this program and see what it does.

B) Make your animation a little bit more complicated! Play around with the values of x and y, and run your program to see the results:

For instance:

```
x = x+1;
y = y+1;
```

or

```
double x = 1;
double y = 1;

x = x*1.1;
y = y*1.1;
```

C) Now let's make the earth go in a circle. This is kind of complicated as we will need to use sine and cosine to create circular movement. Don't worry too much about the math behind this example right now.

```
final long startNanoTime = System.nanoTime();

new AnimationTimer()
{
    public void handle(long currentNanoTime)
    {
        double t = (currentNanoTime - startNanoTime) / 1000000000.0;

        double x = 232 + 128 * Math.cos(t);  // move the x coordinate on a cosine path
        double y = 232 + 128 * Math.sin(t);  // move the y coordinate on a sine path

        // Clear the canvas
        gc.clearRect(0, 0, 512,512);

        // background image clears canvas
        gc.drawImage( space, 0, 0 );  //draw the background
        gc.drawImage( earth, x, y );  // draw the earth in a new location each tick
        gc.drawImage( sun, 196, 196 ); // draw the sun in the same place every time
    }
}.start();
```

- Run your program!