Essential JavaFX

Graphics

Tobias Andersson Gidlund

tobias.andersson.gidlund@lnu.se

November 22, 2012

Graphics

- Compared to Swing, JavaFX is far better at handling graphics.
- Two of the reasons for this is *Prism* and *Glass*.
  - Prism is a hardware accelerated graphics pipeline.
  - Glass is the new windowing toolkit.
- Underneath Prism, either DirectX or OpenGL is used (and therefore hardware accelerated).
  - If no compatible hardware is found, Java2D will do the rendering.
- Glass is using parts of the native platform for windowing, but also has its own part.
  - The possibility to interact is greater than before.

Quantum Toolkit

- The Prism and Glass parts are not directly reachable through JavaFX.
- Instead, the *Quantum Toolkit* is the public part of graphics in JavaFX.
- In most cases, though, this is used via the scene graph of an application.
- Since the *Media* and *Web* engines are implemented alongside Prism and Glass, it is possible to create both desktop and web applications using the Quantum Toolkit.

Images

- As discussed in the previous lecture, images are displayed as a two part action.
- The image itself is loaded into an object of `Image` type.
- The part of an image to be seen is then defined by a viewport for an `ImageView`.
- It is also possible to *transform* the image, either directly using methods to the `ImageView` or through separate classes.
  - Translation – changing the position of the image.
  - Rotating – along a pivot.
  - Scaling
  - Shearing – moves just one axis.
- Notice that JavaFX supports this as both 2D and 3D functions.

Translation

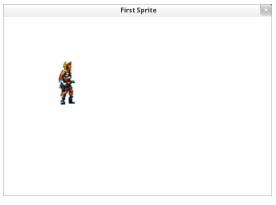
- The first example will show an translation of position.
- Using the methods `setTranslateX` and `setTranslateY` it is possible to decide the position of an image.
  - This *can* be used to animate an image, but we will look at better ways.
- The example, as well as several others in this lecture, will use a *sprite map*.
  - An image with several smaller images where each image is part of a movement.
  - Popular during the 80s and 90s for 2D games.
- The sprites are shamelessly taken from the game *Super The Empire Strikes Back* for the Super Nintendo.

The sprite map

The code

```
// imports omitted

public class JavaFX_L3_Sprite1 extends Application {

    @Override
    public void start(Stage primaryStage) {
        final Image image = new Image(getClass()
                          .getResourceAsStream("lukeskywalker.png"));
        final ImageView sprite = new ImageView(image);
        sprite.setViewport(new Rectangle2D(0, 0, 50, 50));
        sprite.setFitHeight(100);
        sprite.setPreserveRatio(true);
        sprite.setTranslateX(100);
        sprite.setTranslateY(100);

        Scene theScene = new Scene(new Group(sprite), 600, 400);

        primaryStage.setTitle("First Sprite");
        primaryStage.setScene(theScene);
        primaryStage.show();
    }
```


In graphics



Rotation

- For rotation, it is best to use the class `Rotate`.
- It takes three parameters:
  - The first is the number of degrees to rotate.
  - The second and third are the anchor point.
- The anchor point defines the position around which the rotation should take place.
- The transformations are then added, like effects, to the image view.


```
final Image image = new Image(getClass().getResourceAsStream("lukeskywalker.png"));
final ImageView sprite = new ImageView(image);
final ImageView secondSprite = new ImageView(image);

sprite.setViewport(new Rectangle2D(80, 635, 75, 75));
sprite.setFitHeight(100);
sprite.setPreserveRatio(true);
sprite.setTranslateX(100);
sprite.setTranslateY(100);

secondSprite.setViewport(new Rectangle2D(0, 110, 45, 50));
secondSprite.setFitHeight(100);
secondSprite.setPreserveRatio(true);
secondSprite.setTranslateX(200);
secondSprite.setTranslateY(100);

Rotate rotator1 = new Rotate(30, 50, 30);
sprite.getTransforms().add(rotator1);

Rotate rotator2 = new Rotate(90, 0, 0);
secondSprite.getTransforms().add(rotator2);

HBox layout = new HBox();
layout.getChildren().addAll(sprite, secondSprite);

Scene theScene = new Scene(layout, 600, 400);

primaryStage.setTitle("Rotating Sprites");
primaryStage.setScene(theScene);
primaryStage.show();
```


In graphics

Animations

- There are a number of built in animation classes in JavaFX.
  - This in contrast to Swing, where this was basically left to the programmer.
- The animation functionality lies in the `Animation` package, with several classes.
- Two high level categories can be seen:
  - Transitions
  - Timeline animation
- These can be further divided into different classes.

Transitions

- The main idea behind transitions is to have a change of state over time.
- This is done via an internal timeline, in contrast to other animations.
- The `Transition` class is abstract and has several concrete sub classes.
  - `FadeTransistion`
  - `RotateTransition`
  - `PathTransition`
- All of them work on `Nodes`, so most elements can be used.
  - Images, text and so on.
- All transitions set a *duration* for the internal timeline.

`FadeTransistion`

- The `FadeTransistion` makes it possible to fade a node.
- For the node to fade, the start and end values are set.
  - A `double` going from 0.0 (invisible) to 1.0 (fully visible).
- A duration is set for the entire fade, but also an increment for each step in the fade.
  - This is also a `double` from 0.0 to 1.0.
- It is also possible to set it to cycle and to reverse when at the end.
- When the transition is set, the `play()` method will start the animation.

In code

```
final Image theEmperor = new Image(getClass()
                .getResourceAsStream("darthsidious.png"));
final ImageView theImperialView = new ImageView(theEmperor);

FadeTransition fadeToBlack =
        new FadeTransition(Duration.millis(4000), theImperialView);
fadeToBlack.setFromValue(0.0);
fadeToBlack.setToValue(1.0);
fadeToBlack.setByValue(0.3);
fadeToBlack.setCycleCount(Animation.INDEFINITE);
fadeToBlack.setAutoReverse(true);
fadeToBlack.play();

Scene scene = new Scene(new Group(theImperialView), 400, 600);

primaryStage.setTitle("Wipe the out. All of them!");
primaryStage.setScene(scene);
primaryStage.show();
```


In graphics

RotateTransition

- We previously saw how it was possible to rotate an image (or any other node) using the `Rotate` class.
- It is possible to add a rotation animation by repeatedly updating the values, but it is easier to use the `RotateTransition` class.
- The object of `RotateTransition` is given values for:
  - Angle – the complete change from the initial state, 360 for a full circle (obviously).
  - A cycle count for the number of times it needs to be done.
- In the example we also set the *interpolation*.
- This can be done using either a separate class or as a method to the transition.
  - It decides the start and end movement of the transition.

The code

```
final Image itsMe = new Image(getClass()
                  .getResourceAsStream("jag.png"));
final ImageView showMe = new ImageView(itsMe);

RotateTransition snurr =
     new RotateTransition(Duration.millis(3000), showMe);
snurr.setByAngle(360);
snurr.setCycleCount(Animation.INDEFINITE);
snurr.setAutoReverse(true);
snurr.setInterpolator(Interpolator.EASE_BOTH);
snurr.play();

Scene scene = new Scene(new Group(showMe), 500, 400);

primaryStage.setTitle("Hello World!");
primaryStage.setScene(scene);
primaryStage.show();
```


In graphics

Putting it together

- To make an even more controlled animation it is also possible to inherit from `Transition`.
- In this case we will be sending an `ImageView` to the transition class.
  - Called `SpriteAnim` since we are animating sprites.
- In our sprite animation class we will shift the viewport of the `ImageView` to simulate movement.
- The main class will still set and populate the original image and view.
  - This might not be the “best” way, but still quite efficient.
- Animation is set to infinite, but it is possible to start and stop as well as pause an animation in code.

The `interpolate` method

- In our sprite animation class it is vital to override the `interpolate` method.
- This is the method that will be executed for every new frame.
- It is called more often than the duration is set for, though, so it needs to be guarded.
  - This is because this method should be called for every screen redraw.
- In our example we use the input value to the method for calculating the frame number.
  - If it has changed from previous call, that is – the duration is at end – then it will update.
- The interpolation type is set to `LINEAR` since we do not want it to slow down between changes.

The `SpriteAnim` class

```
public class SpriteAnim extends Transition {
    ImageView spriteView;
    int x_coord, y_coord, width, height;
    int count=0;
    int lastIndex;

    SpriteAnim(ImageView theIV, int x, int y, int w, int h, int l){
        spriteView = theIV;
        x_coord = x; y_coord = y; width = w; height = h;
        count = l;
        setCycleDuration(Duration.millis(1000));
        setInterpolator(Interpolator.LINEAR);
    }

    @Override
    protected void interpolate(double d) {
        final int index = Math.min((int) Math.floor(d*count),count-1);
        if(index != lastIndex)
        {
            if(x_coord < width*(count-1))
                x_coord = x_coord + width;
            else
                x_coord=0;
            spriteView.setViewport(
                    new Rectangle2D(x_coord, y_coord, width, height));
            lastIndex = index;
        }
    }
}
```


The main class

```
    public void start(Stage primaryStage) {
        final Image theImage = new Image(getClass()
                .getResourceAsStream("lukeskywalker.png"));
        final ImageView theView = new ImageView(theImage);
        theView.setViewport(new Rectangle2D(0, 50, 50, 50));
        theView.setFitHeight(100);
        theView.setPreserveRatio(true);
        final Animation anim = new SpriteAnim(theView, 0, 50, 50, 50, 9);
        anim.setCycleCount(Animation.INDEFINITE);
        anim.play();

        Scene scene = new Scene(new Group(theView), 300, 250);

        primaryStage.setTitle("Sprite 2");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```


In graphics

PathTransition

- It is also possible to set up a transition over a path.
- The path will then be defined by using a number of path classes like `MoveTo`, `LineTo` and `CubicCurveTo`.
- In the example we only set a path and let our node move over it, put it is possible to make it follow a mouse click, a key press or anything else.
  - Also notice that the previous transition is still in effect.
- When the path is set, a `PathTransition` object must be created taking the path as a parameter.
  - As well as the node to animate.
- For the transition the orientation is set, in this case to `NONE` which means that it will only follow the path.

The code

```
final Image theImage = new Image(getClass().getResourceAsStream("lukeskywalker.png"));
final ImageView theView = new ImageView(theImage);
theView.setViewport(new Rectangle2D(0, 50, 50, 50));
theView.setFitHeight(100);
theView.setPreserveRatio(true);
final Animation anim = new SpriteAnim(theView, 0, 50, 50, 50, 9);
anim.setCycleCount(Animation.INDEFINITE);
anim.play();

PathTransition thePath = new PathTransition();
Path path = PathBuilder.create()
        .elements(new MoveTo(50, 60),
        new LineTo(600, 60)
        ).build();

thePath = PathTransitionBuilder.create()
        .duration(Duration.seconds(5))
        .path(path)
        .node(theView)
        .orientation(OrientationType.NONE)
        .cycleCount(Timeline.INDEFINITE)
        .autoReverse(true)
        .build();

thePath.play();
```


In graphics

Another example

- In the following a more complex path is set.
  - Borrowed from the Internet. . .
- This example sets the orientation to `ORTHOGONAL_TO_TANGENT` which will make the image turn at curves.
- Also notice how we set a background by applying a style to the root.
  - This style sets the image to stretch to fill the background.
  - It also centres it.

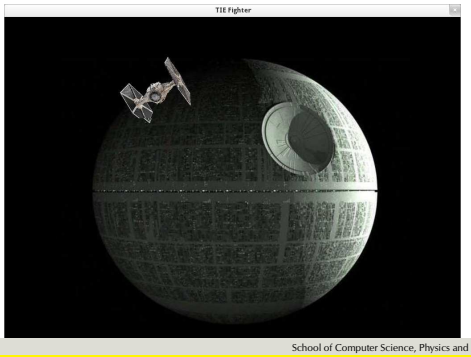
The code

```
public void start(Stage primaryStage) {
    PathTransition pathTransition = new PathTransition();

    Path path = PathBuilder.create()
            .elements(
               new MoveTo(50, 50),
               new LineTo(800, 400),
               new LineTo(50, 600),
               new CubicCurveTo(580, 0, 580, 120, 200, 120),
               new CubicCurveTo(0, 120, 0, 240, 380, 240)
            ).build();
    path.setVisible(false);

    ImageView tie = new ImageView(new Image(getClass().getResourceAsStream("tiefighter.png")));
    tie.setFitWidth(150.0);
    tie.setPreserveRatio(true);

    pathTransition = PathTransitionBuilder.create()
            .duration(Duration.seconds(10))
            .path(path)
            .node(tie)
            .orientation(OrientationType.ORTHOGONAL_TO_TANGENT)
            .cycleCount(Timeline.INDEFINITE)
            .autoReverse(true)
            .build();
```


The code, cont.

```
    primaryStage.setTitle("TIE Fighter");

    StackPane root = new StackPane();
    String image = getClass().getResource("deathstar.jpg").toExternalForm();
    root.setStyle("-fx-background-image: url('" +
            image
            + "'); -fx-background-position: center center; -fx-background-repeat: stretch;");
    root.setAlignment(Pos.TOP_LEFT);
    root.getChildren().addAll(tie, path);
    primaryStage.setScene(new Scene(root, 1024, 768));
    primaryStage.show();
    pathTransition.play();
```


In graphics



Timeline animations

- The other way of creating animations is to use the `Timeline` class.
- It works similar to cartoons, you define key frames between which the node moves.
- In order for this to work, the *properties* of a node is update.
  - Properties are special values that update using *binding*.
  - We will return to these at a later lecture.
- The methods seen before for cycles and repeat are available for timelines as well.
- It is also possible to pause and re-start animations, making them good for sprites as well.

Key frames and values

- When defining a timeline, two things need to be defined:
  - Key frames – each frame defines a time for the frame.
  - Key values – the value for the frame to update.
- Any number of frames can be defined with different values.
- JavaFX calculates the change in between.
- In the example, only the x coordinate is changed, but any other property can be changed as well.

The code

```
public void start(Stage primaryStage) {
    final Image theXwing = new Image(getClass().getResourceAsStream("xwing.png"));
    final ImageView xwingShow = new ImageView(theXwing);

    Timeline time = new Timeline();
    time.setCycleCount(Animation.INDEFINITE);
    time.setAutoReverse(true);

    time.getKeyFrames().addAll(
            new KeyFrame(Duration.ZERO,
                new KeyValue(xwingShow.translateXProperty(), -500)),
            new KeyFrame(Duration.millis(2000),
                new KeyValue(xwingShow.translateXProperty(), 1000)));

    Scene scene = new Scene(new Group(xwingShow), 1000, 500);

    time.play();

    primaryStage.setTitle("X-Wing");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```


In graphics

More graphics capabilities

- One of the more prominent additions to HTML5 is the *canvas*.
  - In essence a drawing area defined in HTML.
  - Shapes and images are drawn using JavaScript.
- One of the goals for JavaFX is to be a major player when it comes to Rich Internet Applications.
- To make the transition from HTML to JavaFX easier (and more worthwhile), JavaFX 2.2 added its own canvas.
  - Very much like the HTML5 version.

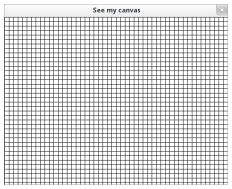
Drawing

- The usage of the JavaFX canvas is similar to that of HTML5.
- Two steps are needed:
  - First the `Canvas` is defined with size.
  - Then a `GraphicsContext` is extracted from the canvas.
- This is exactly how it is done in HTML5 as well.
- When the context is extracted, it is possible to use drawing methods on it.
  - In large they follow the same names as for HTML5.
- The example shows the parts that were possible to copy directly from an HTML5 lecture in another course.

The code

```
final Canvas theCanvas = new Canvas(500, 375);
final GraphicsContext theContext = theCanvas.getGraphicsContext2D();

for (double x = 0.5; x < 500; x += 10) {
    theContext.moveTo(x, 0);
    theContext.lineTo(x, 375);
}

for (double y = 0.5; y < 375; y += 10) {
    theContext.moveTo(0, y);
    theContext.lineTo(500, y);
}

theContext.stroke();
theContext.beginPath();
theContext.moveTo(0, 40);

// lines omitted

theContext.lineTo(60, 375);
theContext.lineTo(55, 370);

Group root = new Group();
root.getChildren().add(theCanvas);
Scene scene = new Scene(root, 500, 375);

primaryStage.setTitle("See my canvas");
primaryStage.setScene(scene);
primaryStage.show();
```


In graphics



More?

- There is plenty more to learn about animation in JavaFX.
- With this, however, you will be well on your way to master it.
- There is plenty of additional information on the Internet, though not as much as one would like.
- This lecture has only studied static images and animations based on images – further down the lecture series, we will be looking at movies as well!