



# JavaOne™

[java.sun.com/javaone](http://java.sun.com/javaone)

## Inside The JavaFX™ Script Technology- Based Runtime APIs: Scene Graph & JWebPane Component

Artem Ananiev  
Igor Nekrestyanov  
Jim Graham

TS-6610



- ▶ A close look at two of the libraries JavaFX™ Script software is based on: the Scene Graph library, and the JWebPane HTML component.



GOAL

# Agenda

- Project Scene Graph
- Web Component
- Summary and Conclusions

# *The Scene Graph Project is a Java™* library for Scene Graphs

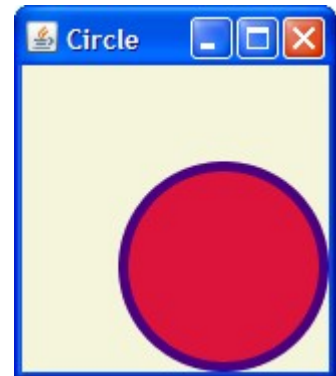
- A scene graph is a retained view of a graphical scene
- Swing is a scene graph, but...
  - By convention elements are fairly high level (GUI components)
  - Component positioning is limited to integer position and size
  - And there's no intrinsic support for transforms or animation
- ... and Java 2D™ API is not
  - It's an “immediate mode” API
- Scene graphs simplify creating graphical applications
  - Declare where you want things, and when
  - Let the system figure out how to draw them

# JavaFX Script Software and Scene Graphs

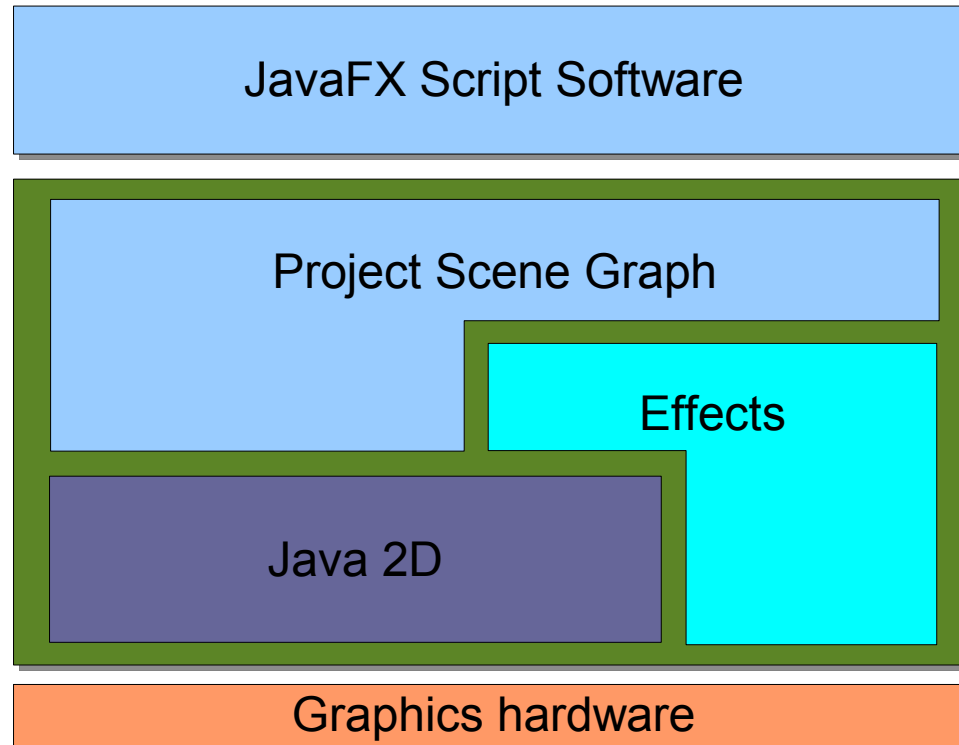
- Object literal syntax simplifies defining scenes

```
var scene = Circle {  
    centerX: 100  
    centerY: 100  
    radius: 50  
    fill: Color.CRIMSON  
    stroke: Color.INDIGO  
    strokeWidth: 5  
};
```

```
Frame {  
    title: "Circle"  
    content: Canvas { content:scene }  
    background: Color.BEIGE  
    visible: true  
}
```



# JavaFX Script Software Implementation Depends on Project Scene Graph



# Scene Graph Elements

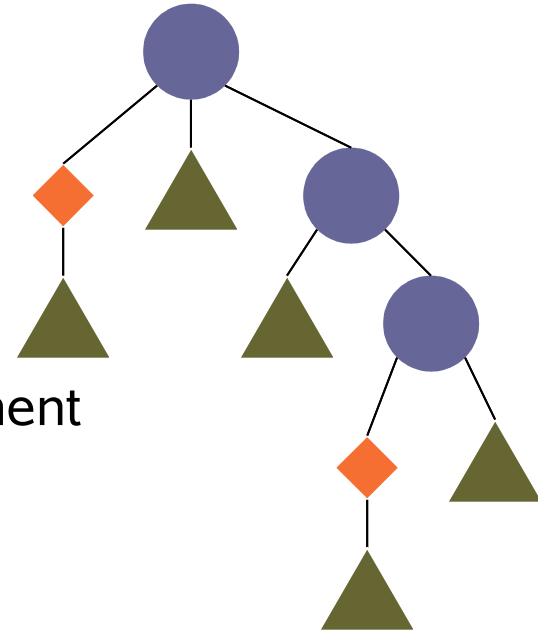
- Nodes
  - Graphics, text, components, images
- State
  - Transforms, effects, visibility, ...
- Events/Listeners
  - Mouse, keyboard, node updates, ...
- Animation
  - Varying properties over time

# Scene Graph Approaches

- Atomic behaviors
  - Every node has a single specific role
  - Nodes can be content
  - Nodes can be modifiers
  - Easier to implement, (more customizable)
  - (SGNode and friends)
- Embellished content
  - Nodes are content
  - Modifiers are attributes of nodes
  - Easier for developers, especially casual developers
  - (FXNode and friends)
- Project Scene Graph provides both
  - FXNode for ease of use, SGNode for customization (and implementation ease)

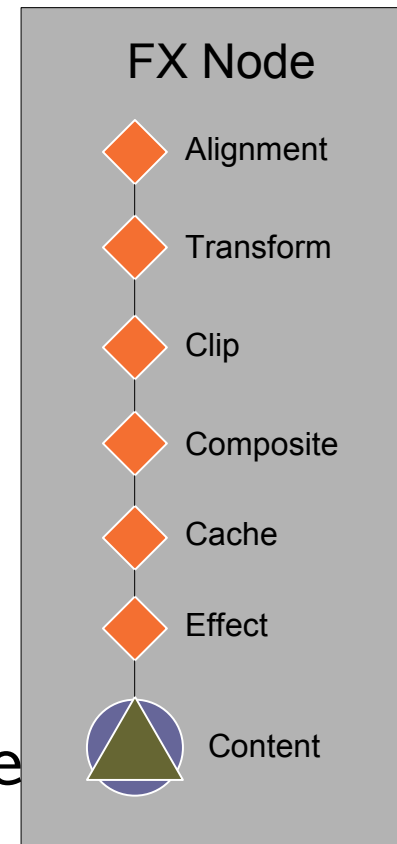
# Scene Graph Nodes - SGNODE

- SGNODE base class
  - Attributes and Content are both nodes
- Leaves and Parents
  - SGLeaf, SGParent
- SGLeaf subclasses
  - SGImage, SGShape, SGText, SGComponent
- SGParent subclasses
  - SGGroup: parent of multiple children
  - SGFilter: state filtering of single child
    - Attributes as SGFilter subclasses
    - SGTransform, SGComposite, SGClip, etc.
- `com.sun.scenario.sceneagraph` package

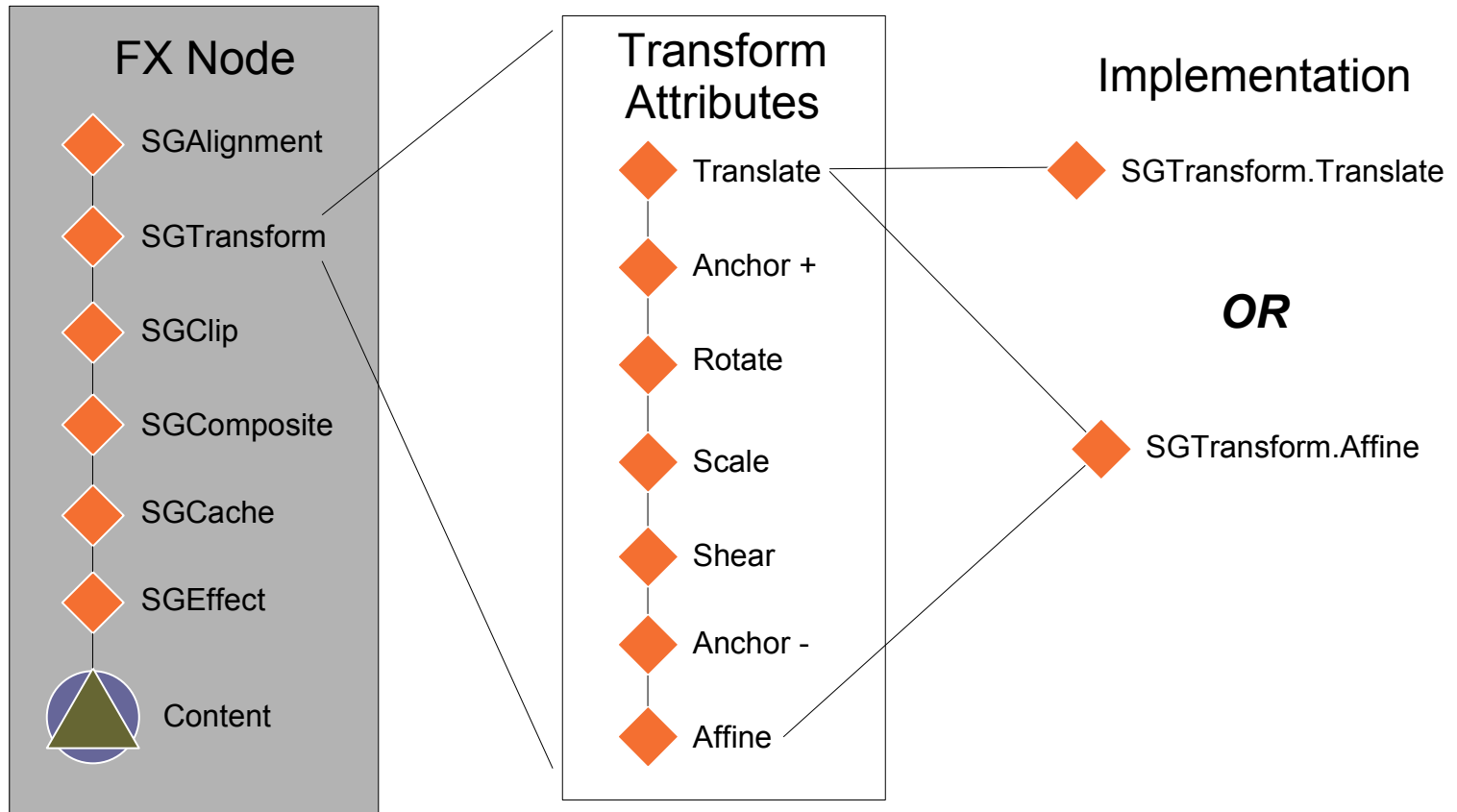


# Scene Graph Nodes - FXNode

- FXNode base class
  - Provides common attributes
  - Internally maintains an SG subtree
- FX content subclasses
  - FXImage: Images
  - FXShape: Geometry display
  - FXText: Stroked/filled text
  - FXComponent: Swing component
- Grouping subclass
  - FXGroup: parent of multiple children
- `com.sun.scenario.sceneagraph.fx` package



# FXNode Attributes map to SGFilter nodes



# Nesting – Component Nodes

- Component nodes can embed a Swing component
  - `FXComponent.setComponent(comp)`
  - `SGComponent.setComponent(comp)`
- Embedded Swing components work as is
  - Receive events and draw as if part of a Swing application
    - mouse, keyboard, focus, repaints, etc. all work as usual
  - But they respond to the Scene Graph attributes/properties
    - Transforms, Opacity, Clips, Effects, etc.

# Events

- Very similar to AWT events
  - Add Listeners to any SGNODE
  - Callback methods have AWT event + SGNODE
- Mouse events – SGMouseListener
  - eg. mouseClicked(MouseEvent e, SGNODE n)
- Keyboard events – SGKeyListener
  - eg. keyPressed(KeyEvent e, SGNODE n)
- Focus events – SGFocusListener
  - eg. focusGained(FocusEvent e, SGNODE n)
- Node events – SGNODEListener
  - boundsChanged(SGNODEEvent e)

# Animation

- Vary object/node properties over time
  - This property goes from this value to that value over N seconds
- With interesting behaviors
  - Non-linear interpolation (acceleration/no acceleration)
  - Repeating/reversing animations (A->B, A->B or A->B->A)
  - Chained animations (run A, then start B, etc.)

# Animation classes

- `com.sun.scenario.animation` package
  - Usable with either `FXNodes` or `SGNodes`
- `Clip` – a single animation event
  - Specify a property, duration, repeat count, interpolation type
  - Can trigger other Clips on begin/end
  - Specify `TimingTargets` to notify, or `BeanProperties` to modify
  - All attributes on `SGNode` and `FXNode` classes are Bean-friendly
- `Timeline` – several scheduled animations
  - A list of `Clip` objects with relative starting times

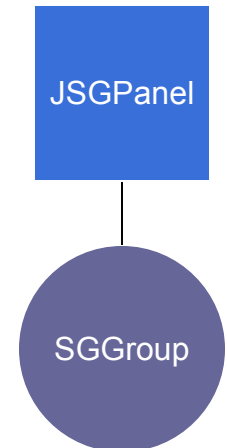
# Scene Graph Effects Framework

- aka “Eye Candy 2008”
- `com.sun.scenario.effect` package
  - Designed to stand on its own as a pixel effects package
  - Transparent interoperability with `SCEffect` or `FXNode.setEffect`
- Flexible implementations
  - Run everywhere with fast Java technology-based and native backends
  - Run faster on D3D and OpenGL hardware (where supported)
- Easily apply many advanced effects to nodes or trees:
  - DropShadow, InnerShadow, Reflection, Perspective
  - Gaussian/Motion Blur, Glow, Bloom, Phong Lighting
  - Blend, ColorAdjust, SepiaTone
  - Chains of any of the above effects
  - More effects on the way

# Using Project Scene Graph with Java Platform

- Create a JSJPanel component
- Set its scene property to your root SGNode
- Add the JSJPanel to a JFrame, and voila

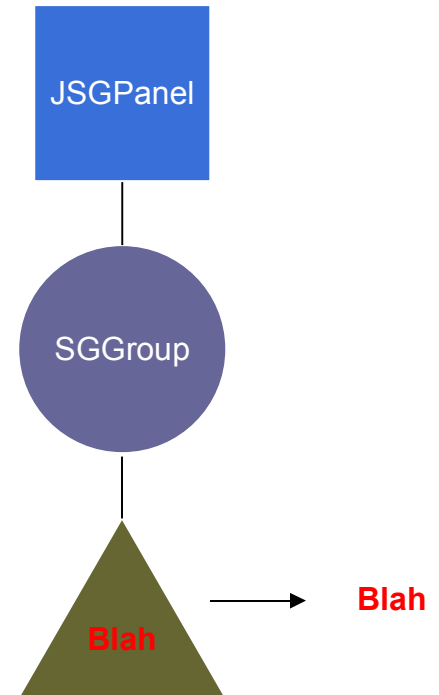
```
JFrame f = new JFrame("Scene");  
SGGroup rootNode = new SGGroup(); ●  
// add the rest of your scene to rootNode  
JSJPanel p = new JSJPanel();  
p.setScene(rootNode);  
f.add(panel);  
f.pack();  
f.setVisible(true);
```



# Adding a Text Node

- Create the SGText node
- Set its content and state
- Add it into the scene

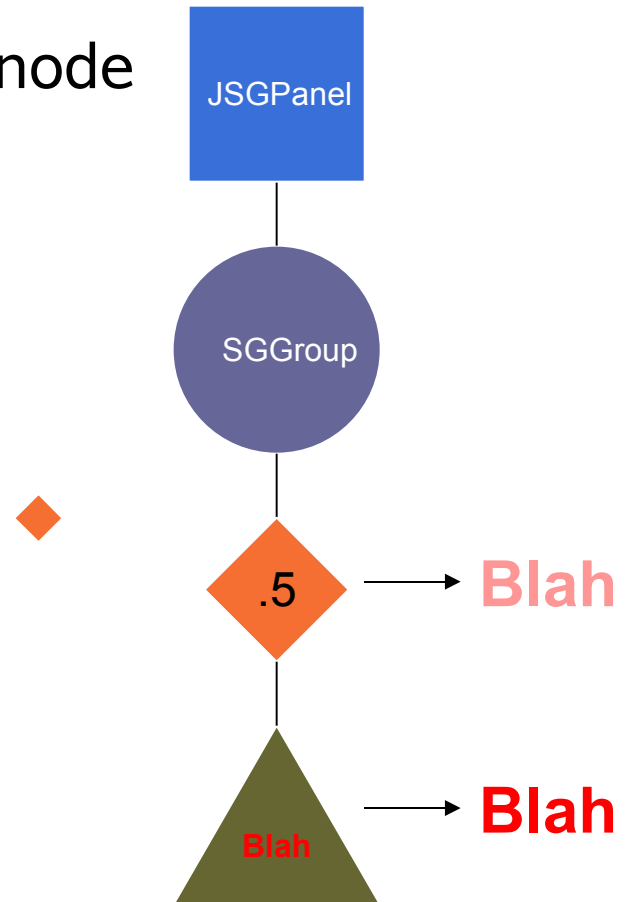
```
SGText textNode = new SGText(); ▲  
textNode.setText("Blah");  
textNode.setMode(SGText.FILL);  
textNode.setFillPaint(Color.RED);  
rootNode.add(textNode);
```



# Adding an Alpha Composite Filter

- Create a SGComposite (an SGFilter) node
- Set its state
- Parent the child that it affects
- Add it to the scene

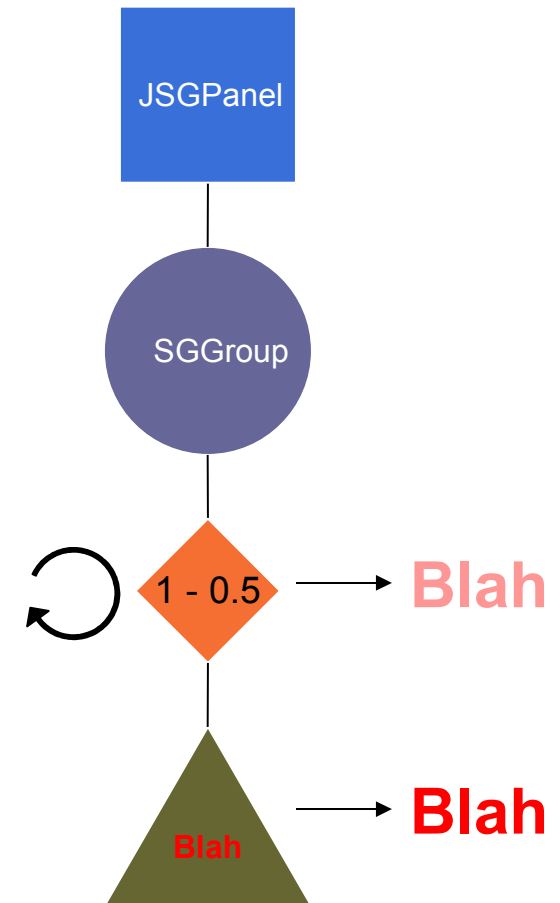
```
// Make textNode translucent
SGComposite c = new SGComposite();
c.setOpacity(.5f);
c.setChild(textNode);
rootNode.add(c);
```



# Animating a Node's Property

- Create an animation Clip
  - Set the duration, object/property, and values to be animated
- Start the Clip at the desired time

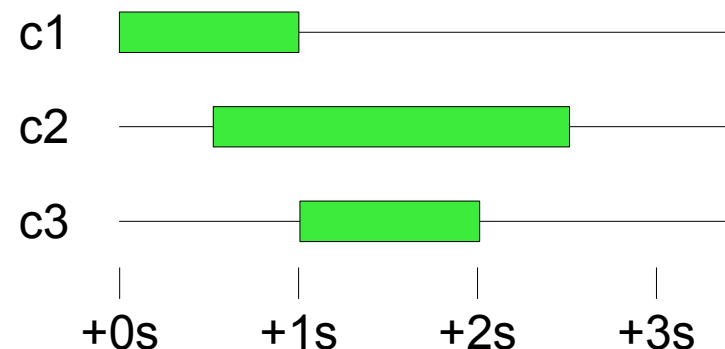
```
// Fade comp filter over 1 sec
Clip fader = Clip.create(
    1000,          // duration in ms
    c,            // animating obj
    "opacity",    // animating prop
    1.0f,         // "from" value
    0.5f);        // "to" value
fader.start();
```



# Animating a sequence of Clips - Timeline

- Create several animation Clips
- Schedule the Clips in a Timeline
- Start the Timeline at the desired time

```
Clip c1 = Clip.create(1000, ...);  
Clip c2 = Clip.create(2000, ...);  
Clip c3 = Clip.create(1000, ...);  
Timeline t = new Timeline();  
t.schedule(c1, 0);  
t.schedule(c2, 500);  
t.schedule(c3, 1000);  
t.start();
```



# Animating values - KeyFrames

- Create several KeyFrame objects
  - Include fraction(0..1), value
- Collect KeyFrame objects into KeyFrames
  - Include object, property, list of KeyFrame objects
- Create a clip to drive the KeyFrames
- Start the clip at the desired time

```
kf1 = KeyFrame.create(0.0, 10.0f);  
kf2 = KeyFrame.create(0.2, 40.0f);  
kf3 = KeyFrame.create(1.0, 50.0f);  
keyframes = KeyFrames.create(  
    obj, "foo", kf1, kf2, kf3);  
c = Clip.create(10000, keyframes);  
c.start();
```

# Adding Effects

- Create an Effect instance or chain
- Insert into SGNODE tree using SGEffect
- Add to an FXNode using FXNode.setEffect

```
DropShadow shadow = new DropShadow();  
shadow.setRadius(5f);  
shadow.setOffsetX(3);  
shadow.setOffsetY(3);  
shadow.setColor(Color.DARK_GRAY);
```

```
FXText node = new FXText();  
node.setText("Hello World");  
node.setEffect(shadow);
```



# Rendering the Scene

- Nothing to do!
- A visible scene knows how to draw itself

# Scene Graph Demo

DEMO

# Summary

- FXNodes are fun and easy to use!
- SGNodes give full control for the tinkerers
- Both designed to give you optimal results
- Open Source project approach...

# JWebPane HTML Component

# Web Content to Enrich Applications

- Why embed a “street” HTML viewer in you application?
  - Show web advertisements (and make some money!)
  - Display customer locations using Web map service
  - Mix-in web clients for IM, shopping, media playback, etc
  - Create new user interfaces for Web browsing
  - View web content from you new email/IM/calendar/etc clients
- Street HTML is in the AIR that we breathe

# Requirements

- Support modern Web pages
  - JavaScript™, CSS, ...
  - Street HTML
- Should look and behave as part of Java application environment/JavaFX application
  - e.g. under transform or after LAF change
- Interaction support
  - Notifications on important web page events
  - Browser emulation for web pages (status bar, etc.)
  - Control of page behavior (opening new windows, etc.)
  - Execute scripts in the context of the web page from Java code
  - Access to web page content from Java code
- Cross-platform support

# Existing HTML Support in Java Applications

- `javax.swing.text.html`
  - Good for rich text formatting
  - No support for advanced features (e.g. JavaScript technology) and street HTML
- `java.awt.Desktop`
  - Opens Web page in the external browser window
    - Not a part of application
  - No support for interaction
- JDIC and JDICPlus
  - Embedded browser such as IE or Mozilla
  - Heavyweight component
    - Hard to embed into Swing and JavaFX applications
  - Experimental code from SwingLabs
- Number of 3<sup>rd</sup> party libraries

# New HTML Component

## ➤ Based on WebKit

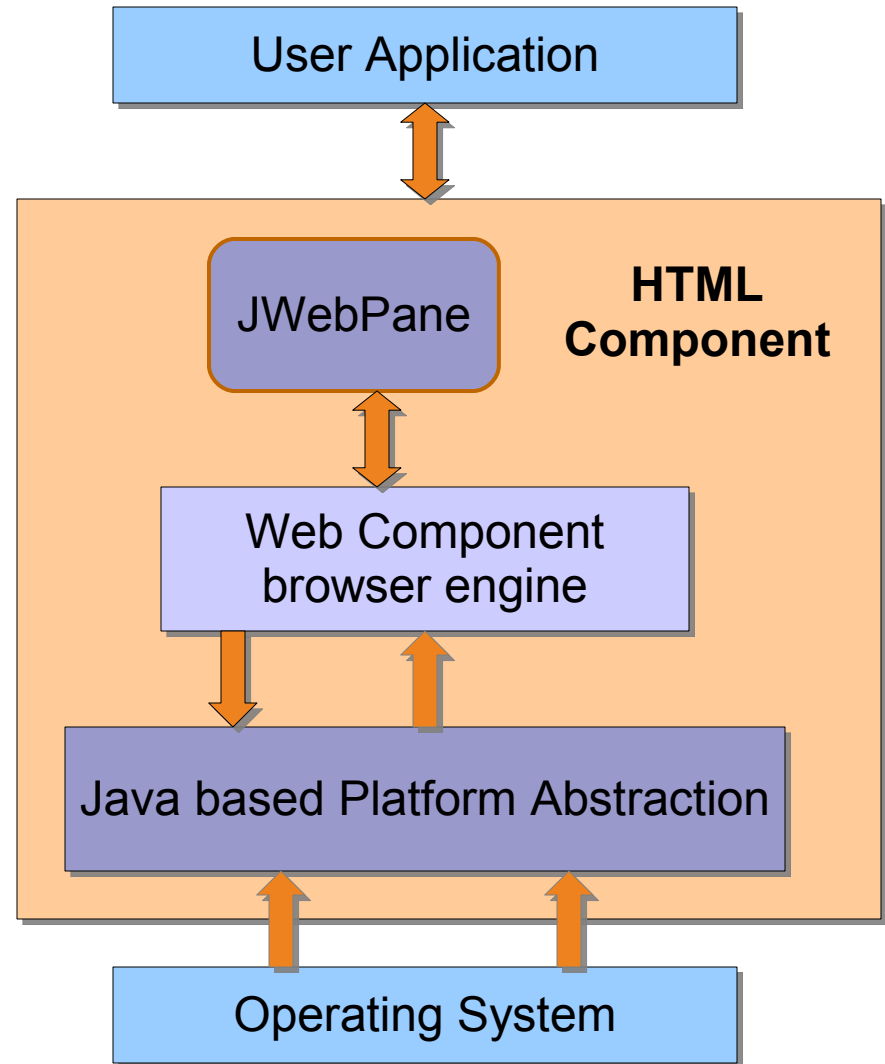
- Native open source web browser engine
- Available on many platforms
- Supports HTML4, CSS, JavaScript programming language, DOM
- Supports street HTML

## ➤ Java technology implementation of WebKit Platform Abstraction

- Metrics calculation
- Painting
- Networking
- Event dispatching

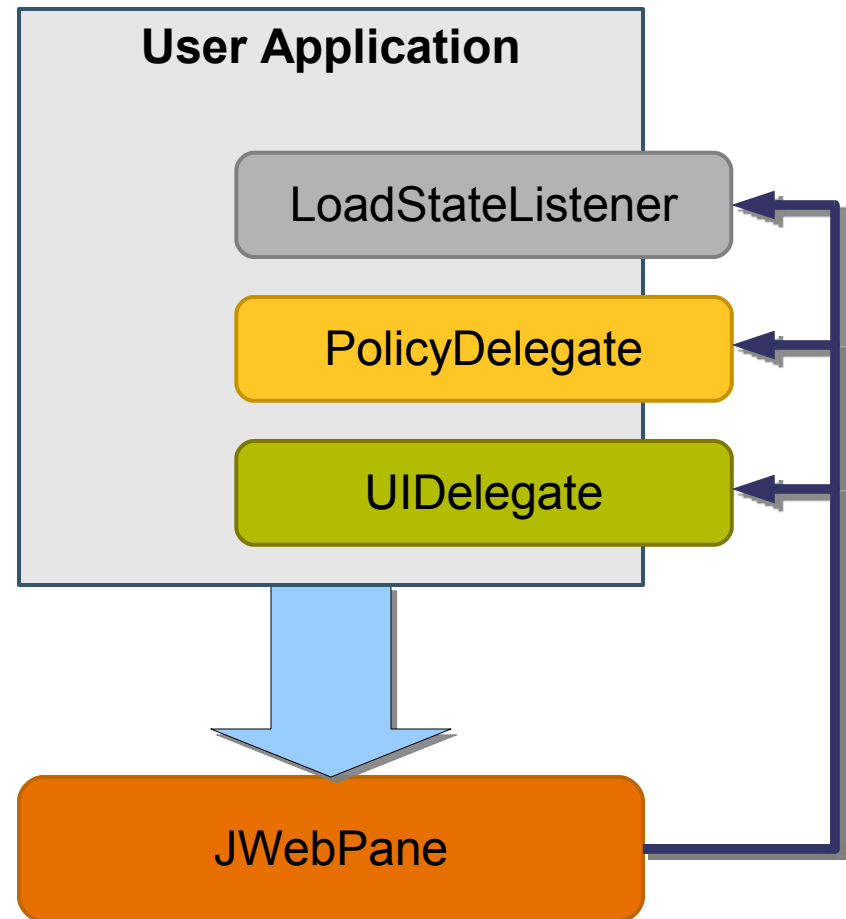
## ➤ Benefits

- True lightweight component



# API at a Glance

- JWebPane
  - Abstraction of web page browser
- LoadStateListener
  - Notifications on loading progress
- PolicyDelegate
  - Decisions on sensitive actions
- UIDelegate
  - Customized “browser” appearance
- API is evolving



# JWebPane

- Web page browser component
  - Load/Print web page
  - Zoom page or scale text
  - Copy/Search page content
  - Navigate through history
- An extension of JComponent
  - Lightweight
- Example: display web page

```

JWebPane webpane = new JWebPane ();
jframe.add(webpane);
...
webpane.load(new URL("http://www.sun.com"));

```

# LoadStateListener

- Notifications on web page loading progress
  - Including partial load status and loading of resources
- Example: create a thumbnail of web page

```

class ThumbnailMaker extends LoadStateAdapter {
    BufferedImage bi = ...
    public void loadingFinished(LoadStateEvent e) {
        Graphics2D g = (Graphics2D) bi.getGraphics();
        JWebPane wp = e.getFrame().getWebPane();
        g.setTransform(getScale(wp.getSize()));
        e.getFrame().getWebPane().paint(g);
        processThumbnail(bi);
    }
}

webpane.addLoadStateListener(new ThumbnailCreator());

```

# UIDelegate

- User defined logic for standard JavaScript programming language calls to browser
  - Show/Hide menu or toolbar
  - Set status bar text

- Example: custom status bar

```

class MyUI implements UIDelegate {
    JLabel  statusLabel;
    public void setStatusbarText(
        JWebPane pane, String message) {
        statusLabel.setText(message);
    }
    ....
}
...
JWebPane p = new JWebPane(null, new MyUI());
  
```

# PolicyDelegate

- Delegation of behavioral decisions
  - Accept/reject: page navigation, opening windows, script execution, redirection, etc.
- Example: popup blocker

```

class MyPolicy implements PolicyDelegate {
    public boolean permitAction(PolicyRequest r) {
        if (PolicyRequest.RequestType.NEW_WINDOW
            == r.getType()) {
            return isTrustedSite(r.getURL());
        }
        return true;
    }
}

JWebPane p = new JWebPane(new MyPolicy(), null);
  
```

# Advanced features

- Executing JavaScript code in context of web page
- Persistent cookie support
- Access to DOM tree
- Plugins
- Resource management
  - Cache management
  - History depth
  - Sharing resources between Web Component and Java/JavaFX code
- Example: simple JavaScript code

```
webpane.executeScript(  
    "window.status='Hello from javascript!';"  
);
```

# JWebPane Component Demo

A large, light blue arrow pointing to the right, positioned on the left side of the slide.

DEMO

# Summary

- JWebPane is a Web Browser component
  - Supports Web 2.0 content and street HTML
  - Looks and behaves like a regular Swing component
  - Execution of JavaScript code snippets from Java code
  - Access to DOM tree
- Support for plugins and some other advanced features planned for future release

# For More Information

- Check out the open source project
  - <http://scenegraph.dev.java.net>
- And the JavaFX software project which uses it
  - <http://openjfx.dev.java.net>
- And the demos
  - <http://scenegraph-demos.dev.java.net/>

# THANK YOU



Artem Ananiev  
Igor Nekrestyanov  
Jim Graham

TS-6610

