

ORACLE®



JavaOne™

ORACLE®

JavaFX: New & Noteworthy

Kevin Rushforth & Jonathan Giles
Java Client Group
September 19, 2016

Java
Your
Next
(Cloud)

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.

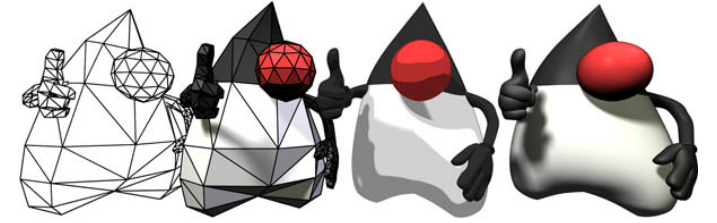
Agenda

- 1 ➤ JDK 8 Update Releases
- 2 ➤ Coming up in JDK 9
- 3 ➤ Looking beyond JDK 9 – update releases and JDK 10
- 4 ➤ Q & A



JDK 8 Update Releases

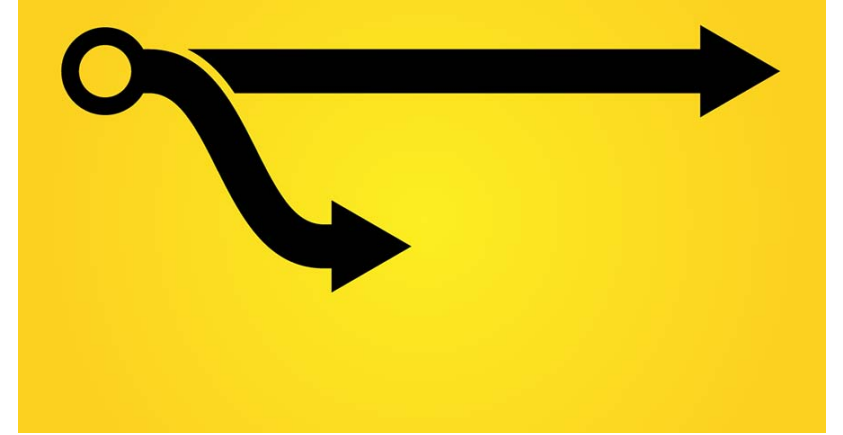
JDK 8 Update Releases



- Typically JDK update releases are not intended to be feature releases
- For JavaFX, we have shipped substantial improvements:
 - 8u20
 - 723 bugs fixed and 83 enhancements
 - 8u40
 - Accessibility, new controls (Spinner, Filtered Text, Dialogs), LCD text on Canvas, 3D user-defined normals.
 - 562 bugs fixed and 89 enhancements
 - 8u60
 - Upgraded to newer WebKit, and improved High-DPI support on Windows.
 - 257 bugs fixed and 29 enhancements

JDK 8 Maintenance Releases

- All development effort is on JDK 9
- No post-8u60 enhancements for JDK 8 line
- A *few* critical bug fixes can be backported to our quarterly releases
 - 8u102 released on 19 July 2016
 - 8u112 scheduled for 18 Oct 2016
 - 8u122 scheduled for 17 Jan 2017
 - Etc.



Coming up in JDK 9



Coming up in JavaFX in JDK 9

- Jigsaw modules
 - JavaFX is now modularized
 - Strong encapsulation
 - JEP 253: Public UI Control skins and more CSS APIs
 - New APIs for previously non-public features
- Other improvements:
 - High DPI: Linux support + API to query & control scaling
 - JEP 283: Enable GTK 3 on Linux
 - Updated GStreamer and WebKit
 - Smaller enhancements + Bug fixes

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)
 - This was (and is) a very large effort
 - The lack of non-Jigsaw-related ‘big-ticket’ features is a result
 - We did add several “smaller” enhancements (more later)

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 is one of:
 - Explicit module: a module that is defined by a module-info.java file
 - Automatic (implicit) module: a jar file on the module-path without a module-info.java
 - Unnamed module: classes on the classpath are in the “unnamed” module

Jigsaw Modularity: Properties of Modules

- An explicit module lists its inputs (requires) and outputs (exports) in its module-info.java file
 - Only exported packages are visible (more on this later)
 - It can only access packages from modules that it reads (requires)
- An automatic module exports all packages and reads all modules and the unnamed module
 - Useful for transitioning non-modular applications
- The unnamed module also exports all packages and reads all modules
 - An explicit module cannot “require” the unnamed module
 - Provides compatibility (no need to explicitly list the modules that you access)

Jigsaw Modularity: Properties of Modules

- The java launcher loads only the transitive closure of modules required by the application
 - All default modules loaded if your application main class is in the unnamed module
- javapackager can produce a bundled app with only needed modules

Jigsaw Modularity: JavaFX Modules

- JavaFX source code is organized as modules

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



Jigsaw Modularity: JavaFX Modules

- JavaFX classes and resources are linked into the JDK image
 - No more jfxrt.jar
- SWT interop is still delivered as a separate jar file:
 - Cannot be linked into runtime image because it depends on third-party swt.jar
 - It is an “automatic” module (meaning no module-info.class)
 - Renamed to javafx-swt.jar (formerly jfxswt.jar)
 - Automatic module name is derived as: javafx.swt



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]

JDK Modules

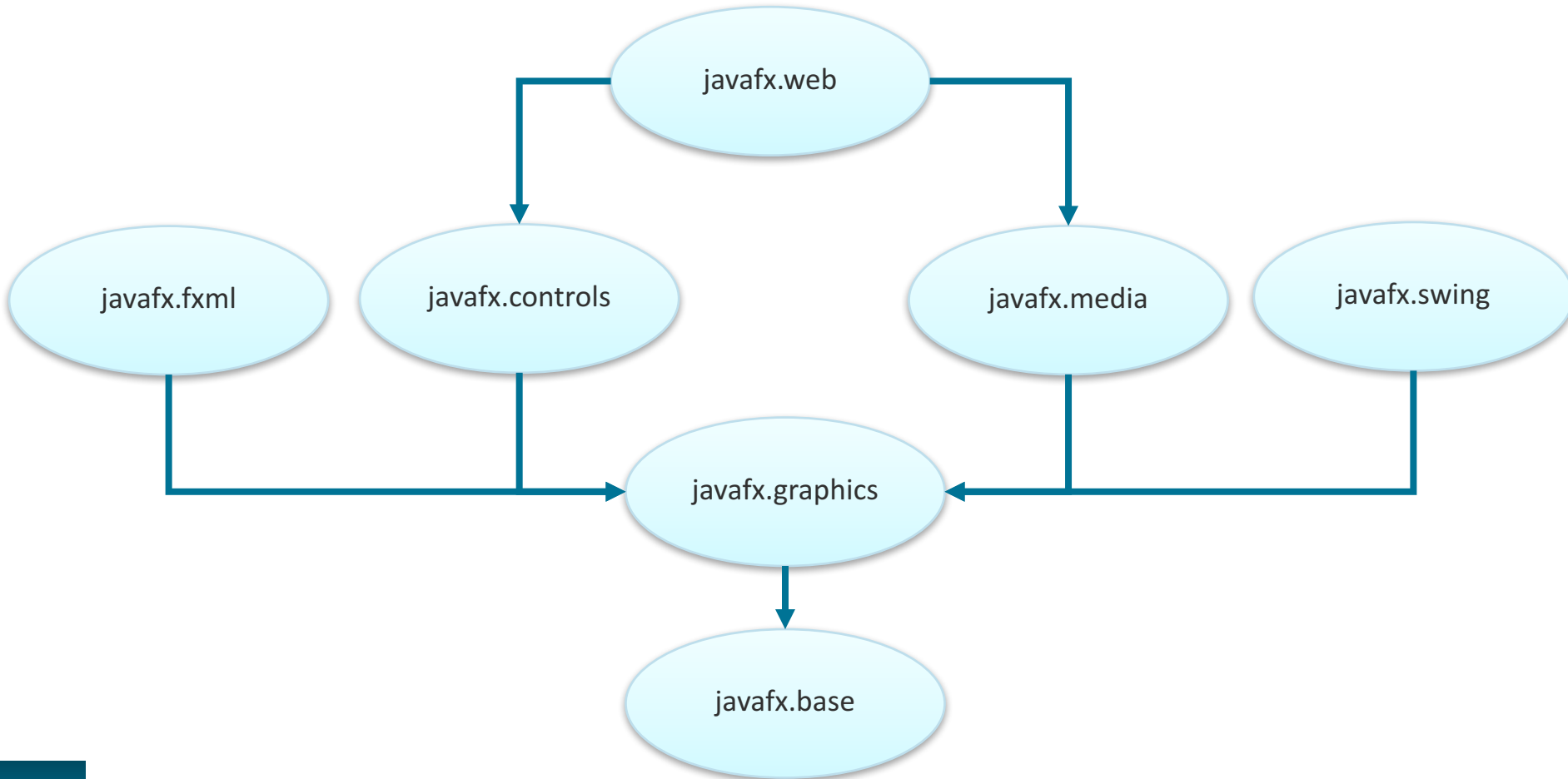
Public

- jdk.packager
- jdk.packager.services



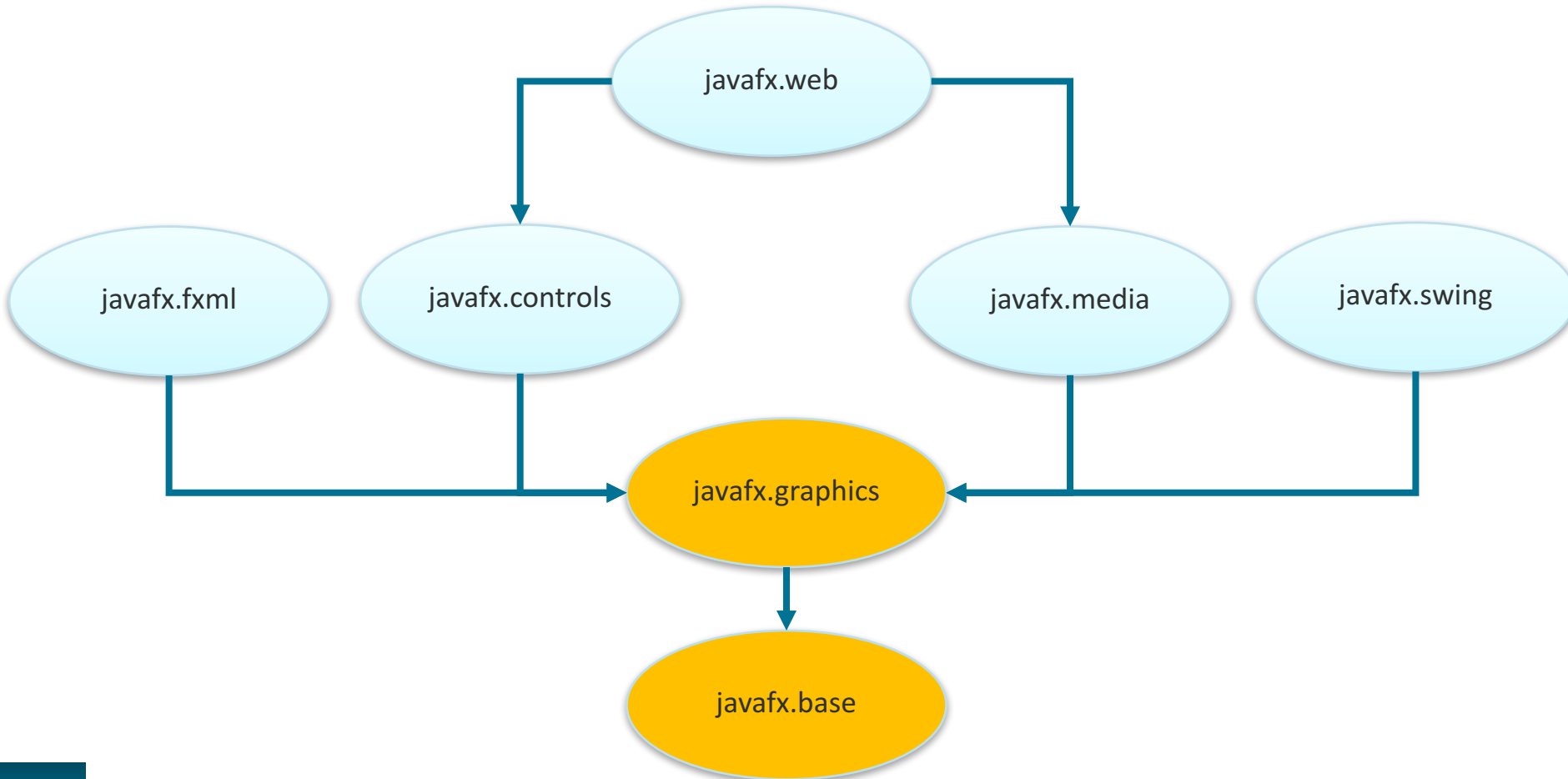
Jigsaw Modularity: JavaFX Modules

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



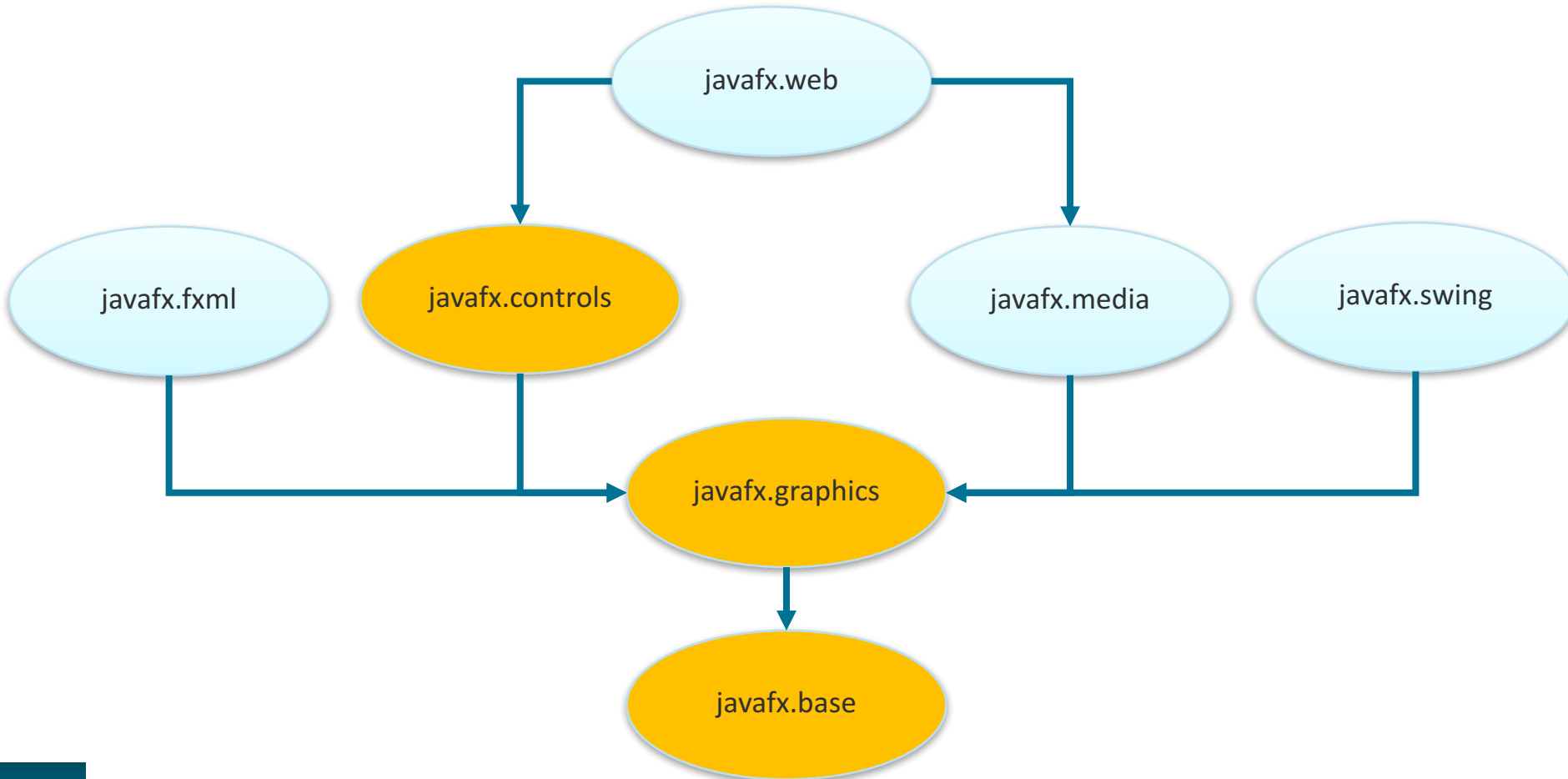
Jigsaw Modularity: JavaFX Modules

- 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 public types in publicly exported packages of all modules without changes
- Modular applications need to list their dependencies in module-info.java
 - Exported packages of required modules are accessible
 - Here is a minimal module-info.java for an application that uses the javafx.controls, javafx.graphics, and javafx.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 the main OpenJFX repo:
 - <http://hg.openjdk.java.net/openjfx/9-dev/rt>
- Stable early access JDK 9 builds are available on java.net:
 - <https://jdk9.java.net/download/>
 - These builds include the `javafx.*` modules
- Early access Jigsaw builds from jake sandbox also on java.net:
 - <https://jdk9.java.net/jigsaw/>
 - This includes “work in progress” proposed changes for `setAccessible`, etc.
 - These builds also include the `javafx.*` modules
 - **Please download this and test your application!**

Strong Encapsulation



Encapsulation of Modules

- Only explicitly exported packages are visible
 - Provides strict encapsulation of implementation details
 - Internal “API” is no longer accessible
 - Accessing any type in a non-exported package will result in an error
 - And no, you can’t simply use reflection to call `setAccessible`
 - This can be overridden with “`--add-exports`” command line switch in extreme need
- Within exported packages, only public types are accessible
 - Accessing any non-public type will result in an error
 - As with the previous case, you can’t simply use reflection to call `setAccessible`
 - This can be overridden with “`--add-exports-private`” command line switch in extreme need

Encapsulation of Modules

- Strong encapsulation means the supported-ness of an API is clear:
 - Each module that you access lists its publicly exported packages
 - A class or method you can access without breaking encapsulation (that is, without using `--add-exports` or `--add-exports-private`) is part of the API
 - No risk of “hidden” dependencies on internal methods / implementation details
- Non-modular apps that only use public API will run unmodified
- Some modifications may be needed when migrating app to modules

Encapsulation: Impact on Modular Applications

- Launching a JavaFX Application
 - The JavaFX launcher constructs an instance of your Application subclass
 - This means JavaFX must be able to access that class
 - You need to export the containing package to javafx.graphics (or publicly)
 - Example:

```
module my.app {  
    requires javafx.controls;  
  
    exports my.pkg to javafx.graphics;  
}
```

Encapsulation: Impact on Modular Applications

- Annotating non-public types with `@FXML`
 - FXML needs ability to access your private fields and methods
 - You need to “exports private” your package to `javafx.fxml`
 - Example:

```
module my.app {  
    requires javafx.controls;  
    requires javafx.fxml;  
  
    exports private my.pkg to javafx.fxml;  
}
```

Resource Encapsulation: Impact on Modular Applications

- Resources are encapsulated similarly to classes
- `Class.getResource()` will find resources in modules if package is accessible
 - No more “dipping into” the internals to load another module’s resource
- `ClassLoader.getResource()` will not find resources in module
 - A modular app cannot simply use `"/some/path/myresource.css"` as stylesheet URL

Resource Encapsulation: Impact on Modular Applications

- JavaFX has several APIs that take a URL (or url String)
 - CSS Stylesheets
 - FXMLLoader
 - Image & Media
 - WebEngine
- A non-modular app can continue to use ClassLoader-relative URLs
- Modular applications should use `Class.getResource()` instead

Encapsulation of impl_ “API”

- JavaFX has always had public methods that started with ‘impl_’, e.g.
 - Image.impl_getUrl() or ContextMenu.impl_showRelativeToWindow()
- These were hidden from API docs and marked @Deprecated to indicate that applications should not use them
 - Most of these were just implementation details
 - Some were there because we weren’t ready to commit to them as final API
- The module system hides non-public packages, so we took the opportunity to clean up our impl_* APIs in JavaFX
- All impl_* methods are gone, but some have become public API

Other compatibility issues:

- Node builder classes
 - Deprecated in JDK 8 (and not part of API docs) and slated for removal
 - Removed from JDK 9 (as of build 51)
- JMX support
 - We used to ship a standalone javafx-mx.jar file with the JDK (not JRE)
 - Was unsupported in JDK 8
 - Removed from JDK 9 (as of build 111)
 - It is still built as part of OpenJFX in case developers need it

Impact on JavaFX Applications: Summary

- All applications need to check and avoid:
 - Using any JavaFX `com.sun.*` package or `impl_` method
 - Many already have a public replacement
 - Using `setAccessible` to access non-public types (even in a `javafx.*` package)
 - Access to any internal JavaFX resources (e.g., `modena.css`)
 - Node builders
- If you want to modularize your application, you also need to avoid:
 - Using `ClassLoader.getResource` (use `Class.getResource` instead)
 - Passing “classpath-relative” URL strings for resources in your module into CSS, etc.
 - Don’t do this: `scene.getStyleSheets().add("/path/to/my/resource/resource.css");`

JEP 253

Prepare JavaFX UI Controls & CSS APIs for Modularization



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	MenuButton	RadioButton	Spinner	TabPane	ToggleButton	TreeTableColumn
CheckBoxTreeItem	CustomMenuItem	Label	MenuItem	RadioMenuItem	SplitMenuButton	TextArea	ToggleGroup	TreeTableView
								TreeView

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

- JavaFX now has a full complement of expected UI controls
 - “Wide but not deep”
- We need to iterate:
 - fill gaps in functionality,
 - fixing bugs,
 - improving support for third party UI controls

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

- Some releases are motivated by changes beyond UI controls, but require significant changes to controls:
 - JavaFX 2.0:
 - Moving to Java
 - Introduction of 'Region'
 - JavaFX 8.0:
 - Migrating to Lambdas
 - JavaFX 8u40:
 - Introduction of accessibility support
- JDK 9: Modules

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

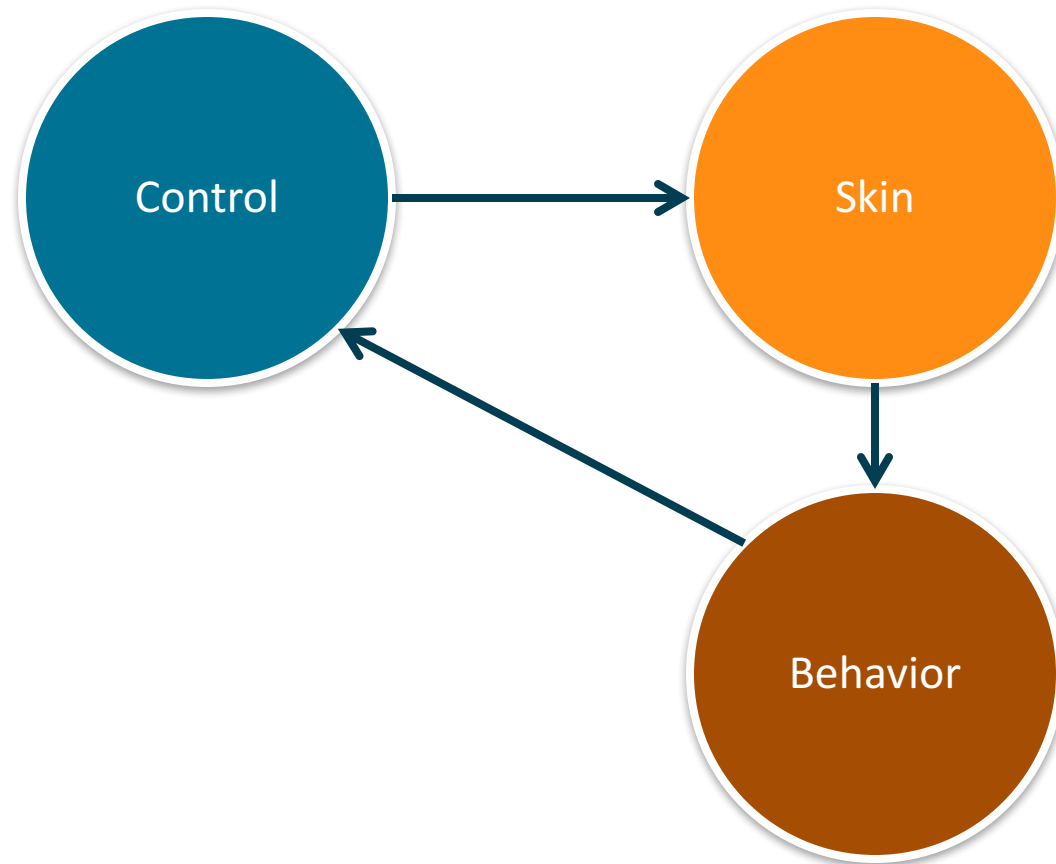
To understand JEP 253, we need to understand:

How are UI Controls built?

(Note: I am presenting a full session on this at 5:30pm today)

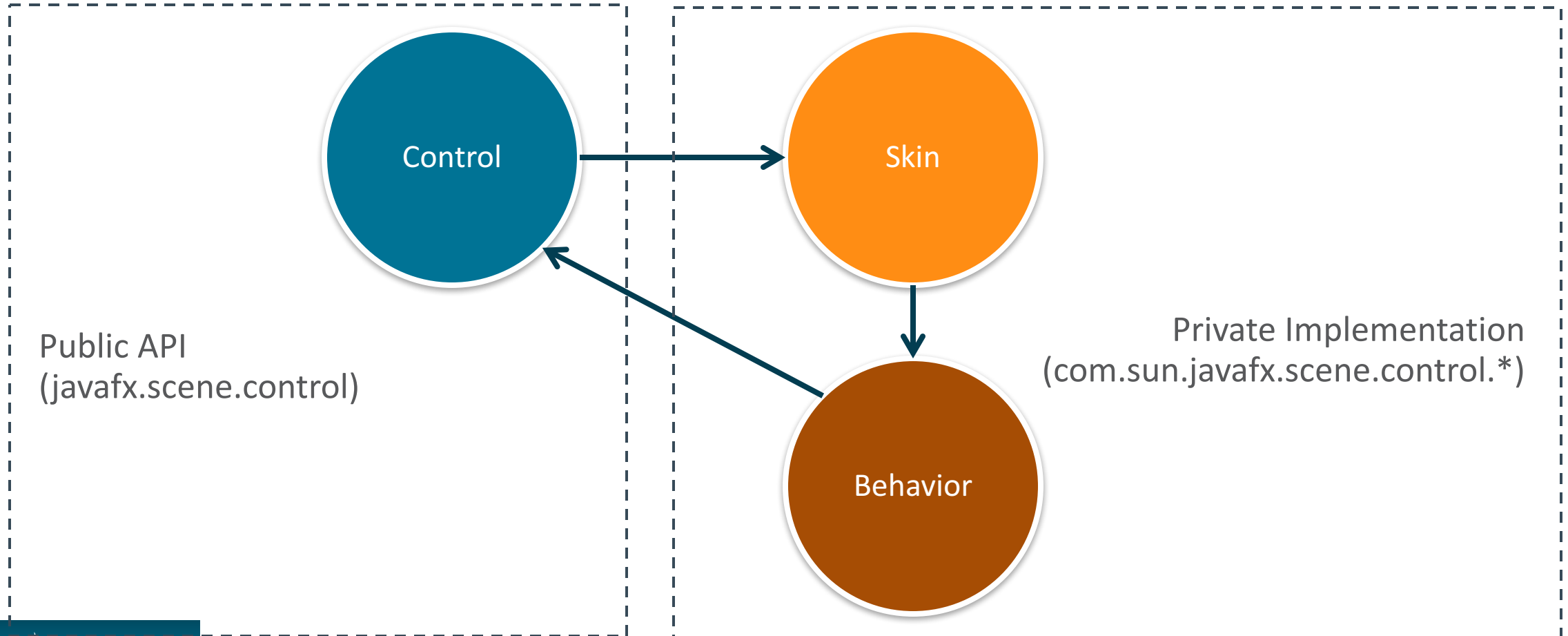
JDK 8 and Earlier :: State of the Nation :: UI Controls

- 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

We also need to ask the same question for CSS APIs:

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
```

```
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

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

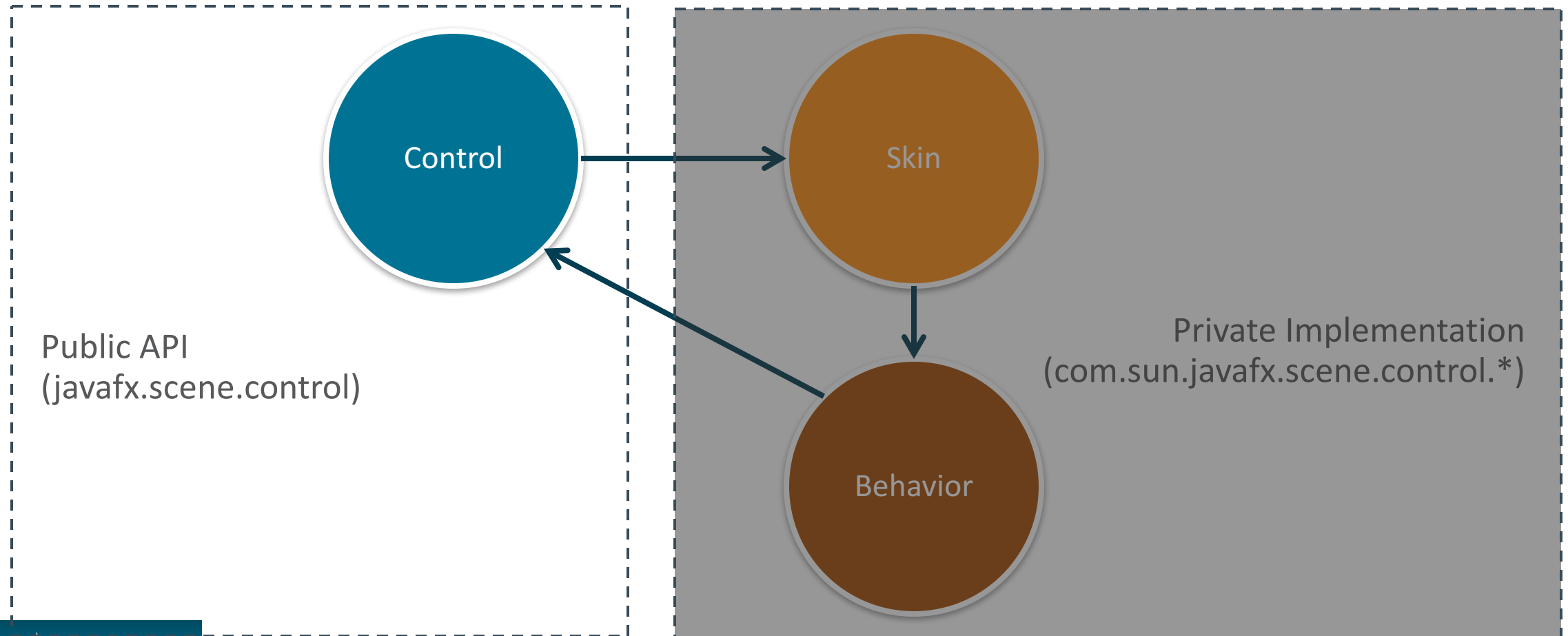
`javafx.css`

```
*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

- UI Controls and CSS APIs have always had public API and ‘private’ API
 - Many projects can’t resist the urge to use `com.sun.*` API.
 - Always frowned upon, but never prevented (and impossible to prevent anyway).
- JDK 9 with Jigsaw modularity is a big game changer:
 - Up until JDK 9, developers could use API in `com.sun.*` packages.
 - JDK 9 enforces boundaries - `com.sun.*` becomes unavailable
- Some JavaFX apps and libraries will fail to compile / execute under JDK 9.

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

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

Motivation for JEP 253

- JDK 9 modules has forced our hand:
 - We need to make public more API
 - We can't do everything
 - We need to identify the most important private API
- Current approach is split into two projects:
 - JEP 253: UI Controls and CSS APIs
 - 'Smaller' JavaFX APIs (discussed later)

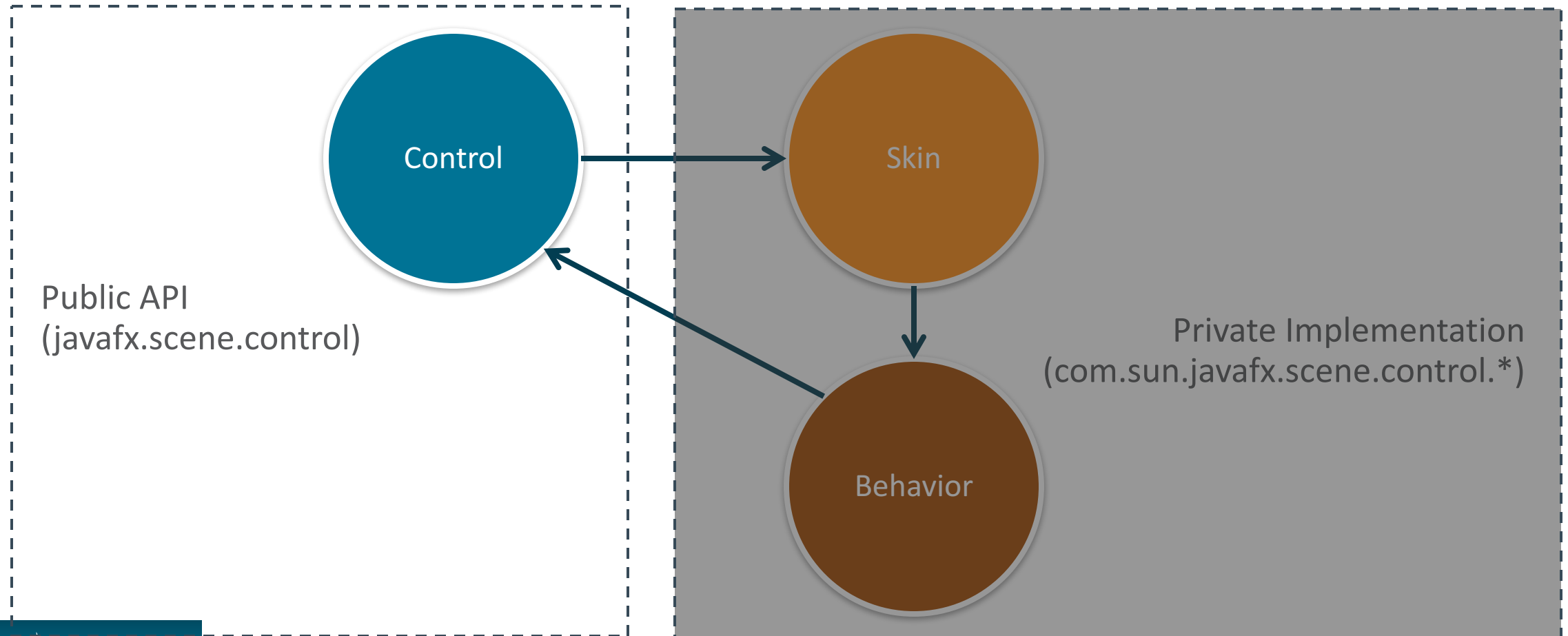
JDK 9 :: What are we doing

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

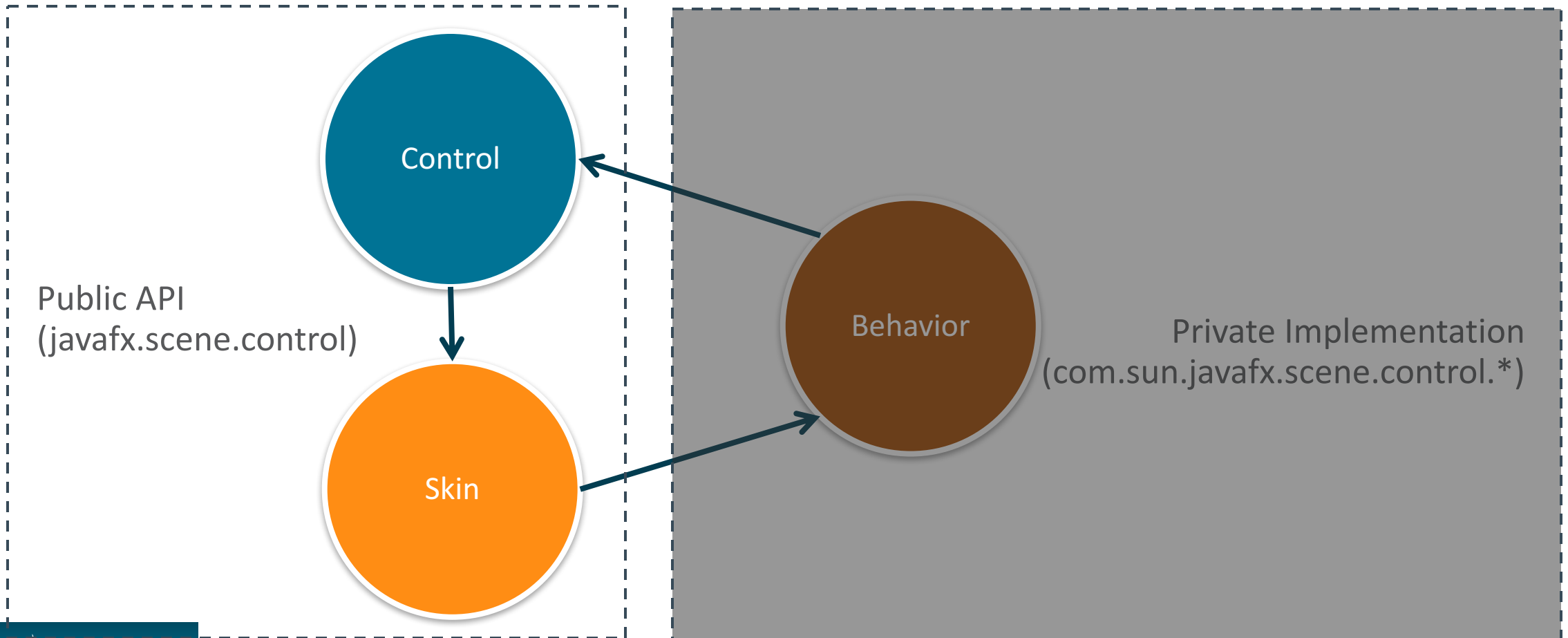
JDK 9 :: What are we doing :: Progress Report

- [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.

JEP 253 :: Public Skins :: Summary



JEP 253 :: Public Skins :: Summary



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.

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
```

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

- Overall, JEP 253 is 'finished' (barring internal testing):
 - All code has been merged into JDK 9 main repo for over a year
- It's critical that the community test and give feedback on JDK 9 as soon as possible!

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
 - Will discuss later in this presentation...



Other JDK 9 Enhancements

New Public APIs

Understanding which private APIs are in use

- In June 2015, we reached out to the community for help [1].
- We needed to understand what private APIs were most commonly being used.
- JDeps is a tool that reports dependencies in code.
- We wrote a small app that would analyse output from ~two dozen projects.
 - If you're interested, run the following command:
`jdeps -v <jarfile> > <textfile>`
e.g.: `jdeps -v SceneBuilder.jar > SceneBuilder.txt`
- In summary: some open source apps and some customer apps break!

[1] <http://mail.openjdk.java.net/pipermail/openjfx-dev/2015-June/017340.html>

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, platform startup
 - Robot
 - Performance Tracker

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>

Firing Pulse

- Sometimes an application wants to be sure another pulse will be run.
- New API in Platform:
 - `public static void requestNextPulse();`
- If no pulse is running, this will run a pulse at some point in the future, even if no pulse is required.
- If a pulse is running, another one will be scheduled once it completes.

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 in 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 in Scene:
 - `public final void addPreLayoutPulseListener(Runnable r);`
 - `public final void addPostLayoutPulseListener(Runnable r);`
- This adds a listener (Runnable) that is called every frame
 - Called after CSS and either before (pre) or after (post) layout
 - Called before rendering
 - Changes to scene graph are rendered this frame, but CSS is not 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 is no longer 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

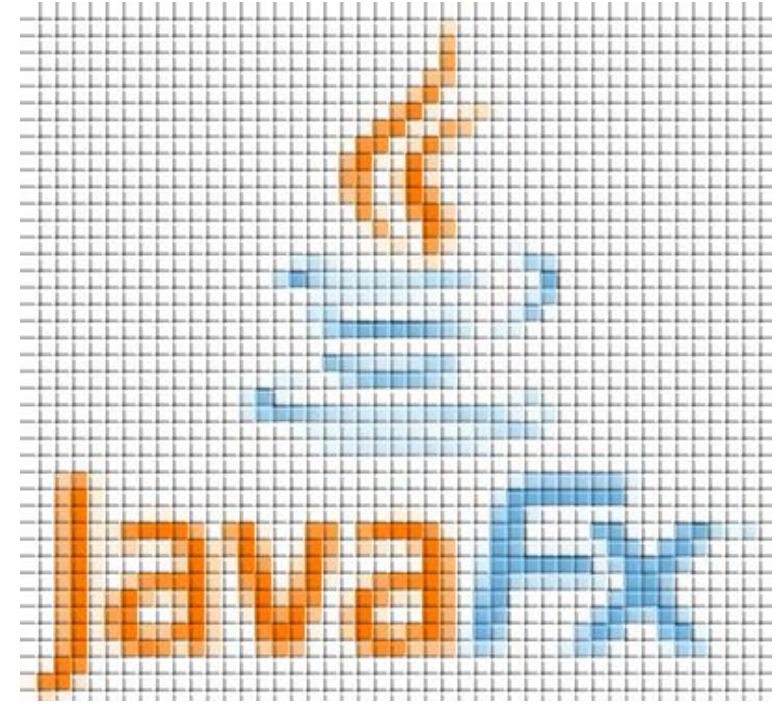
Summary

- Results quite clearly indicated main areas for further analysis:
 - UI Control Skins  (JEP 253)
 - CSS APIs  (JEP 253)
 - Toolkit / Platform APIs
 - firing pulse  (Added as `Platform.requestNextPulse()`)
 - nested event loops  (Added to Platform)
 - pulse listening  (Added to Scene)
 - Platform startup  (Added to Platform)
 - Robot  (Too much work, primary use case is testing)
 - Performance Tracker  (Too much work)

Turning impl_* methods into public API

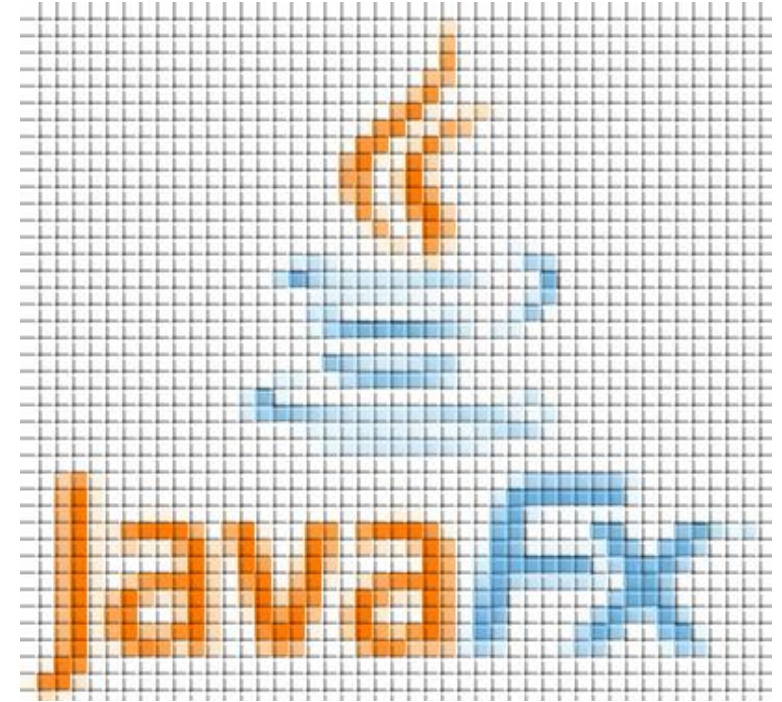
- All impl_* methods are gone...but a small number transitioned into full public API:
 - **FXMLLoader**: LoadListener interface + set / get methods
 - **Image**: getUrl()
 - **KeyCode**: getChar(), getCode()
 - **GridPane**: getRowCount(), getColumnCount(), getCellBounds(col, row)
 - **Text, TextFlow**: selection, caret, hit test APIs
 - **Window**: getWindows()
 - **FXCanvas**: getFXCanvas()
 - **TableColumnBase**: reorderable property

High DPI



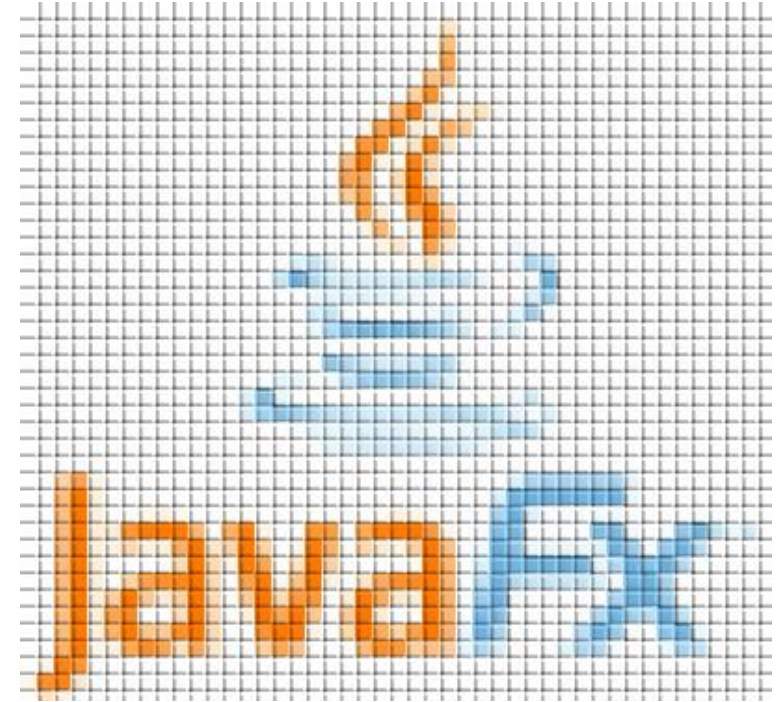
High DPI

- JDK 8u60 has 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 added:
 - High-DPI support for Linux
 - High-DPI threshold changed to 125% on Windows
 - API to get the screen scale
 - API to force integer render scale (false by default)
 - API to set/get the render scale
 - Support for “snap to pixel” even when using non-integer render scale



JEP 283: Enable GTK 3 on Linux

- FX window-toolkit (glass) uses GTK 2 in JDK 8u60
- Ubuntu 16.04: GTK 2 no longer part of typical installation
- SWT 4.x uses GTK 3 by default
 - Cannot load GTK 2 and GTK 3 in same application
 - SWT is usually loaded before FX is loaded
- FX glass and AWT toolkit support both GTK 2 (default) and GTK 3
 - Auto-detect if GTK 3 is already loaded and switch to that
 - Fail-over to GTK 3 if GTK 2 is not present on system
 - Use `java -Djdk.gtk.version=3` (or 2) to force



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
 - Already updated once for JDK 9
 - Planning to update at least one more time...maybe twice
 - Goal: pick up bug fixes and performance improvements in a more timely fashion



Other JDK 9 Enhancements

- 102 smaller enhancements, such as:
 - **Tooltip**: Customizable timing
 - **ComboBox & Spinner**: `commitValue()`, `cancelEdit()`
 - **Spinner**: added `promptText` property
 - **Node**: `viewOrder` property (user-specified rendering order)
 - **Font**: `loadFonts()`
 - **Collections**: `viewIndex` property
 - **FXPermission**: fine-grained permissions
- 756 bug fixes!



Looking beyond JDK 9

What might happen in JDK 9 update releases and JDK 10





Future Investigations

- JDK 9 is nearly frozen!
 - Focus is shifting towards JDK 9 update releases and JDK 10 feature planning
- We are gathering feature ideas from the usual places (JBS, community feedback, internally, etc.)
- Disclaimer: We haven't actually started planning yet, so what we're discussing today might change!



Future Investigations

- Platform improvements:
 - JavaFX ‘Desktop’ API (no AWT)
- Scenegraph improvements:
 - Public focus traversal API
 - New layouts (e.g. Flexbox)
- Graphics improvements
 - Support for multi-resolution images
 - Image writers
 - Updated graphics renderers
 - Interop with platform graphics?
- Controls improvements:
 - Make UI Control behaviors public
 - UI Control Actions API
 - Extended accessibility (e.g. DatePicker)
 - TableView improvements (cell spanning, row/column freeze, commit on focus loss)
 - Draggable / detachable tabs in TabPane
- WebView improvements:
 - WebGL support
 - Accessibility

WHAT DO YOU WANT?



What do you want?

We have a notepad up here on the desk.

At the end of this session, take a moment –

let us know what you want to see in future releases of JavaFX



Useful JavaOne Sessions & BOFs

Useful JavaOne Sessions



Title	When	Where
JavaFX Layouts	Today @ 4:00pm	Hilton—Imperial Ballroom A
Building JavaFX UI Controls	Today @ 5:30pm	Hilton—Imperial Ballroom A
Meet the Oracle JavaFX and JDK Client Team	Today @ 7:00pm	Hilton—Imperial Ballroom A
JavaFX Scenic View	Today @ 8:00pm	Hilton—Imperial Ballroom A

Useful JavaOne Sessions



Title	When	Where
Project Jigsaw: Under The Hood	Tuesday @ 4:00pm	Hilton—Continental Ballroom 4
Project Jigsaw Hack Session	Wednesday @ 8:30am	Hilton—Continental Ballroom 4
The "Unsafe" Zone	Wednesday @ 10:00am	Hilton—Continental Ballroom 5
Pitfalls of Migrating Projects to JDK 9	Thursday @ 1:00pm	Hilton—Continental Ballroom 1/2/3

Q & A

Java Your Next (Cloud)

ORACLE®



JavaOne™

ORACLE®