

ORACLE®

# JavaFX 9: New & Noteworthy

Kevin Rushforth & Jonathan Giles  
Java Client Group  
October 26, 2015



# Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

# Agenda

- 1 JDK 8 Update Releases
- 2 Support for Jigsaw Modularity
- 3 JEP 253: UI control skins and CSS API
- 4 High DPI
- 5 Other new public APIs
- 6 Q & A



# JDK 8 Update Releases

# JDK 8u40

- Released in March 2015
- Added a number of important missing features:
  - Accessibility (Windows and Mac)
  - New Controls
    - Spinner
    - Filtered Text
    - Dialogs (Alerts, TextInputDialog, ChoiceDialog)
  - LCD Text on Canvas
  - 3D User-Defined Normals
- Several bugs fixed in controls, charts, layout, etc.



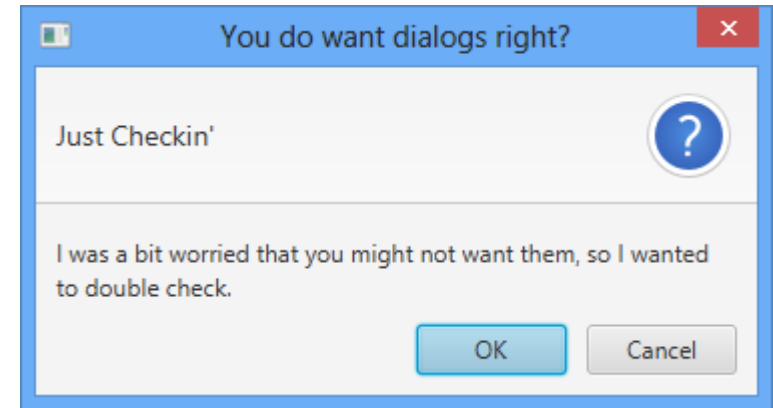
# Accessibility

- A full-featured minimal API and implementation
- Supports native accessibility (VoiceOver, Narrator)
- All built-in controls and charts are accessible
- Developers can make their own controls accessible



# Dialogs

- Finally built-in to JavaFX!
- Simple Alert API for prompting users
- More advanced Dialog / DialogPane for total customization
- OS-specific button ordering





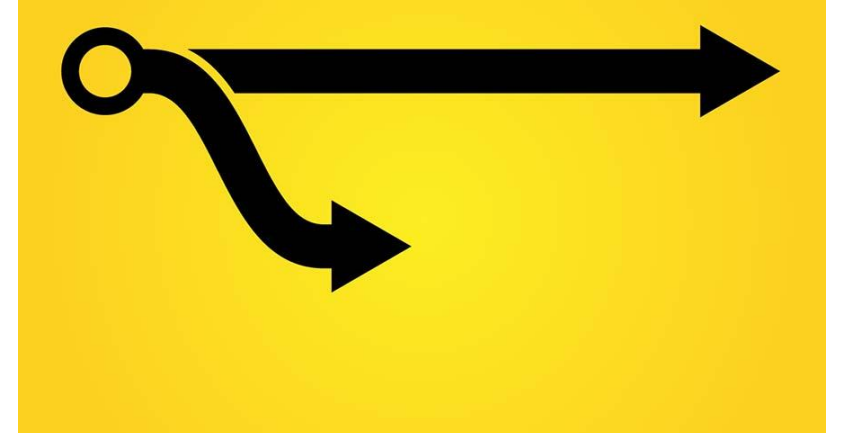
# JDK 8u60

- Released in August 2015
- Focus was on bug fixing and stabilization of the platform
- Small number of new features:
  - Updated to newer version of WebKit
  - Added minimal High-DPI support on Windows
    - Enabled by default when Windows UI scale is  $\geq 150\%$
    - No API added to provide application control (stay tuned)



## After JDK 8u60

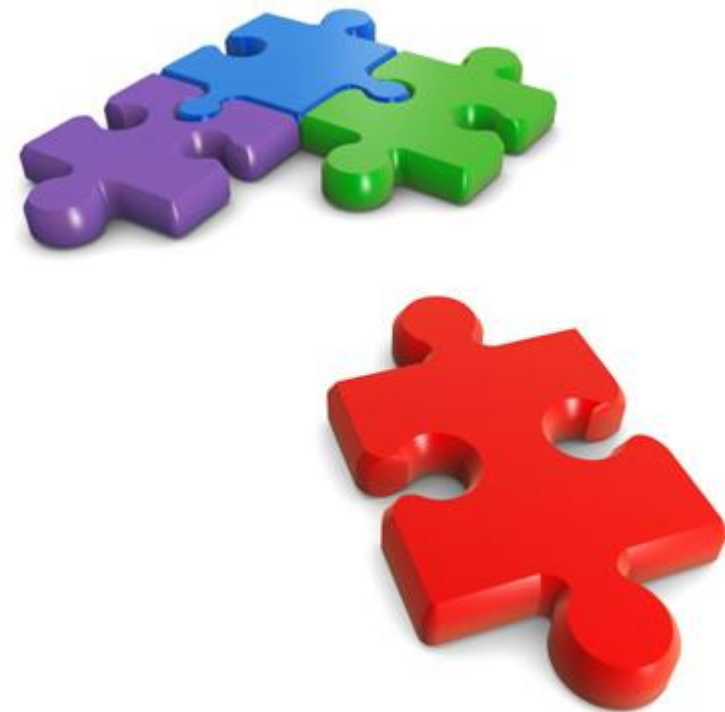
- All development effort is on JDK 9
- A *few* critical bug fixes can be backported to our quarterly releases
  - 8u66 just released last week
  - 8u72 planned for Jan 2016
  - 8u76 planned for Apr 2016
  - Etc.







# Coming up in JavaFX for JDK 9

# Support for Jigsaw Modularity



# Jigsaw Modularity

- Jigsaw Modularity is ***the*** primary feature for JDK 9
  - JEP 200: The Modular JDK (Umbrella)
    - JEP 201: Modular Source Code ✓
    - JEP 220: Modular Run-Time Images ✓
    - JEP 260: Encapsulate Most Internal APIs 
    - JEP 261: Module System 
- Modularizing JavaFX is our main goal for JDK 9!
  - JDK-8092093: Modularization support for JavaFX (Umbrella)



# Jigsaw Modularity: Properties of Modules

- A module is a collection of packages
  - A package belongs to exactly one module
  - Split packages have long been discouraged; with modules they are forbidden
- A module lists its inputs (requires) and outputs (exports)
- Only explicitly exported packages are visible
  - Provides strict encapsulation of implementation details
  - Internal “API” is no longer accessible
  - Accessing a non-exported package will result in a compilation or runtime error
    - And no, you can’t simply use reflection to call `setAccessible`
    - It can be overridden with “-XaddExports” command line switch in extreme need

# Jigsaw Modularity: JavaFX Modules

- JavaFX source code already organized as modules (as of JDK 8)

```
modules/  
  base/  
  controls/  
  graphics/  
  ...
```



# Jigsaw Modularity: JavaFX Modules

- In general, modules are named "javafx.nnnnn"
  - where "nnnnn" is the directory name under the modules directory.

```
modules/  
  base/           // javafx.base module  
  controls/       // javafx.controls module  
  graphics/       // javafx.graphics module  
  ...
```





# Jigsaw Modularity: JavaFX Modules

- JavaFX classes and resources will be linked into the JDK image
  - No more jfxrt.jar
- SWT interop will still be delivered separately:
  - jfxswt.jar file delivered with the JDK
  - Cannot be linked into runtime image because it depends on third-party swt.jar
  - jfxswt.jar will be an "automatic" module (meaning no module-info.class)
  - Implementation classes will go into a named javafx.internal.swt module



# Jigsaw Modularity: JavaFX Modules

## JRE Modules

### Public

- `javafx.base`
- `javafx.controls`
- `javafx.fxml`
- `javafx.graphics`
- `javafx.media`
- `javafx.swing`
- `javafx.web`

### Internal

- `javafx.deploy` [closed]
- `javafx.internal.swt`

## JDK Modules

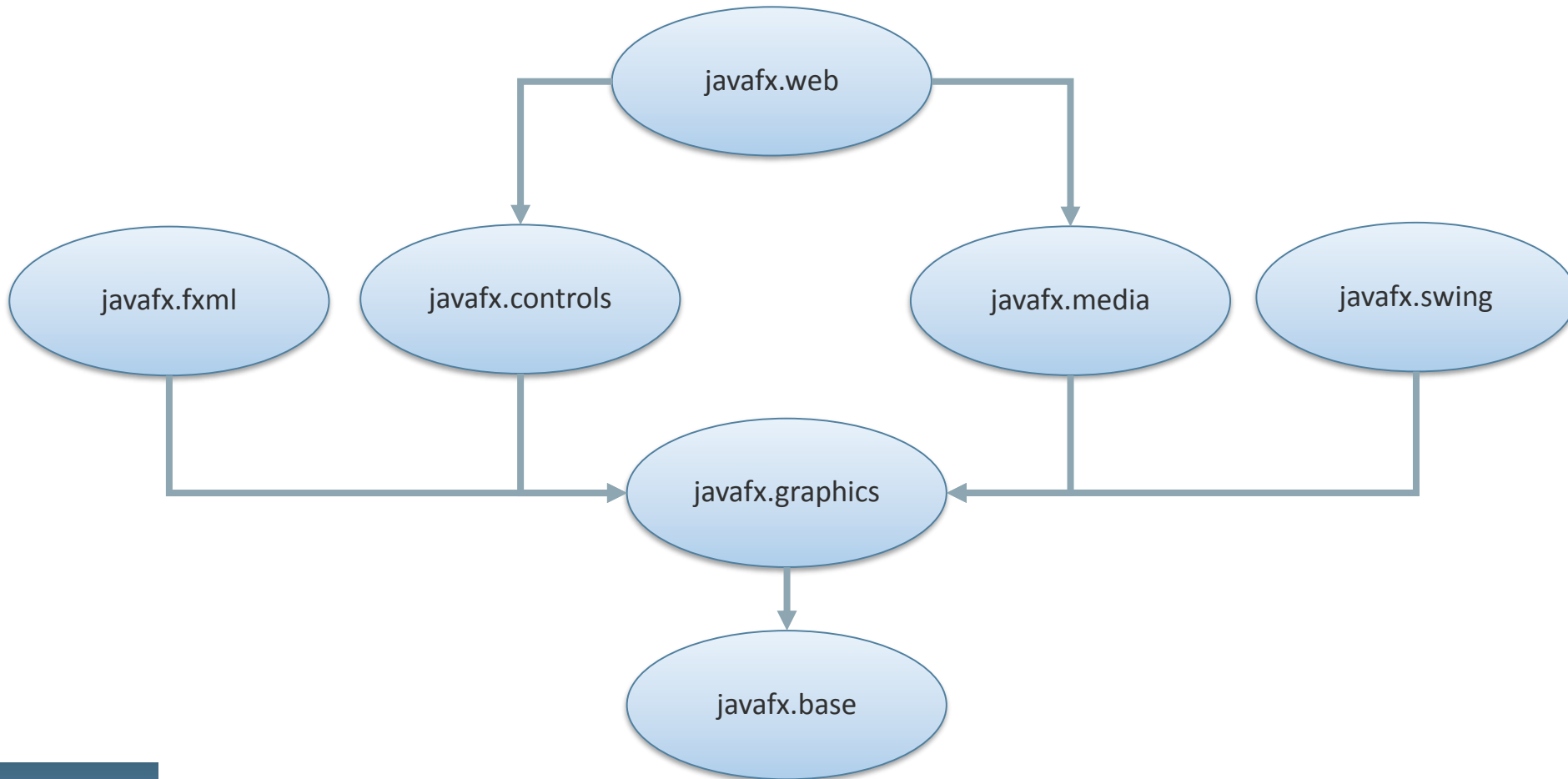
### Public

- `javafx.jmx`
- `jdk.packager`
- `jdk.packager.services`



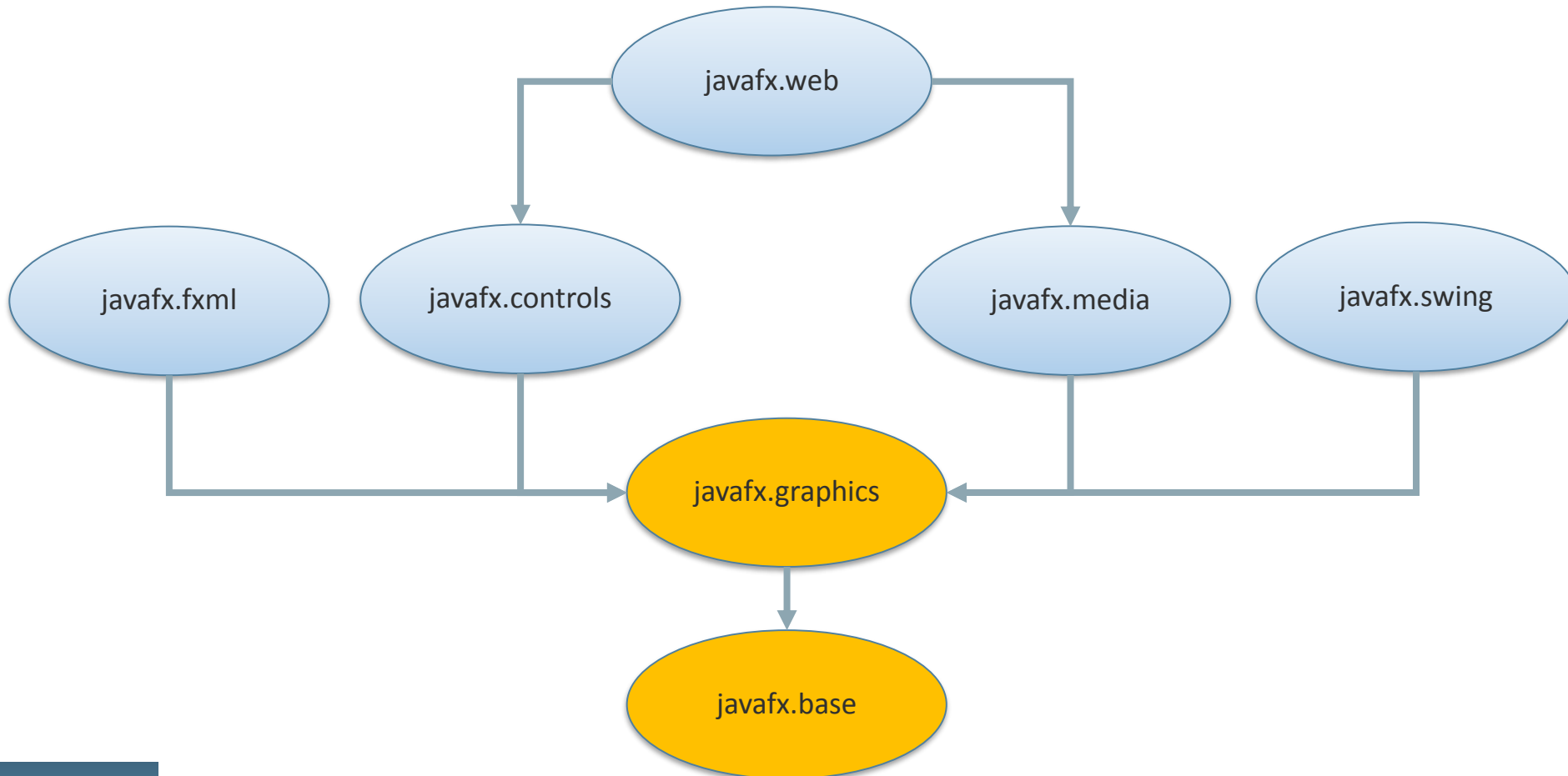
# Jigsaw Modularity: JavaFX Modules

- JavaFX module graph for runtime (JRE) modules: transitive reduction



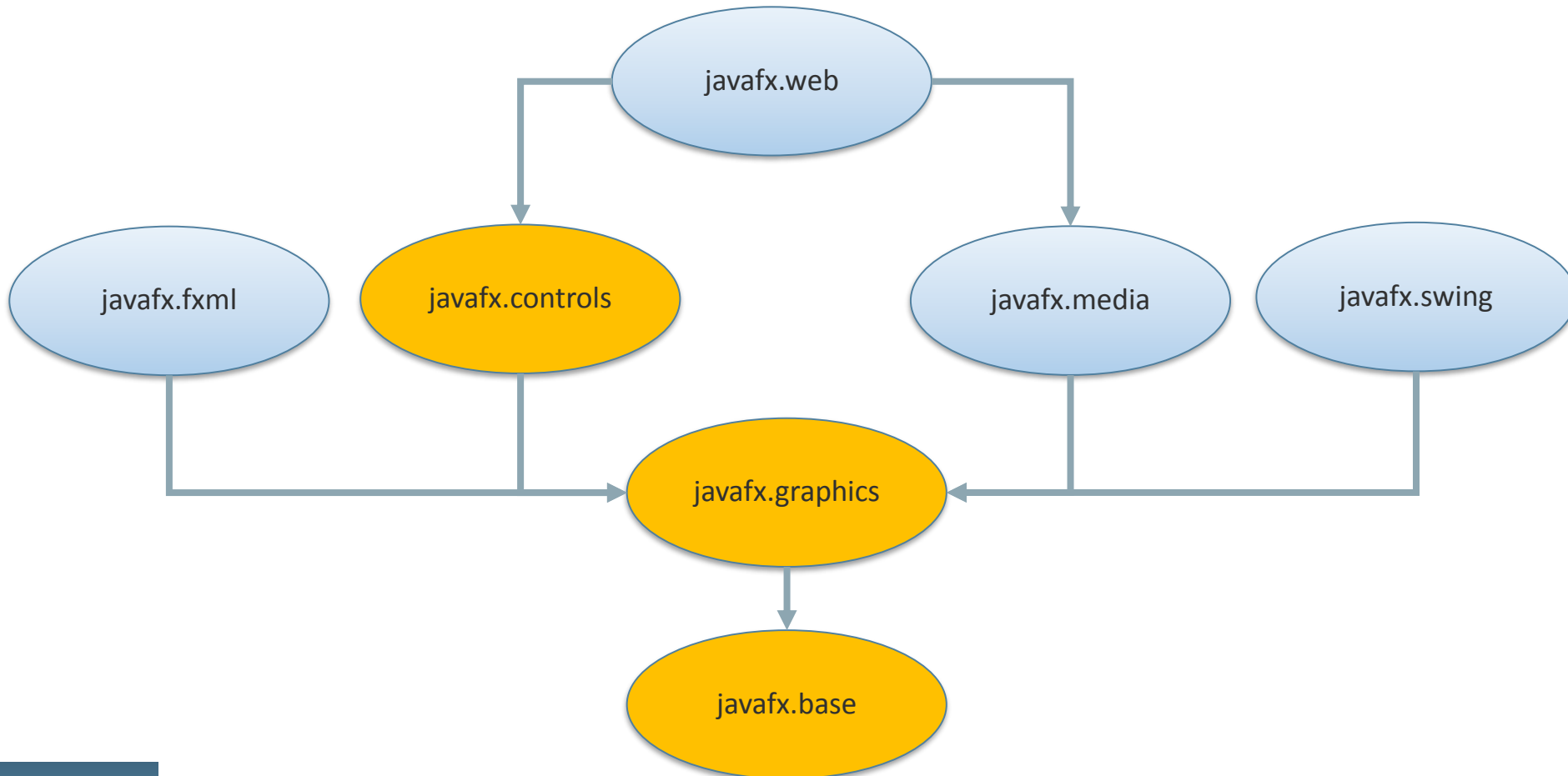
# Jigsaw Modularity: JavaFX Modules

- All JavaFX applications require javafx.graphics



# Jigsaw Modularity: JavaFX Modules

- All JavaFX UI applications require `javafx.controls`



# Jigsaw Modularity: Application Impact

- Classes in non-modular applications are in the “unnamed” module
  - By default, the unnamed module reads (requires) all named modules in the system
  - Can access all publicly exported packages of all modules with no app changes
- Modular applications need to list their dependencies in module-info.java
  - Only public types of required modules are accessible
  - Here is a minimal module-info.java for an application that uses the javafx.controls, .graphics, and .base modules (controls re-exports graphics and base)

```
module my.app {  
    requires javafx.controls;  
}
```

# Jigsaw Modularity: JavaFX Packages

- Only publicly documented packages are exported
  - All are in the "javafx.\*" namespace

```
javafx.base module
package javafx.beans;
package javafx.beans.binding;
package javafx.beans.property;
package javafx.beans.property.adapter;
package javafx.beans.value;
package javafx.collections;
package javafx.collections.transformation;
package javafx.event;
package javafx.util;
package javafx.util.converter;
```

```
javafx.controls module
package javafx.scene.chart;
package javafx.scene.control;
package javafx.scene.control.cell;
package javafx.scene.control.skin;
```

```
javafx.fxml module
package javafx.fxml;
```

# Jigsaw Modularity: JavaFX Packages

## **javafx.graphics module**

```
package javafx.animation;  
package javafx.application;  
package javafx.concurrent;  
package javafx.css;  
package javafx.css.converter;  
package javafx.geometry;  
package javafx.print;  
package javafx.scene;  
package javafx.scene.canvas;  
package javafx.scene.effect;  
package javafx.scene.image;  
package javafx.scene.input;  
package javafx.scene.layout;  
package javafx.scene.paint;  
package javafx.scene.shape;  
package javafx.scene.text;  
package javafx.scene.transform;  
package javafx.stage;
```

## **javafx.media module**

```
package javafx.scene.media;
```

## **javafx.swing module**

```
package javafx.embed.swing;
```

## **javafx.web module**

```
package javafx.scene.web;
```



# Jigsaw Modularity: Availability

- Changes for JavaFX modularity are in an OpenJFX jake sandbox:
  - <http://hg.openjdk.java.net/openjfx/sandbox-9-jake/rt>
- Corresponds to JDK jigsaw/jake sandbox
- Early access Jigsaw builds on java.net include JavaFX modules

# JEP 253

## Prepare JavaFX UI Controls & CSS APIs for Modularization



# Overview

- 1 ➤ JDK 8 and Earlier: State of the Nation
- 2 ➤ Motivation for JEP 253
- 3 ➤ JDK 9: What are we doing?
- 4 ➤ Current Progress

# JDK 8 and Earlier :: State of the Nation :: UI Controls

JDK 8 ships with approximately 64 UI controls (or critical utility classes):

Accordion	CheckMenuItem	DateCell	Labeled	Pagination	ScrollBar	SplitPane	TextField	ToolBar
Alert	ChoiceBox	DatePicker	ListCell	PasswordField	ScrollPane	Tab	TextInputControl	Tooltip
Button	ChoiceDialog	Dialog	ListView	PopupControl	Separator	TableCell	TextInputDialog	TreeCell
ButtonBar	ColorPicker	DialogPane	Menu	ProgressBar	SeparatorMenuItem	TableColumn	TitledPane	TreelItem
Cell	ComboBox	Hyperlink	MenuBar	ProgressIndicator	Slider	TableView	Toggle	TreeTableCell
CheckBox	ContextMenu	IndexedCell	MenuItem	RadioButton	Spinner	TabPane	ToggleButton	TreeTableColumn
CheckBoxTreeItem	CustomMenuItem	Label	MenuItem	RadioMenuItem	SplitMenuItem	TextArea	ToggleGroup	TreeTableView
								TreeView

# JDK 8 and Earlier :: State of the Nation :: UI Controls

**Controversial Statement Time:**

# JDK 8 and Earlier :: State of the Nation :: UI Controls

## **Controversial Statement Time:**

JavaFX 8 now has a full complement of expected UI controls

# JDK 8 and Earlier :: State of the Nation :: UI Controls

## **Controversial Statement Time:**

JavaFX 8 now has a full complement of expected UI controls

What is missing now is ‘smoothing out’ a lot of API:  
filling gaps in functionality, fixing bugs, and  
improving support for third party UI controls

# JDK 8 and Earlier :: State of the Nation :: UI Controls

## **Controversial Statement Time:**

JavaFX 8 now has a full complement of expected UI controls

What is missing now is ‘smoothing out’ a lot of API:  
filling gaps in functionality, fixing bugs, and  
improving support for third party UI controls

JDK 9 is a ‘smoothing out’ release.

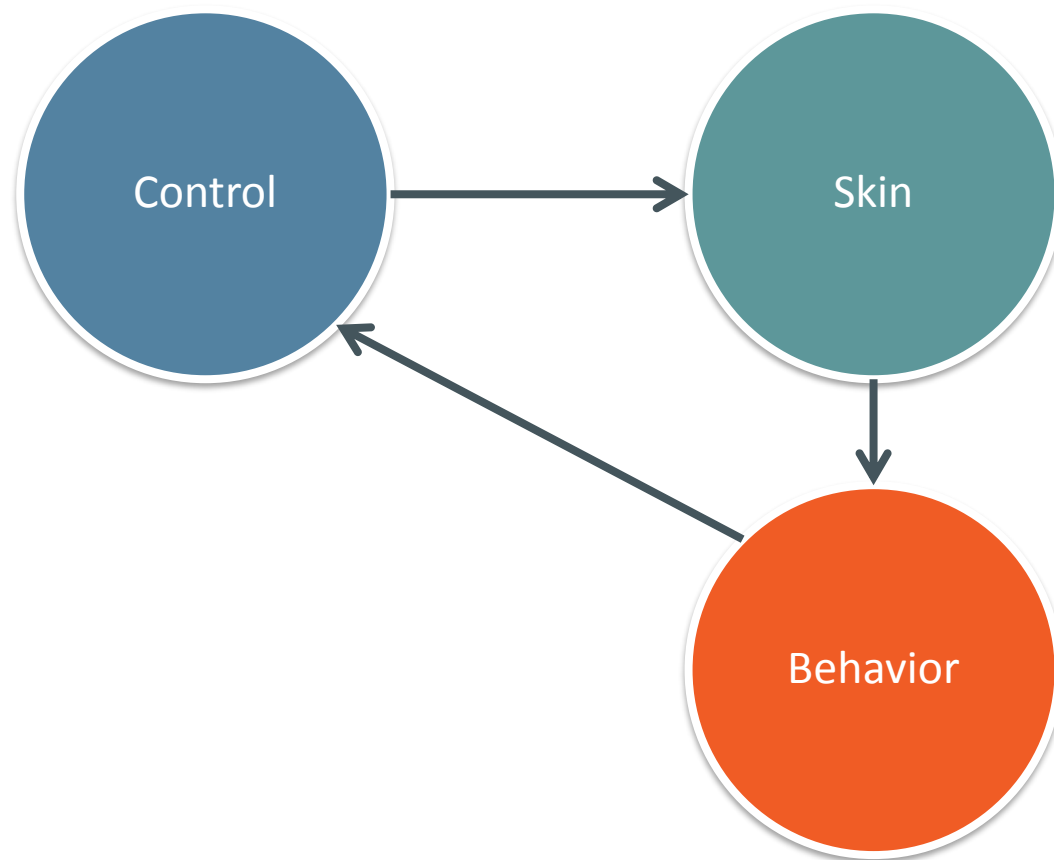


# JDK 8 and Earlier :: State of the Nation :: UI Controls

**How are UI Controls built?**

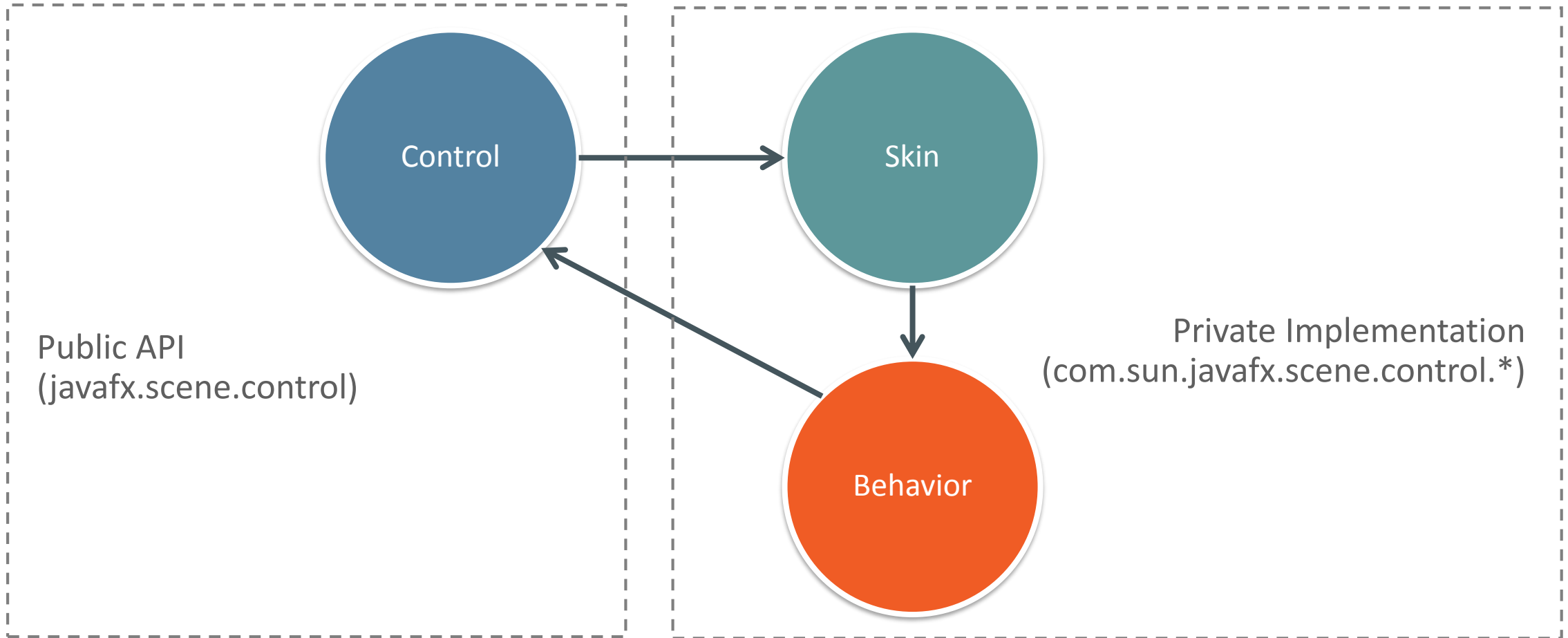
# JDK 8 and Earlier :: State of the Nation :: UI Controls

- Most UI controls are split into three components:



# JDK 8 and Earlier :: State of the Nation :: UI Controls

- Most UI controls are split into three components:



# JDK 8 and Earlier :: State of the Nation :: CSS

**What CSS APIs exist, and where are they used?**

# JDK 8 and Earlier :: State of the Nation :: CSS

CssMetaData  
Styleable  
StyleableProperty  
SimpleStyleable\*Property



Support for custom styleable properties coming in from CSS,  
e.g.  
-fx-custom-property: true;

# JDK 8 and Earlier :: State of the Nation :: CSS

CssMetaData  
Styleable  
StyleableProperty  
SimpleStyleable\*Property  
PseudoClass ←

Support for custom pseudoclass states.

e.g.  
.button:javaone {  
 ..  
}


# JDK 8 and Earlier :: State of the Nation :: CSS

CssMetaData  
Styleable  
StyleableProperty  
SimpleStyleable\*Property  
PseudoClass  
ParsedValue  
StyleOrigin



Generally unused publicly (heavily used internally however)...

# JDK 8 and Earlier :: State of the Nation :: CSS



```
CssMetaData  
Styleable  
StyleableProperty  
SimpleStyleable*Property  
PseudoClass  
ParsedValue  
StyleOrigin
```

javafx.css



# JDK 8 and Earlier :: State of the Nation :: CSS

Used to convert CSS values into Java values, e.g. `BooleanConverter` converts 'true' or 'false' strings into Boolean values.

 → \*Converter

# JDK 8 and Earlier :: State of the Nation :: CSS

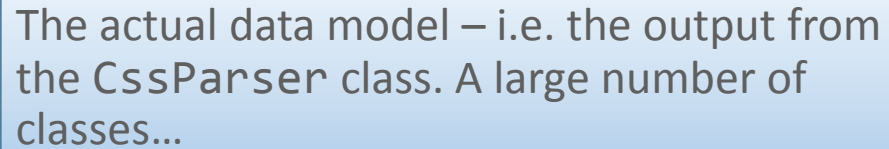
Implementation and API responsible for converting css syntax into a JavaFX CSS data model.

\*Converter  
CssParser



# JDK 8 and Earlier :: State of the Nation :: CSS

The actual data model – i.e. the output from the `CssParser` class. A large number of classes...



`*Converter`  
`CssParser`  
CSS data model:  
    `CalculatedValue`  
    `CascadingStyle`  
    `Combinator`  
    `CssError`  
    `Declaration`  
    `Rule`  
    `*Selector*`  
    `Size`  
    `Style`  
    `StyleCache`  
    `StyleClass`  
    `StyleMap`  
    `Stylesheet`

# JDK 8 and Earlier :: State of the Nation :: CSS

- \*Converter
- CssParser
- CSS data model:
  - CalculatedValue
  - CascadingStyle
  - Combinator
  - CssError
  - Declaration
  - Rule
- \*Selector\*
- Size
- Style
- StyleCache
- StyleClass
- StyleMap
- Stylesheet
- **StyleManager**

Contains the stylesheet state for a single scene. Includes API for adding user agent stylesheets to scenes, etc.

# JDK 8 and Earlier :: State of the Nation :: CSS

```
*Converter  
CssParser  
CSS data model:  
    CalculatedValue  
    CascadingStyle  
    Combinator  
    CssError  
    Declaration  
    Rule  
    *Selector*  
    Size  
    Style  
    StyleCache  
    StyleClass  
    StyleMap  
    Stylesheet  
StyleManager  
  
com.sun.javafx.css
```

# JDK 8 and Earlier :: State of the Nation :: CSS

```
CssMetaData  
Styleable  
StyleableProperty  
SimpleStyleable*Property  
PseudoClass  
ParsedValue  
StyleOrigin
```

`javafx.css`

```
*Converter  
CssParser  
CSS data model:  
    CalculatedValue  
    CascadingStyle  
    Combinator  
    CssError  
    Declaration  
    Rule  
    *Selector*  
    Size  
    Style  
    StyleCache  
    StyleClass  
    StyleMap  
    Stylesheet  
StyleManager
```

`com.sun.javafx.css`

# Motivation for JEP 253

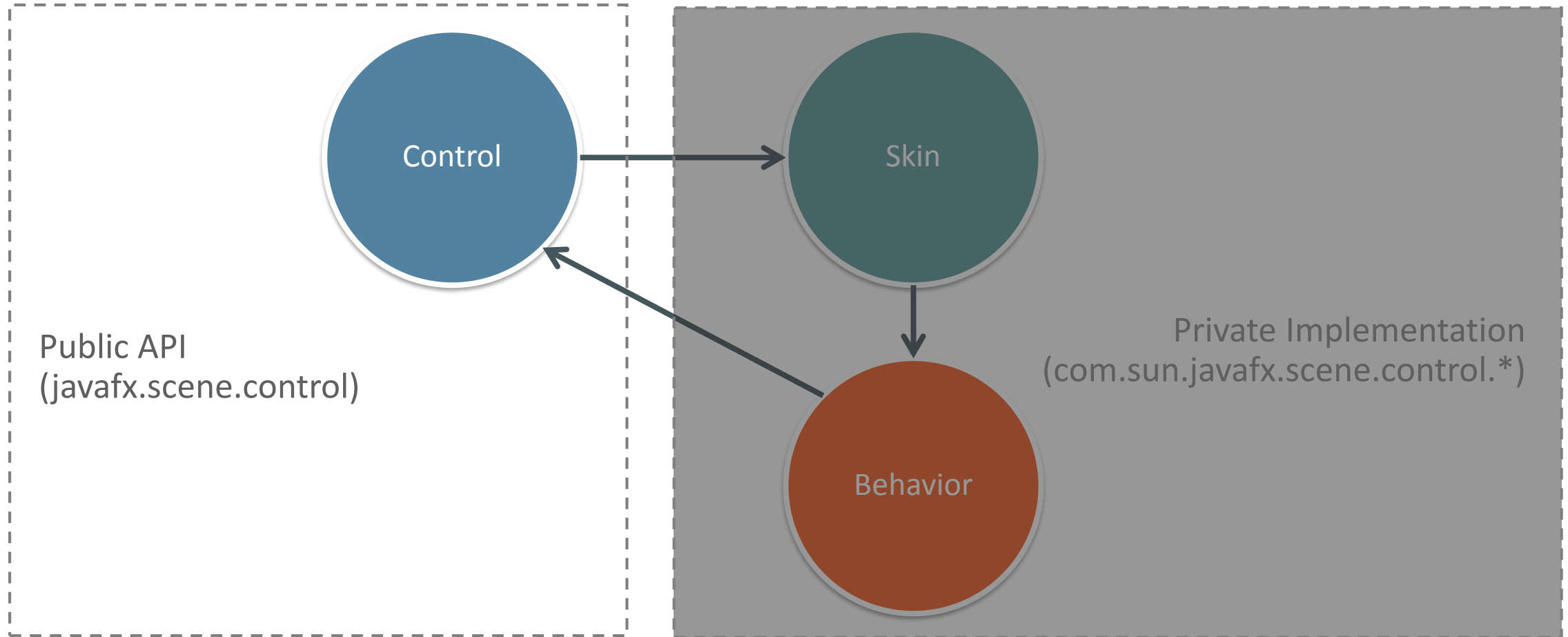
- In summary, UI Controls and CSS APIs have always had public and private APIs
  - Any project that does almost anything with custom UI controls or CSS ends up using implementation API in `com.sun.*` packages.
  - Always frowned upon, but never prevented (and impossible to prevent anyway).

# Motivation for JEP 253

- JDK 9 with Jigsaw modularity is a big game changer:
  - Up until JDK 9, developers could (ab)use API in `com.sun.*` packages.
  - JDK 9 enforces boundaries - `com.sun.*` becomes unavailable at compile time
- Many (if not most) JavaFX apps and libraries will fail to compile / execute under JDK 9.
- I asked community for JDeps output (will discuss this more later).
- In summary:
  - Almost all open source apps and most customer apps break!



# Motivation for JEP 253



# Motivation for JEP 253

```
CssMetaData  
Styleable  
StyleableProperty  
SimpleStyleable*Property  
PseudoClass  
ParsedValue  
StyleOrigin
```

javafx.css

```
*Converter  
CssParser  
CSS data model:  
    CalculatedValue  
    CascadingStyle  
    Combinator  
    CssError  
    Declaration  
    Rule  
    *Selector*  
    Size  
    Style  
    StyleCache  
    StyleClass  
    StyleMap  
    Stylesheet  
StyleManager  
com.sun.javafx.css
```

# Motivation for JEP 253

- In summary:
  - Because of modularity in JDK 9, by default the following private API disappear:
    - UI Controls
      - Skin implementations
      - Behaviors
    - CSS
      - CSS data model
      - Converters (for converting from CSS text into Java)
      - Parsers

# Motivation for JEP 253

- Why do people need to use these APIs?
  - Many custom controls base their skins on existing skins, e.g.
    - CustomTextFieldSkin extends TextFieldSkin
    - TextFieldSkin no longer available at compile time
  - Many custom controls want to support CSS
    - API to convert properties not public, for example
  - Many applications need functionality we just haven't got around to making public yet!

# Motivation for JEP 253

- Two options:
  - Ignore the problem, tell people this is their fault for using non-public APIs
    - Sure fire way to kill community and JavaFX ecosystem!
  - Identify ways to bring most important API into public packages with least amount of work
- Current approach is obviously the latter, split into two projects:
  - JEP 253: UI Controls and CSS APIs
  - Investigation into other API that is also being used (discussed later)

# JDK 9 :: What are we doing

- Useful JEP 253 URLs:
  - <http://openjdk.java.net/jeps/253>
  - [JDK-8076423](#) - Umbrella project
- JEP 253 is split into three subprojects:
  1. [JDK-8077916](#): Make UI control skins into public APIs
  2. [JDK-8077917](#): Improve support for input mapping
  3. [JDK-8077918](#): Review and make public relevant CSS APIs

# JDK 9 :: What are we doing

- Useful JEP 253 URLs:
  - <http://openjdk.java.net/jeps/253>
  - [JDK-8076423](#) - Umbrella project
- JEP 253 was split into three subprojects (now it is two):
  1. [JDK-8077916](#): Make UI control skins into public APIs
  2. ~~[JDK-8077917](#): Improve support for input mapping~~
  3. [JDK-8077918](#): Review and make public relevant CSS APIs

# JDK 9 :: What are we doing :: Progress Report

- [JDK-8077916](#): Make UI control skins into public APIs:
  - All relevant UI control skins have been moved to `javafx.scene.control.skin`
  - Each file has been cleaned up:
    - code reordered to follow standard layout style
    - JavaDocs written for all public API
  - Most importantly: the size of the API per class has been reduced to bare essentials
    - We can grow the API in future releases based on community feedback (but we can never shrink it!)
- Overall: Essentially this subproject is complete. Code is now in public JDK 9 repos for review.



# JDK 9 :: What are we doing :: Progress Report

- [JDK-8077917](#): Improve support for input mapping
  - New public InputMap API created in `javafx.scene.input`
  - Based on ~3 years part time research, tracked in [JDK-8091189](#)
  - All UI controls have been ported to use InputMap implementation
    - No known functional regressions, and all existing UI control unit tests (>7000) continue to pass
    - It was decided early on that all behaviors should remain private. Could make public in a future release.
- Overall: This project is ‘complete’, but community discussion has resulted in more questions being raised.
- In hindsight, this subproject should have been removed when behaviors were left private.
- As of now, this project has been removed.
  - Can do a ‘big bang’ release consisting of InputMap, public behaviors, and maybe Action framework
  - Better than doing partially now and living to regret it

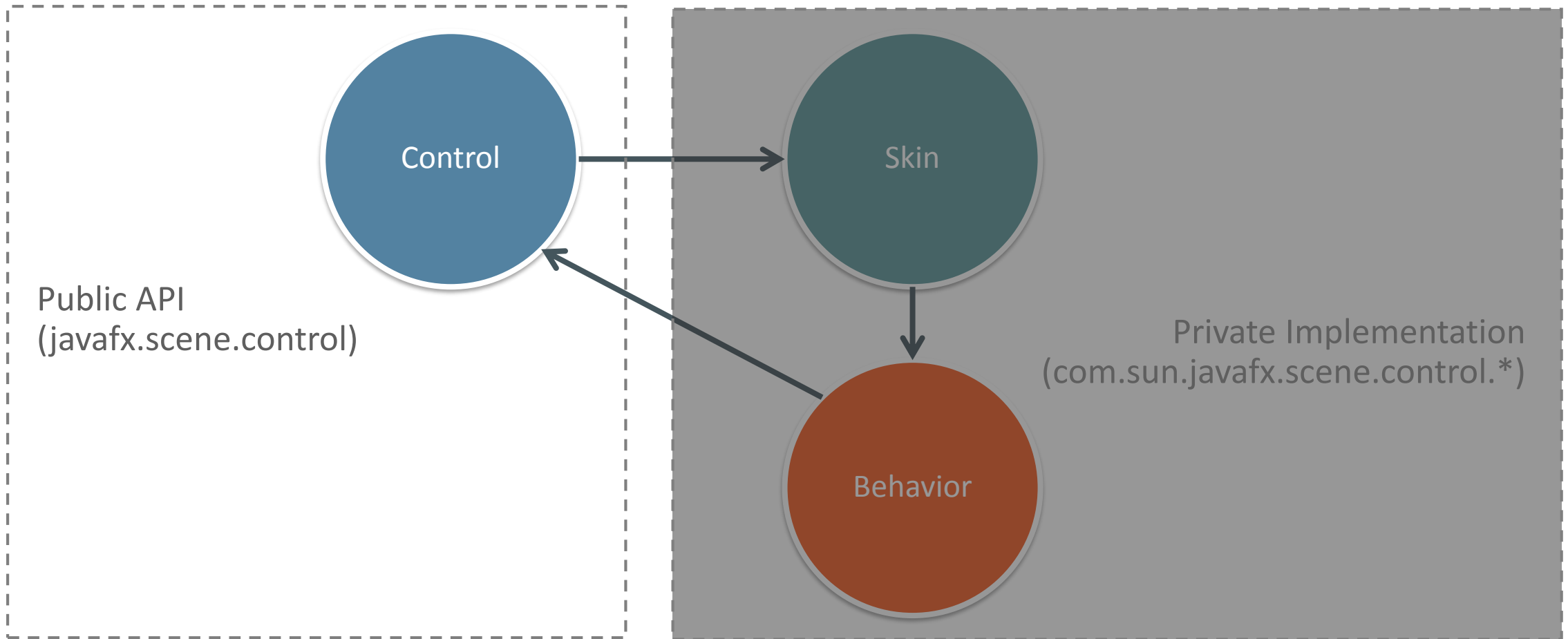
# JDK 9 :: What are we doing :: Progress Report

- [JDK-8077918](#): Review and make public relevant CSS APIs
  - Core CSS APIs have been made public, including
    - CSS data model
    - Converters (for converting from CSS text into Java)
    - Parsers
- Overall: Essentially this subproject is complete. Code is now in public JDK 9 repos for review.

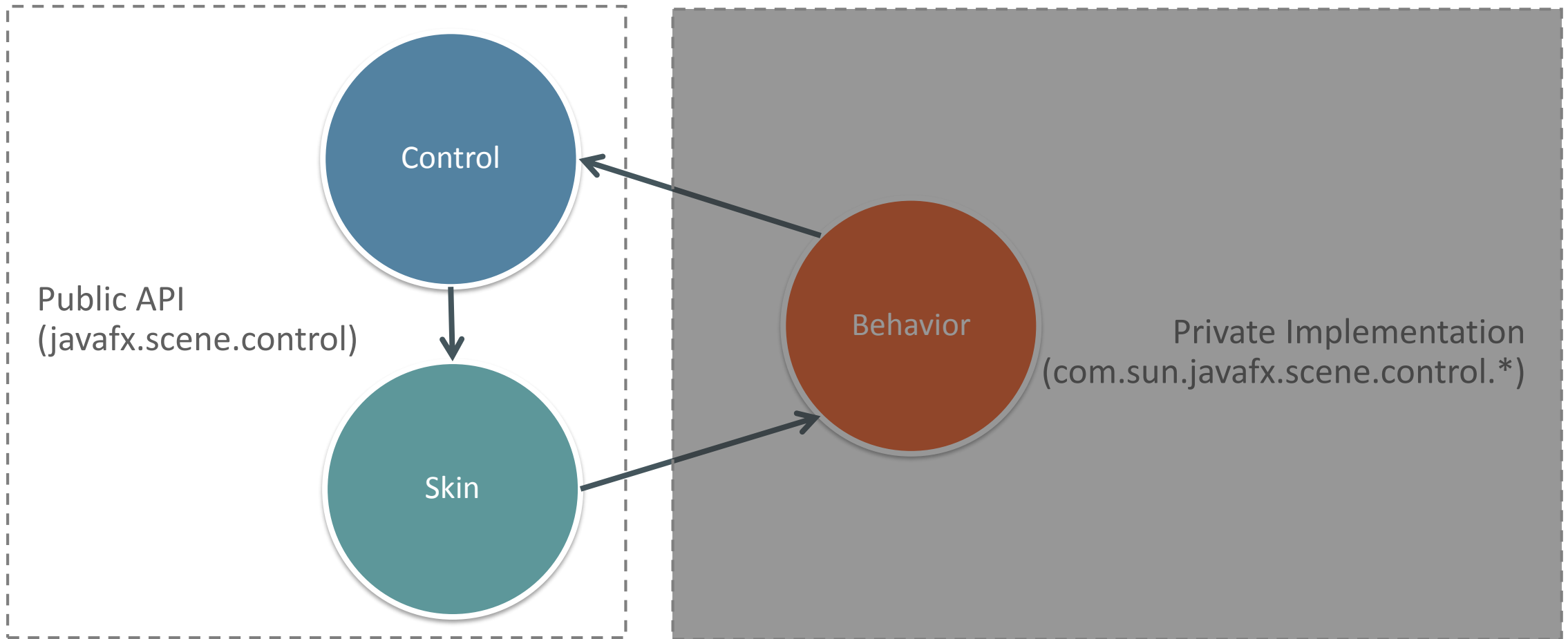
# JDK 9 :: What are we doing :: Progress Report

- Overall, progress is going very well:
  - All three subprojects have been out for public review
  - Subproject 2 has been removed from the JEP to allow for more ‘bake time’
  - The JEP 253 sandbox repo has been merged into the main repo – now in public builds
- It’s up to the community to test and give feedback on APIs as soon as possible!

# JEP 253 :: Summary



# JEP 253 :: Summary



# JEP 253 :: Summary

- Free side-effect of JEP 253:
  - Move closer to providing a full API for third-party UI controls
  - This has been a feature we've been wanting for a very long time
  - We now have all controls and skins available as public API
- Next target is to make behaviors public API too
  - Outside scope of JDK 9



# JEP 253 :: Summary

```
CssMetaData  
Styleable  
StyleableProperty  
SimpleStyleable*Property  
PseudoClass  
ParsedValue  
StyleOrigin
```

javafx.css

```
*Converter  
CssParser  
CSS data model:  
    CalculatedValue  
    CascadingStyle  
    Combinator  
    CssError  
    Declaration  
    Rule  
    *Selector*  
    Size  
    Style  
    StyleCache  
    StyleClass  
    StyleMap  
    Stylesheet  
StyleManager  
com.sun.javafx.css
```

# JEP 253 :: Summary

```
CssMetaData
Styleable
StyleableProperty
SimpleStyleable*Property
PseudoClass
ParsedValue
StyleOrigin
CssParser
CSS data model:
  CssError
  Declaration
  Rule
  *Selector*
  Size
  Style
  StyleClass
  Stylesheet
  javafx.css
```

```
*Converter
```

```
javafx.css.converter
```

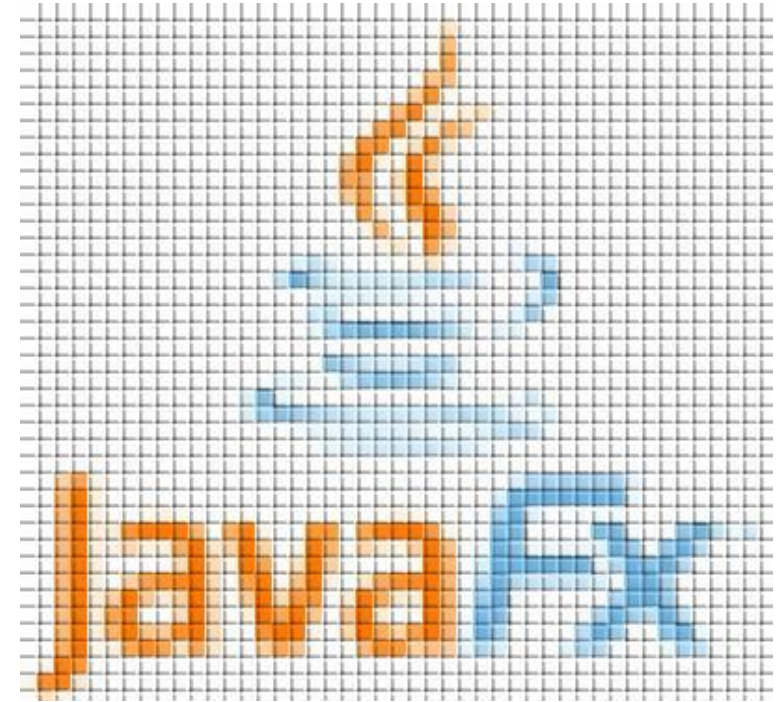
```
CSS data model:
  CalculatedValue
  CascadingStyle
  Combinator
  StyleCache
  StyleMap
  StyleManager
```

```
com.sun.javafx.css
```



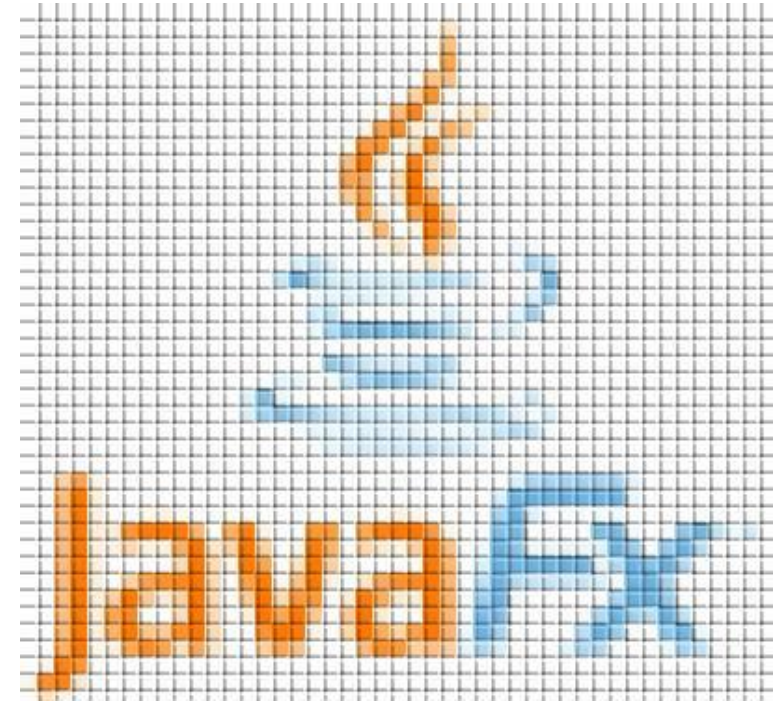
# Other JDK 9 Enhancements

# High DPI



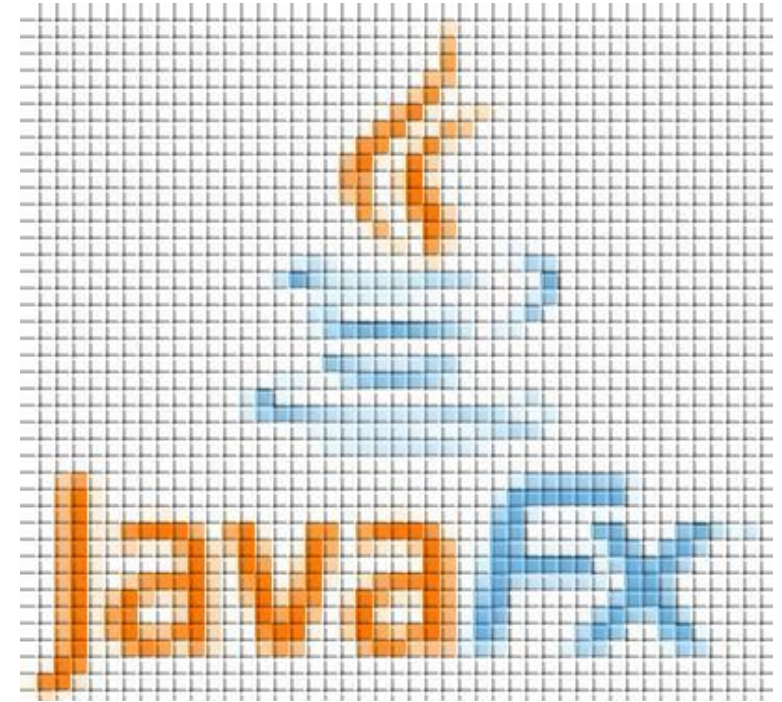
# High DPI

- Today we have the following support for High-DPI:
  - Mac: for retina display, integer scales (200%)
  - Windows: for High-DPI settings  $\geq 150\%$
  - No Linux support
  - No API to allow application control over High-DPI scaling
    - Rendering is always done with an integer scale
    - On Windows, the blit to the screen might use a non-integer scale



# High DPI

- In JDK 9 we will add:
  - High-DPI support for Linux
  - API to set the render scale and the threshold for enabling High-DPI
  - Support for “snap to pixel” even when using non-integer render scale



# New Public APIs

# Overview

- As mentioned, JDK 9 modules means `com.sun.*` private API disappears
- We know that people often use private APIs because public API is unavailable
- Two projects have been ongoing:
  - JEP 253: UI control skins and CSS APIs
  - Other, smaller APIs are being reviewed as candidates for making public
- The following is the result of our analysis on the other, smaller APIs

# Overview

- Used JDeps submissions from community to understand what functionality was being used.
- 20 different projects were analysed (commercial and open source)

# Overview

- Results quite clearly indicated main areas for further analysis:
  - UI Control Skins
  - CSS APIs
  - Toolkit / Platform APIs: firing pulse, nested event loops, pulse listening
  - Robot
  - Performance Tracker
  - Static utility functions



# Overview

- Results also showed areas where developers were doing the wrong thing
- E.g.

Bad API	Proper API
<code>com.sun.javafx.collections.ObservableListWrapper</code>	<code>javafx.collections.FXCollections</code>
<code>com.sun.javafx.css.StyleConverterImpl</code>	<code>javafx.css.StyleConverter</code>
<code>com.sun.javafx.css.PseudoClassState</code>	<code>javafx.css.PseudoClass</code>

# Nested Event Loop

- Sometimes an application wants to process events without returning from the current flow of control
- JavaFX internally uses nested event loops in some cases:
  - Calling `showAndWait` on Stage or Dialog
  - For displaying printer dialogs
- New API on Platform:
  - `public static Object enterNestedEventLoop(Object key);`
  - `public static void exitNestedEventLoop(Object key, Object rval);`
  - `public static boolean isNestedLoopRunning();`

# Pulse Listener

- Some applications want a callback during the pulse for each frame
  - Using `AnimationTimer` provides a similar capability, but runs before CSS and layout (and forces a continuous pulse)
- New API on Scene:
  - `public final void addScenePulseListener(Runnable r);`
- This will add a listener (`Runnable`) that is called every frame
  - Called after CSS and layout have been done
  - Called before rendering
  - Any changes to the scene graph will be rendered this frame, but will not have CSS or layout applied until next frame









# Platform Startup

- The JavaFX runtime is initialized in one of the following ways:
  - For standard applications that extend `javafx.application.Application`:
    - Running `'java MainClass'` where `MainClass` is a sub-class of `Application`
    - Calling `Application.launch` to launch the `Application` sub-class
  - For Swing application that use `JFXPanel`:
    - The first time an instance of `JFXPanel` is constructed
  - For SWT applications that use `FXCanvas`:
    - The first time an instance of `FXCanvas` is constructed
- Applications that don't fit one of these patterns often resort to calling `PlatformImpl.startup` which will no longer be accessible

# Platform Startup

- New API on Platform:
  - `public static void startup(Runnable);`
- Starts the JavaFX runtime and then calls the run method of the Runnable on the JavaFX Application Thread
  - The startup method returns before the Runnable is run
- Must not be called if the JavaFX runtime has already been started
  - Cannot be used to restart the JavaFX runtime after it has terminated

# Summary

- Results quite clearly indicated main areas for further analysis:
  - UI Control Skins  (JEP 253 – merged into repo)
  - CSS APIs  (JEP 253 – merged into repo)
  - Toolkit / Platform APIs
    - firing pulse  (No strong use case outside of testing)
    - nested event loops  (Added to Platform – merged into repo)
    - pulse listening  (Add to Scene – merged into repo)
  - Robot  (Too much work, primary use case is testing)
  - Performance Tracker  (Too much work)
  - Static utility functions  (Still investigating which methods need to be made public)

# Other JDK 9 Enhancements

- JEP 257: Update to Newer Version of GStreamer
  - Requires newer version of GLIB so some older Linux distros will no longer work
- Updated Version of WebKit
  - We will do this at least one time in JDK 9, and maybe twice
  - Goal: pick up bug fixes and performance improvements in a more timely fashion



# Future Investigations



# Future Investigations



- JDK 9 is still a long way off!
  - Our real focus for now is finishing the feature work discussed previously
  - Also: bug fixing!
- Looking further ahead, once JDK 9 ships, there are various things we might explore:
  - Provide a JavaFX equivalent for JEP 272 / AWT ‘Desktop’ API
  - Make UI Control Behaviors public
  - UI Control Actions API
  - Provide a public Focus Traversal API
  - JavaFX support for multi-resolution images



# Important JavaOne Sessions & BOFs

From the Oracle Java Client Team

# Important JavaOne Sessions



Title	When	Where
The New JavaFX Accessibility API	Today @ 12:30	Hilton—Yosemite A/B/C
HiDPI in Java and JavaFX	Today @ 4:00	Hilton—Yosemite A/B/C
JavaFX Layout: Everything You Wanted to Know	Tuesday @ 12:30	Hilton—Yosemite A/B/C
Migrating Java UI Client Apps to the Modular JDK	Tuesday @ 2:30	Hilton—Yosemite A/B/C
Packaging Java Applications	Thursday @ 10:30	Hilton—Yosemite A/B/C

# Important JavaOne BOFs



Title	When	Where
Meet the Oracle Java and JavaFX Client Team	Today @ 7:00	Hilton—Yosemite A/B/C
Introduction to the JavaFX Scenic View Tool	Today @ 8:00	Hilton—Yosemite A/B/C
Tips, Tricks, and Hidden Knowledge for Java Packagers	Tuesday @ 7:00	Hilton—Yosemite A/B/C

Thanks – Any Questions?

ORACLE®