6

Visual Effects, Animation, WebView, and Media

ORACLE

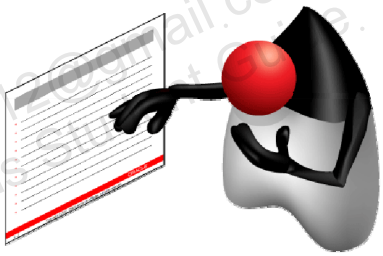
Objectives

After completing this lesson, you should be able to:

- Use animation and effects in an application
- Describe how to implement media in an application
- Describe the benefits of using WebView

Topics

- Using animation and effects in an application
- Describing how to implement media in an application
- Describing the benefits of using WebView



Animation: Timelines and Transitions

Animation in JavaFX can be divided into transitions and timelines.

`Transition` and `Timeline` are subclasses of the `javafx.animation.Animation` class.

Transitions: Transitions in JavaFX provide a way to incorporate animation in an internal timeline. Transitions can be composed to create multiple animations that are executed in parallel or sequentially.

Timelines: An animation is driven by its associated properties, such as size, location, and color. Timelines provide the capability to update the property values along the progression of time. JavaFX supports key frame animation. In key frame animation, the animated state transitions of the graphical scene are declared by start and end snapshots (*key frames*) of the scene's state at certain times. The system can automatically perform the animation. It can stop, pause, resume, reverse, or repeat movement when requested.

Using Animation in JavaFX: Introduction to Computer Animation

- In traditional hand-drawn animation, a lead animator draws key drawings in a scene.
- These key drawings are passed to assistant animators, who draw the in-between frames to achieve smooth movement.
- The action duration dictates how many in-between frames are needed. This process is called *tweening*.
- Borrowing from the traditional animation process, computer animation is a time period sliced with key frames.
- The computer does the *tweening* process by applying mathematical formulas to adjust the position, opacity, color, and other aspects required for the action. This is based on the timing constraints placed between two key frames.

ORACLE

The key drawings in a scene are the most important actions and represent the extremes of the action. The idea is to provide enough detail to present the main elements of movement. The lead animator decides how long each action should last on the screen.

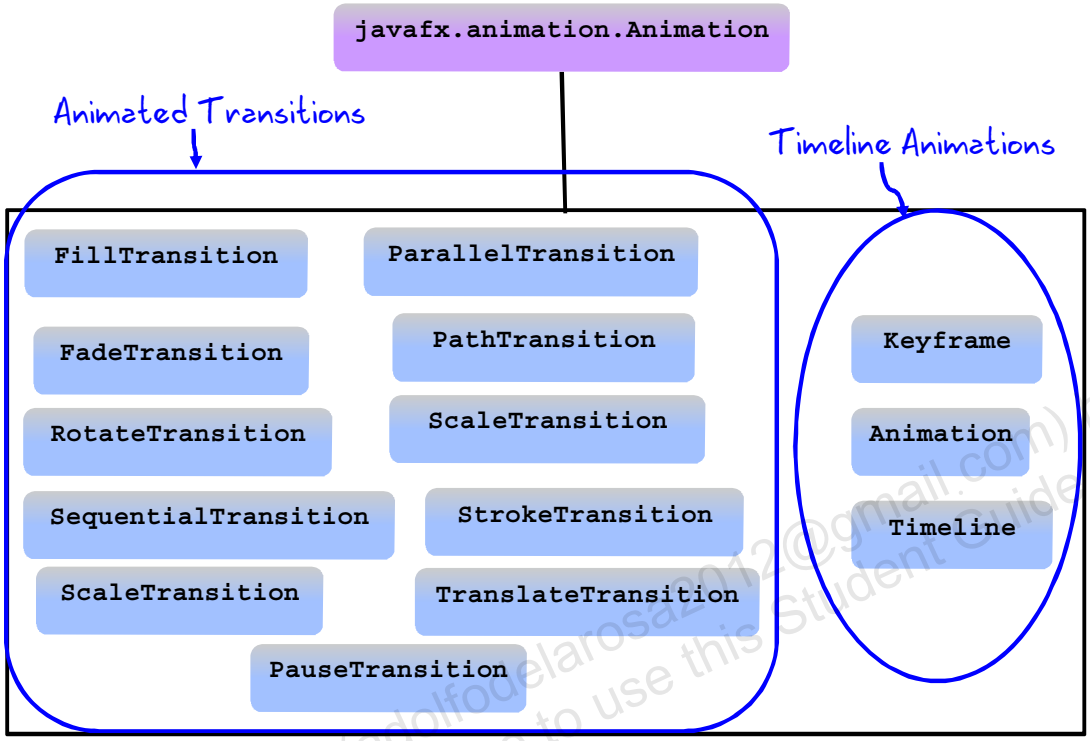
The word *tweening* is short for “in-betweening.”

In computer animation, the computer takes the role of the assistant animators. The computer decides how many in-between frames are required based on the timing constraints placed between two key frames.

The primary property of animation is time, so JavaFX supports the time period of an animation sequence with a timeline. The JavaFX class that represents animation actions spread across a time duration is `javafx.animation.Timeline`. Key frames, derived from `javafx.animation.KeyFrame`, are interspersed across the timeline at specified intervals, represented by time durations (`javafx.lang.Duration`).

These key frames might contain key values (`javafx.animation.KeyValue`) that represent the end state of the specified application values such as `position`, `opacity`, and `color`. These key values might also include actions that execute when the key time occurs. Key values containing a declaration of the mathematical formula, or interpolator (`javafx.animation.Interpolator`), should be used in the tweening process.

`Animation` Is Applied to a Node




```
public abstract class Animation
extends java.lang.Object
```

The Animation class provides the core functionality of all animations used in the JavaFX Runtime. An Animation can run in a loop by setting `cycleCount`. To make an animation run back and forth while looping, set the `autoReverse` flag.

Call `play()` or `playFromStart()` to play an animation. The animation progresses in the direction and speed specified by `rate`, and stops when its duration has elapsed. An animation with indefinite duration (a `cycleCount` of `INDEFINITE`) runs repeatedly until the `stop()` method is explicitly called, which will stop the running animation and reset its play head to the initial position.

An animation can be paused by calling `pause()`, and the next `play()` call will resume the animation from where it was paused.

An animation's play head can be randomly positioned whether it is running or not. If the Animation is running, the play head jumps to the specified position immediately and continues playing from the new position. If the animation is not running, the next `play()` will start the animation from the specified position.

Inverting the value of `rate` toggles the play direction.

Timeline Animation

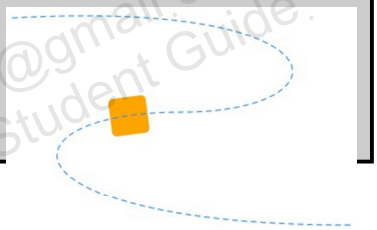
```
final Rectangle rectBasicTimeline = new Rectangle(100,
   50, 100, 50);
rectBasicTimeline.setFill(Color.BROWN);
...
final Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
final KeyValue kv = new
   KeyValue(rectBasicTimeline.xProperty(), 300);
final KeyFrame kf = new KeyFrame(Duration.millis(500),
   kv);
timeline.getKeyFrames().add(kf);
timeline.play();
```

Using the Transition Classes

Class	Description
TranslateTransition	Translates (moves) a node from one location to another
RotateTransition	Rotates a node
ScaleTransition	Scales (increases or decreases the size of) a node
FadeTransition	Fades (increases or decreases the opacity of) a node
PathTransition	Moves a node along a geometric path
SequentialTransition	Allows you to define a sequential series of transitions
PauseTransition	Used in a SequentialTransition to wait for a period of time
ParallelTransition	Allows you to define a parallel series of transitions
StrokeTransition	Changes the stroke of a color over a period of time
FillTransition	Allows a shape to fill over a period of time

Animated Transition: Path Transition

```
final Rectangle rectPath = new Rectangle (0, 0, 40, 40);
rectPath.setArcHeight(10);
rectPath.setArcWidth(10);
rectPath.setFill(Color.ORANGE);
...
Path path = new Path();
path.getElements().add(new MoveTo(20,20));
path.getElements().add(new CubicCurveTo(380, 0, 380, 120, 200, 120));
path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 380, 240));
PathTransition pathTransition = new PathTransition();
pathTransition.setDuration(Duration.millis(4000));
pathTransition.setPath(path);
pathTransition.setNode(rectPath);
pathTransition.setOrientation(PathTransition.OrientationType.ORTHOGONAL_
    TO_TANGENT);
pathTransition.setCycleCount(Timeline.INDEFINITE);
pathTransition.setAutoReverse(true);
pathTransition.play();
```



A path transition moves a node along a path from one end to the other over a given period of time.

In the example in the slide, a path transition is applied to a rectangle. The animation is reversed when the rectangle reaches the end of the path. In the code, a rectangle with rounded corners is created, and then a new path animation is created and applied to the rectangle. Setting the orientation `ORTHOGONAL_TO_TANGENT` keeps the node perpendicular to the path's tangent along the geometric path.

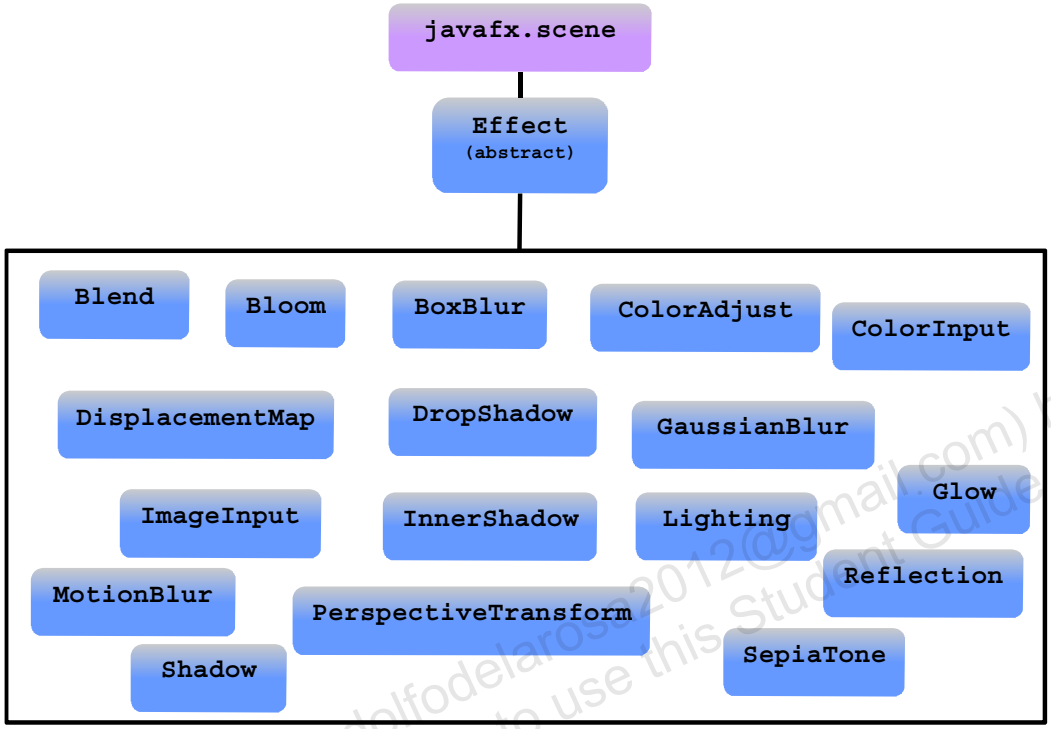
Effects

The Effect class `javafx.scene.effect` creates filter effects that operate on an image and on any node they are applied to and use algorithms that apply to pixels. Effects leverage SSE, OpenGL, D3D, or plain Java. You can apply effects to any node, and effects can be combined using the Blend effect.

Effects are graphical algorithms that produce an image, usually as a modification to a source image.

Effects can be associated with a node by setting the node's effect attribute.

`Effect` Is Applied to a Node



The `javafx.scene.effect` package provides the set of classes for attaching graphical filter effects to JavaFX scene graph nodes.

```
public abstract class Effect
extends java.lang.Object
```

`Effect` is the abstract base class for all effect implementations. An effect is a graphical algorithm that produces an image, typically as a modification of a source image. An effect can be associated with a scene graph node by setting the `Node.effect` attribute. Some effects change the color properties of the source pixels (such as `ColorAdjust`), others combine multiple images together (such as `Blend`), and still others warp or move the pixels of the source image around (such as `DisplacementMap` or `PerspectiveTransform`).

Each effect has at least one input defined. The input can be set to another effect to chain the effects together and combine their results, or it can be left unspecified—in which case, the effect will operate on a graphical rendering of the node it is attached to.

Note: This is a conditional feature. The `ConditionalFeature.EFFECT` indicates that filter effects are available on the platform. If an application uses an effect on a platform that does not support it, the effect will be ignored.

Effects can be combined using the `Blend` effect.

Effect: Example

```
static Node innerShadow() {
        InnerShadow is = new InnerShadow();
        is.setOffsetX(2.0f);
        is.setOffsetY(2.0f);

        Text t = new Text();
        t.setEffect(is);
        t.setX(20);
        t.setY(100);
        t.setText("Inner Shadow");
        t.setFill(Color.RED);
        t.setFont(Font.font("null", FontWeight.BOLD, 80));

        t.setTranslateX(300);
        t.setTranslateY(300);

        return t;
    }
```

An inner shadow is an effect that renders a shadow inside the edges of the given content with the specified color, radius, and offset.

Quiz

Which two classes are types of transitions?

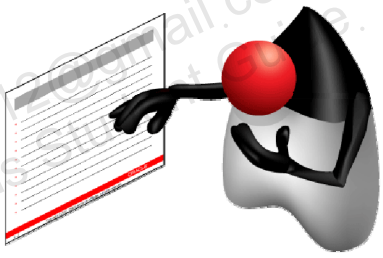
a. PerspectiveTransform, PathTransition
b. PathTransform, RotateTransform
c. SequentialTransition, PauseTransition
d. PathTransition, BlendTransition

Answer: c

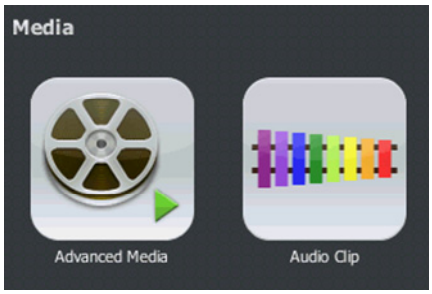
a. PerspectiveTransform is an effect.
b. PathTransform and RotateTransform are not transitions.
c. SequentialTransition and PauseTransition are transitions.
d. BlendTransition is not a transition.

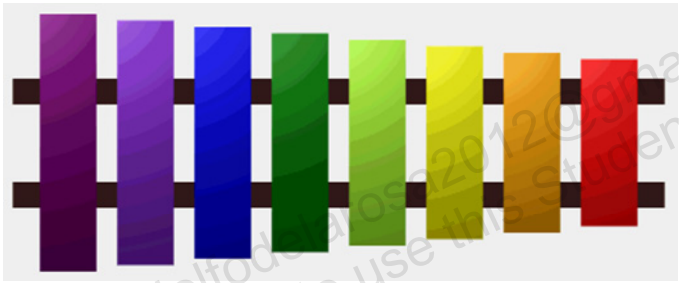
Topics

- Using animation and effects in an application
- Describing how to implement media in an application
- Describing the benefits of using WebView



Media





The `javafx.scene.media` package enables developers to create media applications that provide media playback in the desktop window or in a web page on supported platforms.

The media formats that are currently supported are the following:

- **Audio:** MP3, AIFF containing uncompressed PCM, WAV containing uncompressed PCM
- **Video:** FLV containing VP6 video and MP3 audio

Some of the features supported by the media stack include the following:

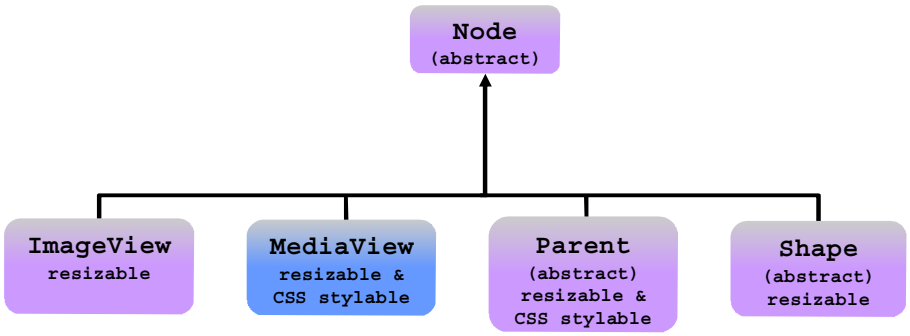
- VP6 + MP3 using a FLV container
- MP3 for audio
- HTTP, FILE protocol support
- Progressive download
- Seeking
- Buffer progress
- Playback functions (Play, Pause, Stop, Volume, Mute, Balance, Equalizer)

Building a Media Player

- `Media`: A media resource containing information about the media, such as its source, resolution, and metadata
- `MediaPlayer`: The key component providing the controls for playing media
- `MediaView`: A Node object that supports animation, translucency, and effects

Each element of the media functionality is available through the JavaFX API. The media classes are interdependent and are used in combination to create an embedded media player.

MediaView Is a Node




```
public class MediaView
extends Node
```

`MediaView` is a node that provides a view of media being played by a MediaPlayer.

MediaView

```
public MediaView(MediaPlayer mediaPlayer)
```

Creates a `MediaView` instance associated with the specified `MediaPlayer`. Equivalent to

```
MediaPlayer player; // initialization omitted
MediaView view = new MediaView();
view.setPlayer(player);
```

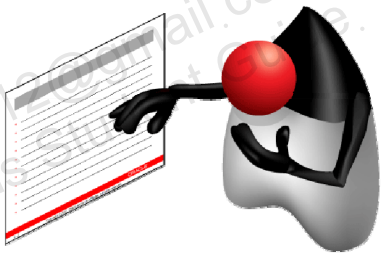
Parameters:

`mediaPlayer` - the `MediaPlayer` the playback of which is to be viewed via this class.

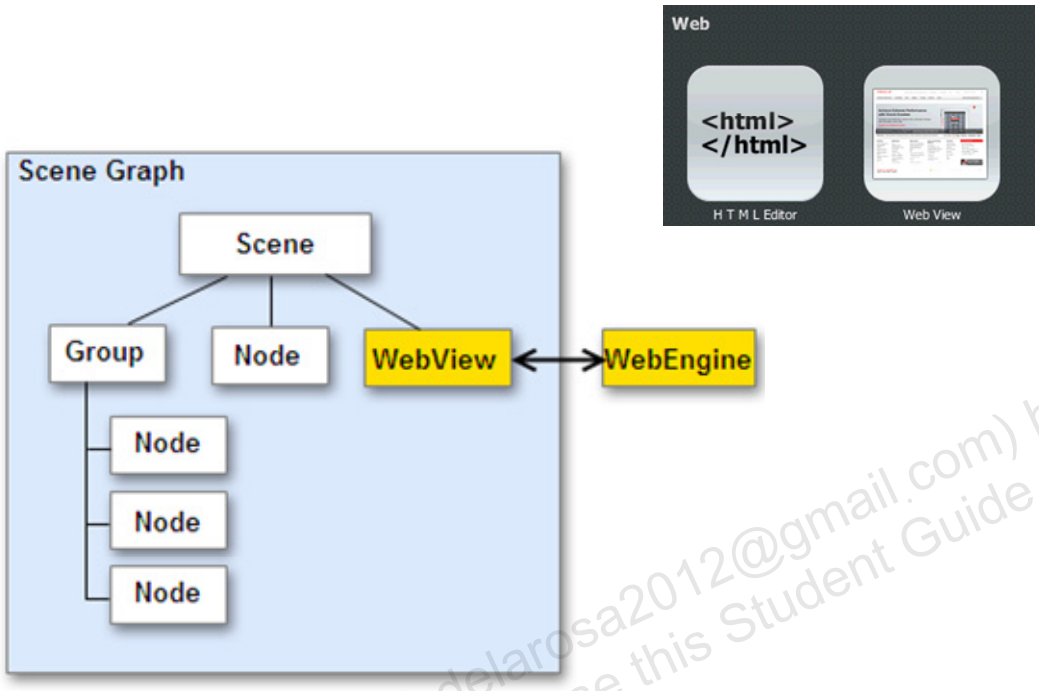
ORACLE

Topics

- Using animation and effects in an application
- Describing how to implement multimedia in an application
- Describing the benefits of using WebView



WebView

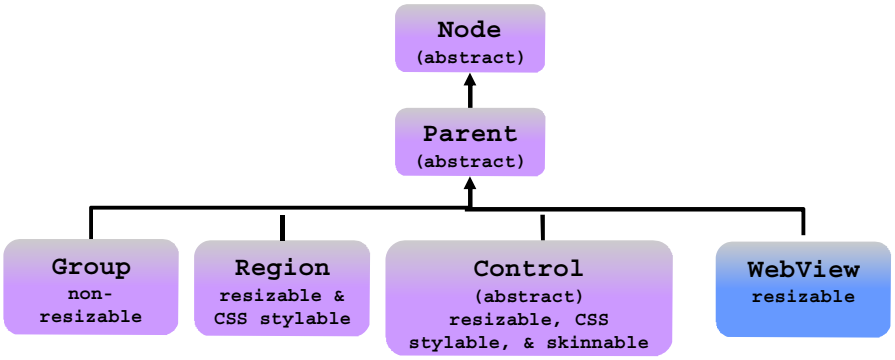


The JavaFX SDK introduces the embedded browser, which is a user interface component that provides a web viewer and full browsing functionality through its API. The embedded browser component is based on WebKit, an open source web browser engine. It supports Cascading Style Sheets (CSS), JavaScript, Document Object Model (DOM), and the following features of HTML5: rendering canvas and timed media playback.

The embedded browser enables you to perform the following tasks in your JavaFX applications:

- Render HTML content from local and remote URLs.
- Execute JavaScript commands.
- Access the document model from Java code.
- Handle events.
- Manage web pop-up windows.
- Apply effects to the embedded browser.

WebView Is a Node




```
public final class WebView
extends Node
```

`WebView` is a node that manages a WebEngine and displays its content. The associated WebEngine is created automatically at construction time and cannot be changed afterward.

`WebView` handles mouse and some keyboard events, and manages scrolling automatically, so there is no need to put it into a ScrollPane.

`WebView` objects must be created and accessed solely from the FX thread.

WebView: Example

```
WebView webView = new WebView();
WebEngine webEngine = webView.getEngine();
webEngine.load("http://javafx.com");
// add WebView to the scene
```

Use Cases for WebView

- Displaying oAuth login dialogs (Facebook, Flickr, and so on)
- Embedding existing web forms or UI
- Existing GWT pages for form submission
- Embedding maps
- Rendering FX content over nodes on a web page
- Displaying and editing rich text
- Complicated text layout

Quiz

An embedded browser can access only a remote URL.

a. True
b. False

Answer: b

a. Incorrect answer. An embedded browser can access a remote and local URL.
b. Correct answer. It can access a remote and local URL.

Summary

In this lesson, you should have learned how to:

- Use animation and effects in an application
- Describe how to implement media in an application
- Describe the benefits of using WebView

Practice 6: Overview

6-1: Adding a Fade Transition