HOVER ZOOM: MOUSE EVENTS & EXTRACTING PART OF AN IMAGE IN JAVAFX

INTRODUCTION

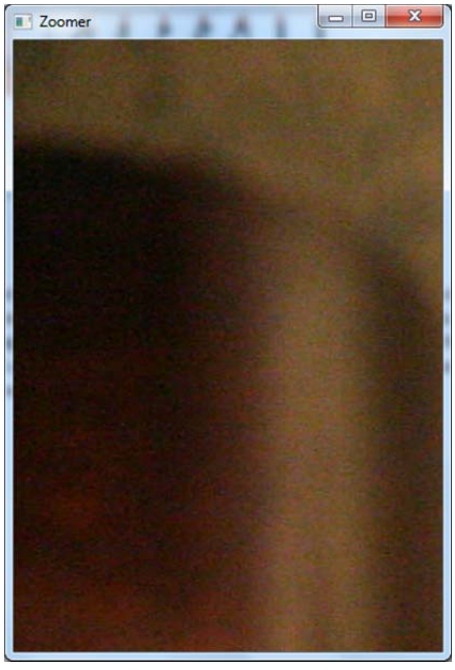
Previous tutorials have focussed on using keyboard events. Quite often we need to make use of the mouse in an application. This simple application will make use of a mouse over event to trigger a method that extracts part of an image.

Extracting parts of an image are useful in applications where you don't want delay in displaying images, for example extracting frames from a spritesheet, or a web app where you don't want the user to wait for a new image to be retrieved from the server every time they rollover an on-screen object..

Example spritesheet:

GETTING STARTED

Create a new JavaFX project. Title the stage "Zoomer". Make sure the scene has a size of 350x 500. Use a HBox as the root layout. Create two images using the `Image` class, and two viewports using `ImageView`. The first will be for the original image, and the second for the target, zoomed in image. Copy the graphic turntable.jpg into the project folder and assign it to the first ImageView. You should end up with something like the window below. My basic class is overleaf.



```
package zoomer;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Zoomer extends Application {

    @Override
    public void start(Stage primaryStage) {
        Image imgOriginal = new Image(getClass().getResourceAsStream("turntable.jpg"));
        ImageView ivOriginal = new ImageView(imgOriginal);
        final ImageView ivTarget = new ImageView();

        VBox root = new VBox();

        root.getChildren().add(ivOriginal);
        root.getChildren().add(ivTarget);

        Scene scene = new Scene(root, 350, 500, Color.CORNFLOWERBLUE);

        primaryStage.setTitle("Zoomer");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

}
```

Adjusting an ImageView

The first thing that would be useful to us here is to be able to see the whole image.

We can configure an `ImageView` via various methods. Try the following, saving and re-running your project each time to see what happens. Add the code after you have initialised your `ImageView` with the `Image` of the turntable.

```
ivOriginal.setFitWidth(345);
ivOriginal.setPreserveRatio(true);
ivOriginal.setSmooth(true);
```

Adding a mouse listener

In order to bring up the zoomed in image for the current region of the image, we need to add a mouse listener. This is similar to the process of adding a key listener. We will create an anonymous inner class to handle the event. I have placed this below where I show my stage.

Note: if you are using Java 8, you will be prompted to use a **Lambda expression**. The use of an anonymous inner class to handle events was to streamline code. This was still considered too bulky, so Java 8 now offers lambda expressions to simplify the code further.

```
primaryStage.show();

ivOriginal.setOnMouseMoved(new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent event) {
      //Not implemented yet
      }
});
```

CREATING A VIEWPORT INTO AN IMAGE

A viewport into an image is akin to a window. It lets us look at a specified part of the image. To do this I draw a rectangle at a specified point on the image, then add this to the root node of my scene graph. In this case, the specified point is the mouse X and Y on the image. That makes the handle for my mouse listener now look like:

```
int x = (int) event.getX()*10;
int y = (int) event.getY()*10;
ivTarget.setImage(imgOriginal);
Rectangle2D viewportRect = new Rectangle2D(x, y, 345, 230);
ivTarget.setViewport(viewportRect);
```

The getX and getY return a double, so they need to be typecast into an integer. I have also multiplied them by ten, as the image has been shrunk by a factor of ten to fit into the thumbnail. Without the multiplication, you would only see a very small segment of the image in the zoom panel.

This is a simple way of looking at a segment of an image. The problem with this method is that it will load the entire image into memory and only show a part of it. This is a very expensive operation, particularly when dealing with lots of images.

EXTRACTING PARTS OF AN IMAGE

An alternative method is to extract only the parts of the image that we want to use. This can be done by using a `PixelReader` to read the pixels that we want and extracts parts into a `WritableImage`. This can then be placed into the target `ImageView`.

My new handler looks like:

```
int x = (int) event.getX()*10;
int y = (int) event.getY()*10;
PixelReader reader = imgOriginal.getPixelReader();
Image currentFrame = new WritableImage(reader,x , y, 300, 250);
ivTarget.setImage(currentFrame);
```

IMPROVEMENTS

Using a PixelReader was primarily driven by a concern for memory. Replace the original image with a thumbnail.

You may have noticed that when the mouse goes close to the left edge, you do not see the left edge of the zoomed in image. Experiment with subtracting 150 from x (the midpoint of the thumbnail) and similar from y. Check and adjust x and y so if they go below a particular they are always set to 0.

Add a similar adjustment to the right edge.