
# JavaFX

SUN TECH DAYS 2008–2009
A Worldwide Developer Conference

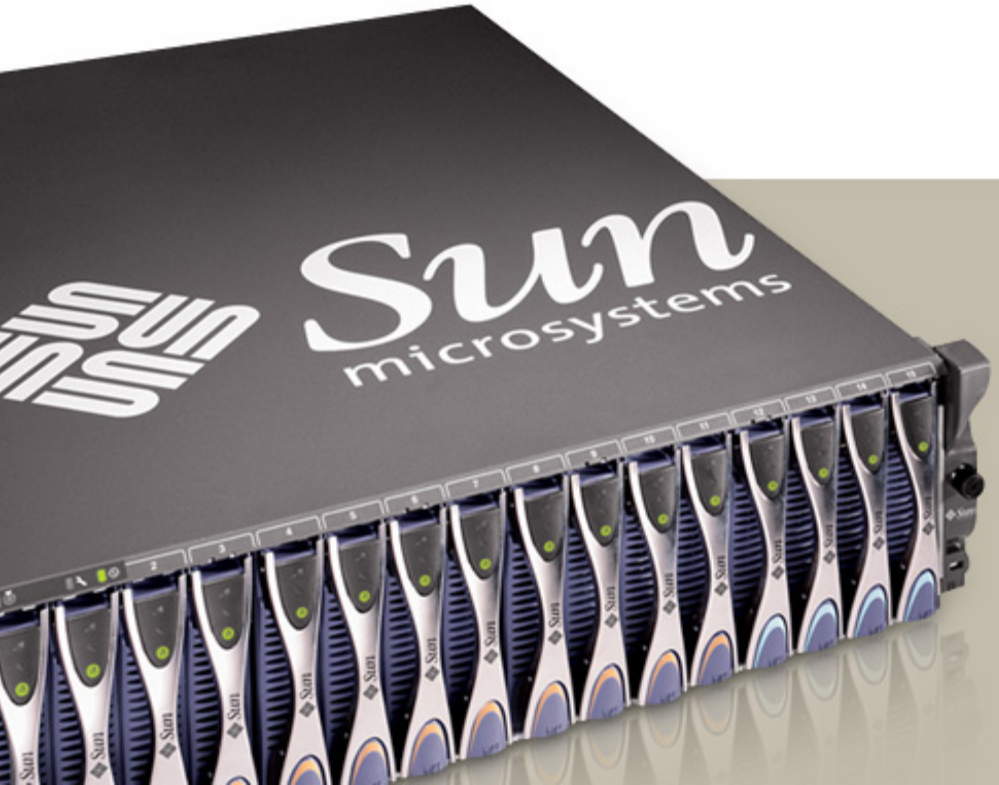
Simon Ritter
Technology Evangelist

# JavaFX Vision

JavaFX is the platform for creating and delivering
Rich Internet Applications
across all the screens of your life


***JavaFX is Powered by Java***

# Overview



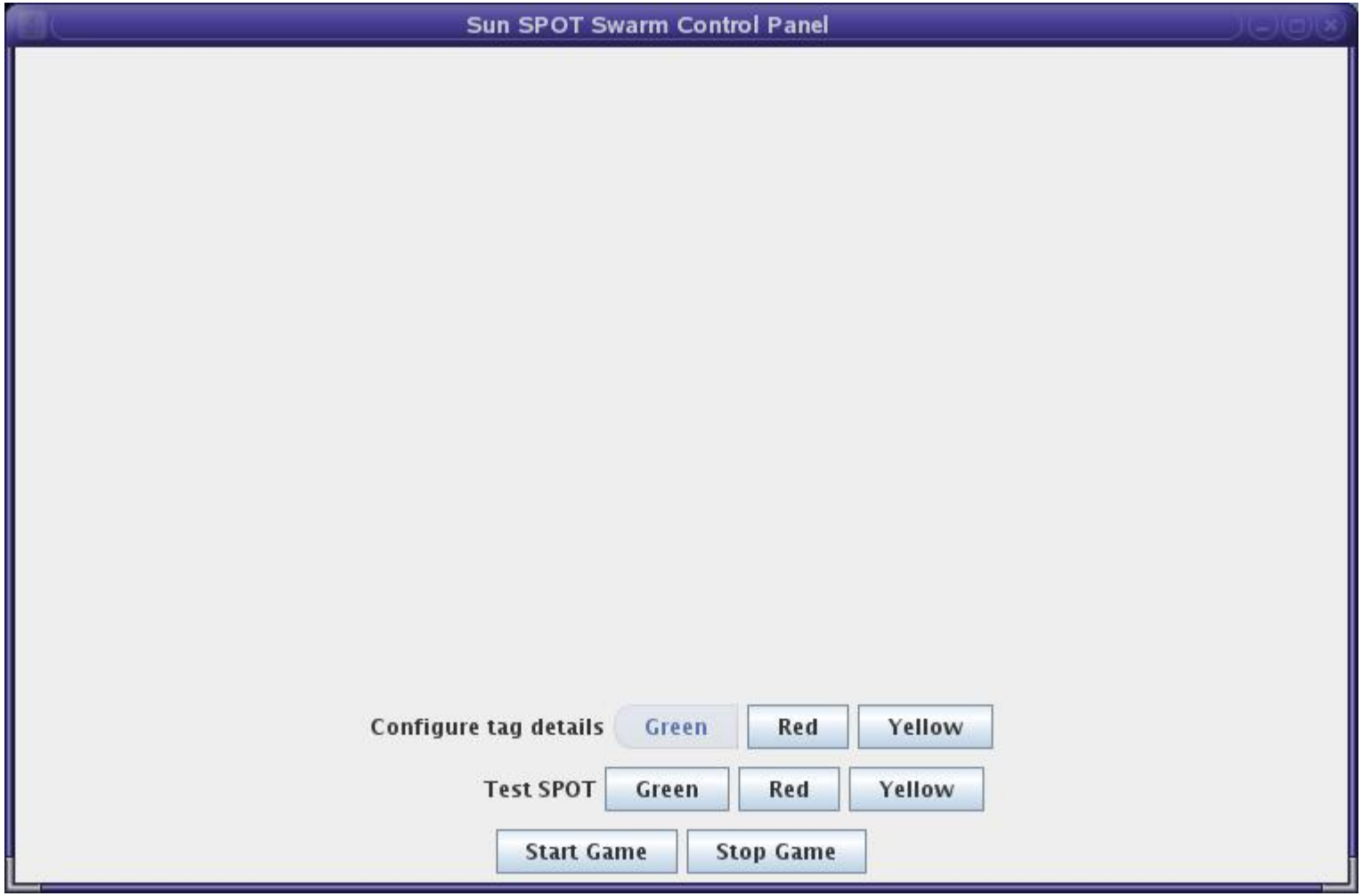
# JavaFX: Design Questions

- Why does it take a long time to write Java GUI programs?
- How can we avoid the “Ugly Java GUI” stereotype?
- Why do Flash programs look different to Java programs?
- Why does it seem easier to write web-apps than Swing programs?
- And how can I avoid having an enormous, writhing mass of listener patterns?

# Java GUIs: Why are they not so rich?

- AWT/Swing Container/Component Hierarchy
  - A tree of rectangular (mostly grey) boxes
  - If all you do is compose Swing components together, the result is typically “the Ugly Java GUI”
  - Same problem exists with other toolkits, e.g., GTK, VB
- UI Designers and Swing programmers are using different building blocks
  - UI Designers compose designs in tools like Photoshop and Illustrator
  - The building blocks they use have direct analogs in Java 2D, but not always directly in Swing

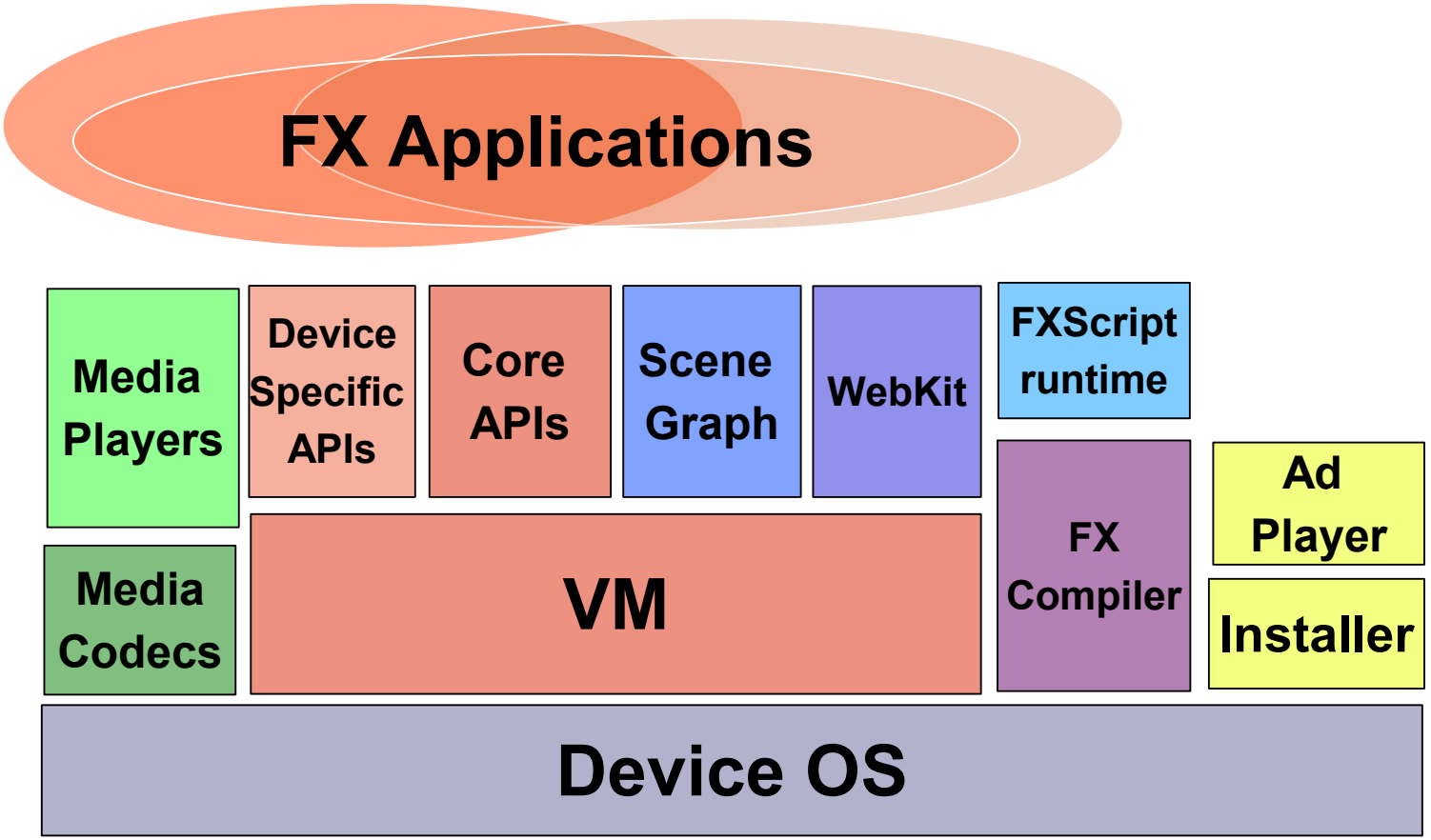
# A Basic Java GUI: Not Very Pretty



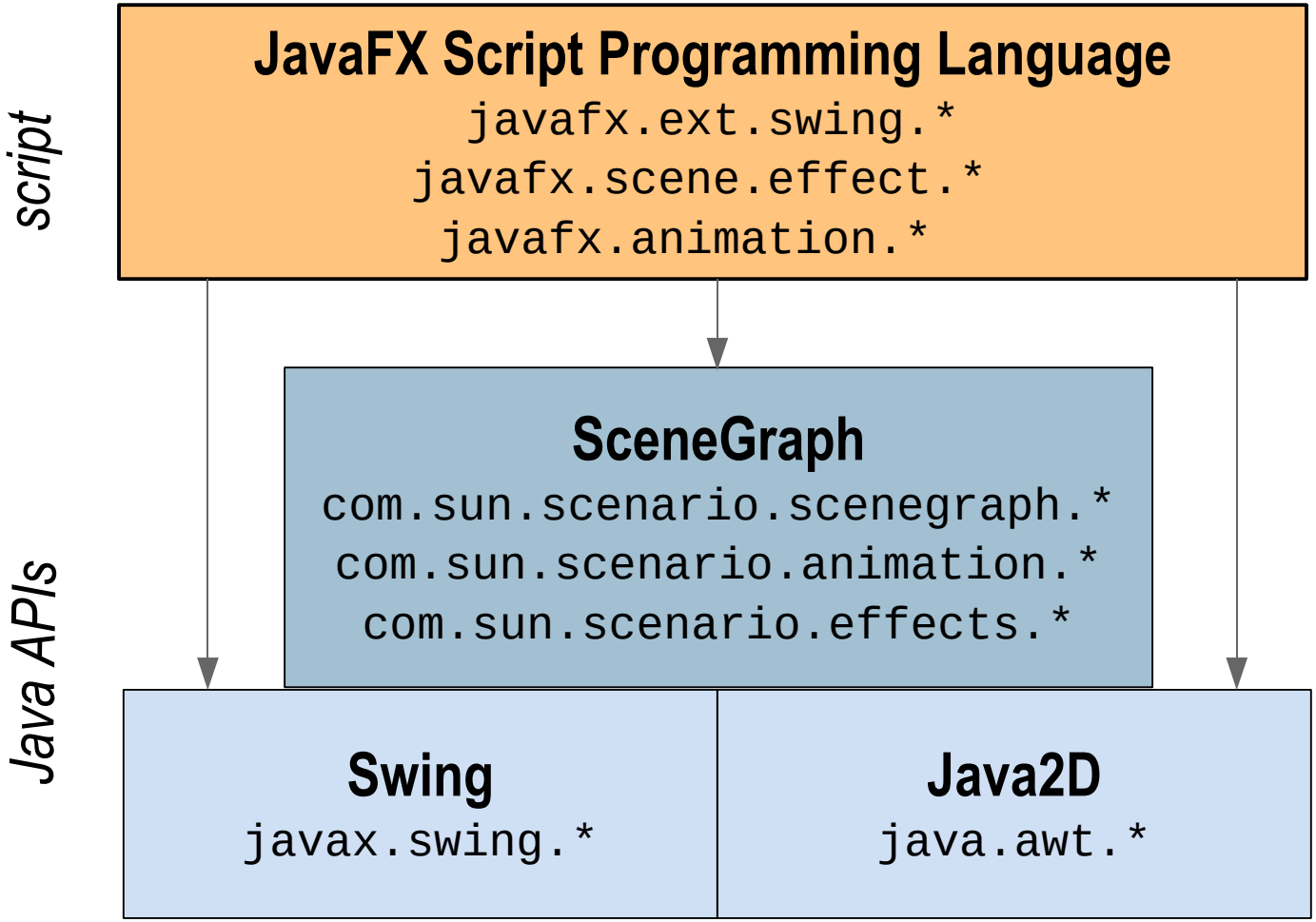
# Java 2D API

- To match the designs of UI designers requires using Java 2D API
- Java 2D API doesn't have compositional behavior
  - Makes it too complex for many programmers to use efficiently
- JavaFX combines elements of Swing with Java2D
- Goal is to move away from direct Swing usage to a node based scene graph
  - More like 3D (which will also be supported)

# JavaFX Platform Architecture

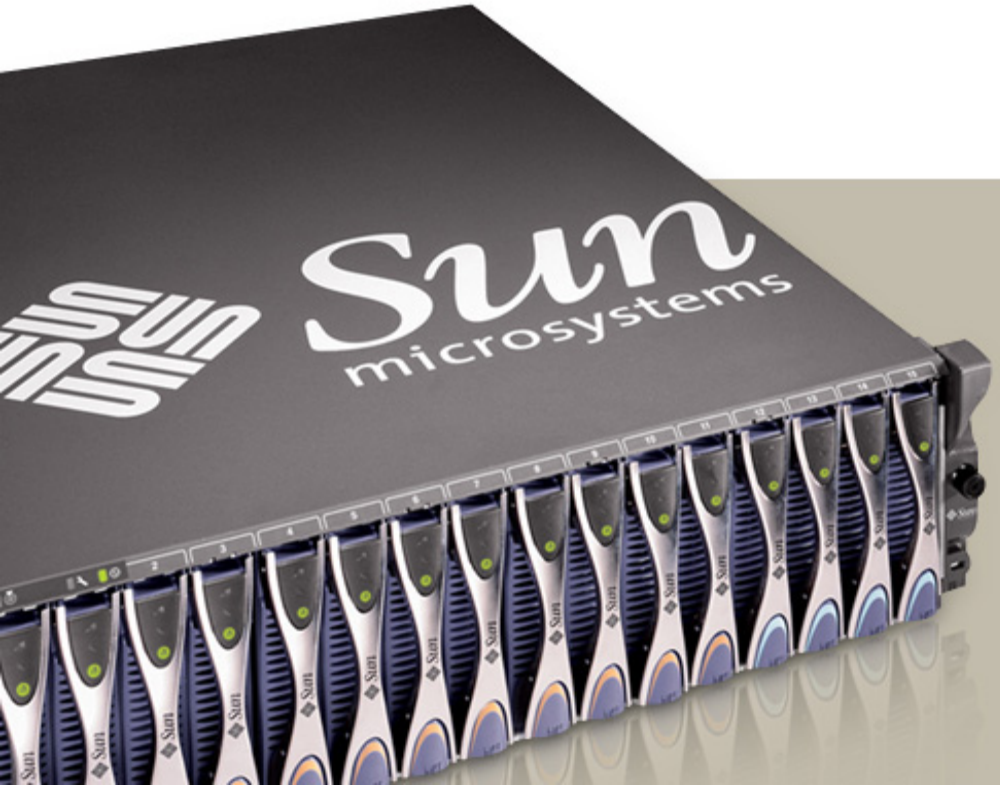


# JavaFX Technology Stack




*Note: JavaFX Script programs can call any Java APIs*

# JavaFX Script



# JavaFX Script Basics

- Declarative, statically-typed scripting language
- Facilitates rapid GUI development
- Many cool, interesting language features
- Runs on the Java Virtual Machine
- Deployment options same as Java programs
  - Applet, Application, WebStart
- Fully utilizes Java class libraries behind the scenes
- For content designers and media engineers

# Declarative Syntax

- Stop thinking in Java, start thinking more scripting
- Think “what”, not “how”
- In Java we need to program how a GUI is displayed
  - Layout managers, Panels, Components, etc
- JavaFX is more like HTML
  - Tell JavaFX what you want
  - Let JavaFX figure out how to display it
  - No porting between screens (desktop, mobile, etc)

# Basic JavaFXScript Class

Syntax is Java-like with shades of JavaScript

```
class HelloWorldNode extends CustomNode {
    public var text:String;
    public override function create(): Node {
        return Text {
            x: 10, y: 50
            font: Font {
                size: 50
            }
            content: bind text
        }
    };
}
```

# JavaFX Types

- Number
- Integer
- Boolean
- String
- Duration
- Void can be used for function return types

# Declarations

- Variables
- `var fred:Number;`
- Constants
- `def PI:Number = 22 / 7;`
- Access modifiers

> **`public`** – everyone should know this
> **`public-init`** – can only be set in constructor
> **`public-read`** – can only be set by the object

# Sequences

- Sequences

```
var time:Duration[] = [60ms, 60s, 60m, 60h];
var days = [1..31];
```

- Insert, delete, membership and reverse

```
insert 5s into time;
delete 31 from days;
var revDays = reverse days;
if (!(31 in days) or (30 in days)) “February”
```

- Slice via range and predicate

```
var oddDays = days[n | (n % 2) == 1];
var firstThree = time[0..<2]; //Include 3
```

# Classes

```
class Person {
  var name: String;
  var parent: Person inverse
    Person.children;
  var children: Person* inverse
    Person.parent;
  function getNumberOfChildren(): Number {
    return sizeof this.children;
  }
}
```

# Binding in JavaFX

- Cause and Effect - Responding to change
- bind operator - Allows dynamic content to be expressed declaratively
- Automated by the system—Rather than manually wired by the programmer
- You just declare dependencies and the JavaFX runtime takes care of performing updates when things change
- Eliminates listener patterns

# Binding

- Dependency-based evaluation of expressions
- Enables expression of dynamic data relationships

```
var x = 10;
var y = bind x + 100;
x = 50;
y == 150; // true
```

- Any expression can be bound
  - conditionals, loops, functions, etc...
- bind “with inverse” when 2-way is needed
- lazy binding to only evaluate when used

# Binding Example

```
Class SceneElement extends SceneNode {
  attribute sx: Number;
  attribute sy: Number;
  attribute r: Number;
  attribute canSee: Boolean;

  public function create(): Node {
    return Circle {
      radius: bind r
      centerX: bind sx
      centerY: bind sy
      fill: Color.RED
      translateX: bind sx + transX
      translateY: bind sy + transY
      visible: bind canSee
    }
  }
```

# Binding Example

```
    function update(nx: Number, ny: Number) {
      sx = nx;
      sy = ny;

      // Even one line if statement must have {}
      if (nx > 0 and ny > 0) {
        canSee = true;
      } else {
        canSee = false;
      }
    }
}
```

# Bound Functions

- Changes to the internal values of a function will cause the entire function to be reevaluated

>Used in conjunction with **bind**

```
var scale = 1;
function makePoint(xt:Number, yt:Number): Point {
    return Point {
        x: xt * scale, y: yt * scale
    };
}
var x = 3;
var y = 3;
var myPoint = bind makePoint(x, y);
x = 5;
FX.println(myPoint.x) //The value is 5
scale = 3;
FX.println(myPoint.x) //The value is 5
```

# Example – Bounded Function

```
var scale = 1;
bound function makePoint(xt:Number, yt:Number): Point {
    return Point {
        x: xt * scale, y: yt * scale
    };
}

var x = 3;
var y = 3;
var myPoint = bind makePoint(x, y);
x = 5;
FX.println(myPoint.x) //The value is 5
scale = 3;
FX.println(myPoint.x) //The value is 15
```

# Expressions

- while, try – Same syntax as Java
- if statement must always have braces
  - Even for single line
- Use **`and`**/**`or`** rather than **`&&`**/**`||`**
- **`for (i in [0..10]) ...`**
- **`for (i in [0..10] where i%2 == 0)`**
  **`...`**
- **`for (i in [0..10], j in [0..10]) ...`**

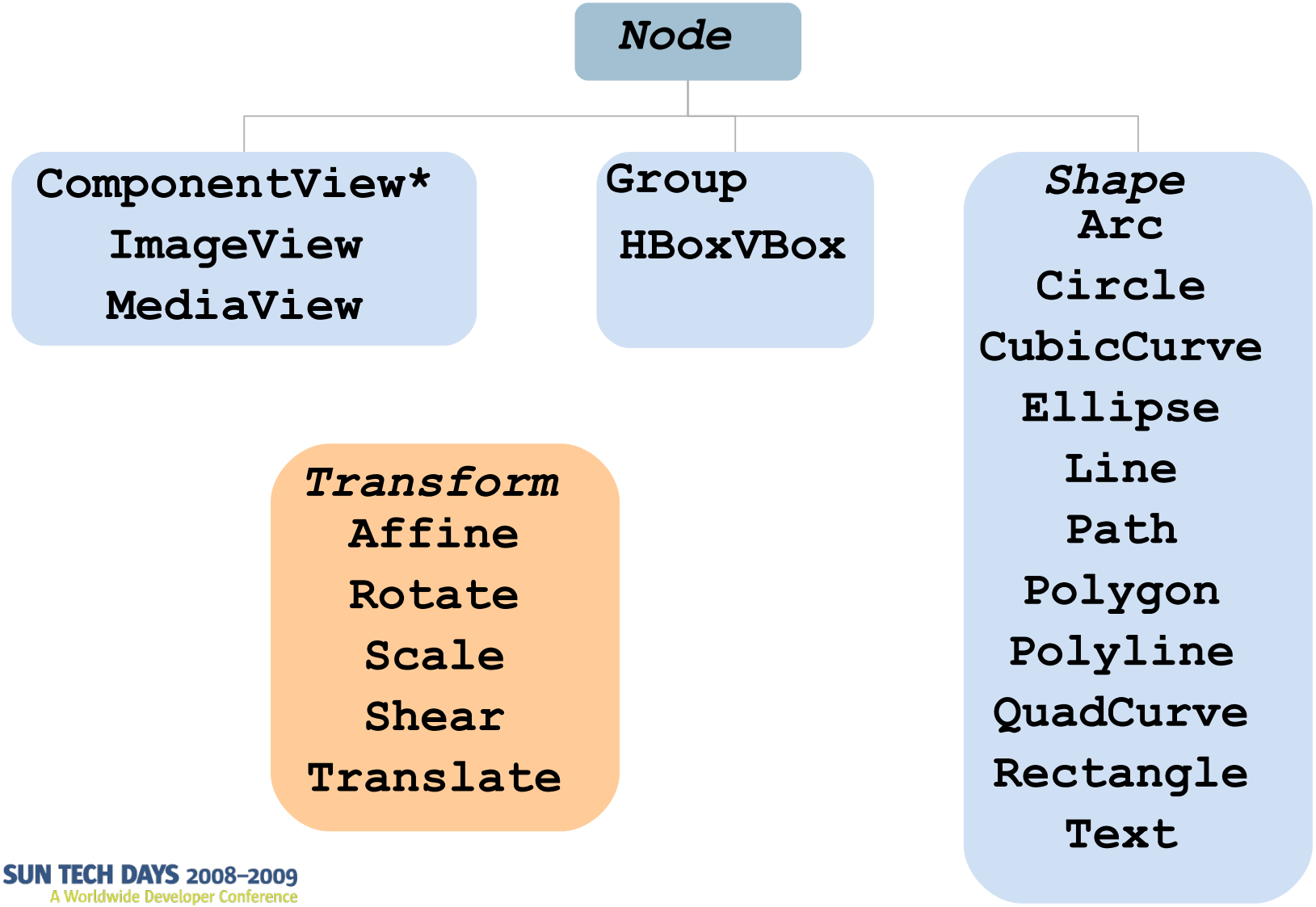
# Triggers

- Associate a block of code to a variable
- When the value of the variable changes, the code is executed

>Similar to **PropertyChangeListener**

```
//oldValue is optional
var text on replace oldValue {
   FX.println("Old value = '{oldValue}'");
   FX.println("New value = '{text}'");
}
text = "Hello"
Old value = ''
New value = 'Hello'
```

# Scene Graph Nodes: `javafx.gui.*`



# Custom Shapes

- Two ways of defining custom shapes

>Combining existing shapes

>Drawing a totally new shape

- Combine existing shape with `ShapeIntersect` or `ShapeSubtract`

>Operate on the union of one set of shape with another set

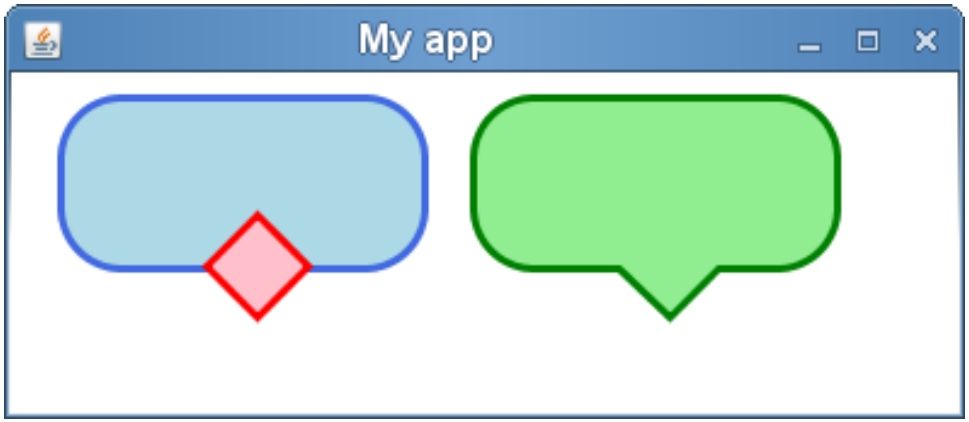
- Draw new shapes with `Path` and path elements

>Path elements include `MoveTo`, `ArcTo`, `HLine`, `VLine`, `QuadCurve`, etc.

- Can be manipulated like a regular geometric shape

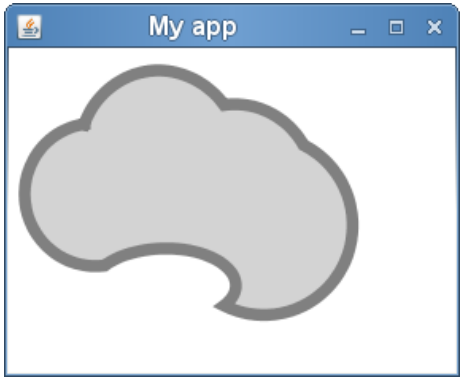
# Example – ShapeIntersect

```
ShapeIntersect {
   transforms: [ Translate { x: 170 } ]
   fill: Color.LIGHTGREEN
   stroke: Color.GREEN
   strokeWidth: 3
   //Union of the 2 shapes
   a: [rectangle diamond ]
}
```



# Example – Path

```
Path {
    fill: Color.LIGHTGRAY
    stroke: Color.GRAY
    strokeWidth: 3
     elements: [
         MoveTo { x: 15 y: 15 },
         ArcTo { x: 50 y: 10 radiusX: 20
              radiusY: 20 sweepFlag: true},
         ArcTo { x: 70 y: 20 radiusX: 20
              radiusY: 20 sweepFlag: true},
         ArcTo { x: 50 y: 60 radiusX: 20
              radiusY: 20 sweepFlag: true},
         ArcTo { x: 20 y: 50 radiusX: 10
              radiusY: 5 sweepFlag: false},
         ArcTo { x: 15 y: 15 radiusX: 10
              radiusY: 10 sweepFlag: true},
     ]
}
```



# Effects: javafx.gui.effects.*

*Effect*

Blend
Bloom
Glow
Lighting
Flood
Reflection
Shadow
InnerShadow
DropShadow

DisplacementMap
PerspectiveTransform
InvertMask
ColorAdjust
SepiaTone

GaussianBlur
MotionBlur

*Light*

*DistanceLight*
*PointLight*
*SpotLight*

# Some Effects Supported In JavaFX

```
effect: SepiaTone { level: 0.5 }
```

```
effect: Glow { level: 0.7 }
```

```
effect: GaussianBlur {
    input: SepiaTone {
        level: 0.5 }
    radius: 10.0
}
```

```
effect: Reflection {
    fraction: 0.7
}
```

Original image

# Lighting Effect

```
effect: Lighting{
    surfaceScale: 7
    light: DistantLight{
        azimuth: 90
        elevation: 30
    }
}
```

```
effect: Lighting{
    surfaceScale: 7
    light: SpotLight {
        x: 0 y :0 z: 50
        pointsAtX: 10
        pointsAtY: 10
        pointsAtZ: 0
    }
}
```

# Animation - javafx.animation.*

*TimeLine*
autoReverse
INDEFINITE
keyFrames
repeatCount
running
toggle

*KeyFrame*
action
canSkip
time
timelines
values

InterPolator

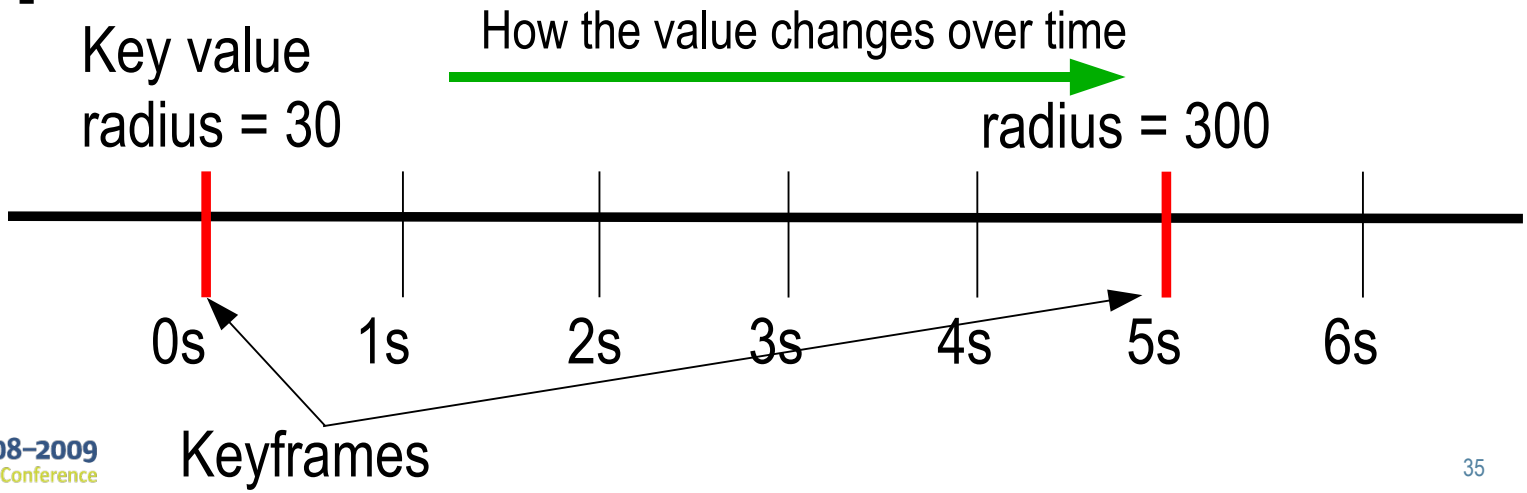
DISCRETE
EASEBOTH
EASEIN
EASEOUT
LINEAR

# Animation

- Timelines handles the animation in JavaFX
- Timelines are first-class citizen in the language along with the duration time constants (1s, 10s)
- They can have one or more KeyFrames
- Methods: start(), stop(), pause(), resume()
- Properties: autoReverse, repeatCount, toggle
- Timelines are nestable

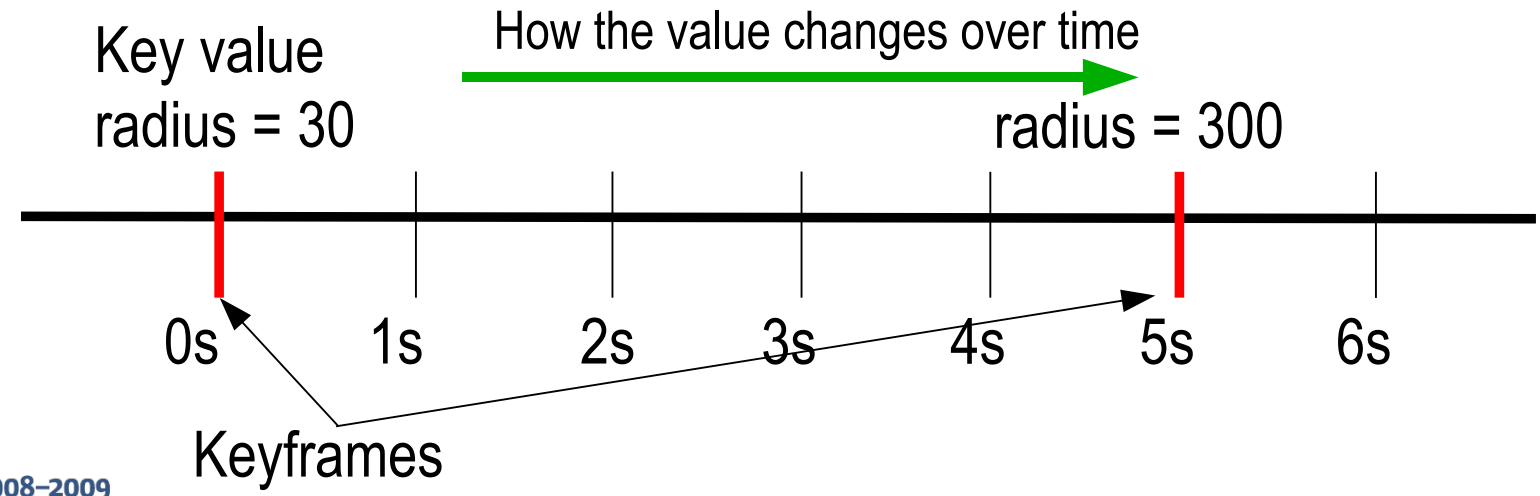
# Example – Defining Key Frames

```
Timeline {
    keyFrames: [
        KeyFrame {
            time: 0s
            values: [ radius => 30 ]
          }
        KeyFrame {
            time: 5s
            values: [
                radius => 300 tween Interpolator.LINEAR
            ]
        }
    ]
}
```



# Example – Shorthand Notation

```
Timeline {
    keyFrames: [
        at(0s) {
            radius => 30
        }
        at(5s) {
            radius => 300 tween Interpolator.LINEAR
        }
    ]
}
```



# Transitions

- Animations classes to animate common node attributes

>Position, rotation, opacity, etc.

- Out of the box transitions

>`RotateTranstion` – rotation

>`FadeTransition` – opacity

>`TranslateTransition` – move a node along a straight line

>`PathTransition` – move an object along a defined path

>`ScaleTranstion` – grows or shrinks a node

# Example – Path Transition

```
var earth:ImageView = ImageView {
    x: sx y: sy
    image: Image { url: "file:////home/cmlee/earth.png" }
}
def path = [
   MoveTo { x: sx y: sy}
   ArcTo { x: 0 y: 200
      radiusX: 50 radiusY: 50 sweepFlag: true
   }
];
var aniPath:PathTransition = PathTransition {
    node: earth
    path: AnimationPath.createFromPath(Path {elements: path })
    duration: 1500ms
}

aniPath.playFromStart();
```

# JavaFX Media Design Goals

- Media Playback is of primary importance
- Simple API: only a few lines of coded needed
- Cross platform A/V format required
- Native support also desirable
  - “Mash ups”
  - Viewing local media
- Zero Configuration plug in support
  - Drop in format support
- Going beyond rectangular video
  - Support lightweight rendering

# Java Media Components (JMC)

- Cross Platform Video Format Support
  - Encode once, play anywhere
  - Over time, multiple formats may be supported
    - Sun Open Media Stack (OMS)
- Leverages native platform
  - Windows
    - Play Windows Media via DirectShow
    - Flash via the ActiveX control
  - Mac
    - Quicktime and Core Audio/Video.
  - Linux and Solaris
    - GStreamer

# Media in JavaFX

- Media classes are part of javafx.gui package
- Media, MediaPlayer and MediaView
  - MediaView is a Node
    - has a MediaPlayer
    - MediaPlayer has a Media object.
  - MediaView may be rotated, translated, sheared, and have filter effects applied to it.
  - Multiple views may be bound to single player.
- MediaTimers allow functions to be invoked at key points in Media.
- Other media events may be triggered

# JavaFX Media Example

```
var media = Media{source: "movie.mov"};
var player = MediaPlayer{media: media, autoPlay:true};

var mediaView = MediaView{
  // To enable audio only, don't create MediaView
  MediaView{
    mediaPlayer: player,
    onMouseClicked: function(e) { // Play/pause control
      if (player.paused) {
        player.play();
      } else {
        player.pause();
      }
    }

    rotate: 90;  // Rotate
}
```

# JavaFX NetBeans IDE Plugin

- New for NetBeans 6.1 and later
- Supports conventional development cycle
  - edit, compile, run, test
  - Also has preview mode (avoid compile/run)
- Specific project type for JavaFX
- Automatic installation of JavaFX SDK
- Editor supports code completion, drag and drop of components
  - Not fully polished yet

# Further Information

http://www.javafx.com

http://www.sun.com/javafx

http://openjfx.org

http://jfx.wikia.com/wiki/Planet_JFX_Wiki

http://learnjavafx.typepad.com/

http://blogs.sun.com/chrisoliver


Sun microsystems
THANK YOU
Simon Ritter
simon.ritter@sun.com
SUN TECH DAYS 2008–2009
A Worldwide Developer Conference