The JavaFX Accessibility API

Jonathan Giles
Principle Member of Technical Staff
Java Client Group
October, 2015

JavaOne
ORACLE

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.

Program Agenda

1. Introduction / Overview
2. Fundamentals of Accessibility
3. Advanced Accessibility API
4. Making Your Code Accessible
5. Summary

What does it mean to be Accessible?

Accessibility is the degree to which a product, device, service, or environment is available to as many people as possible. Accessibility can be viewed as the "ability to access" and benefit from some system or entity. (*)

(*) http://en.wikipedia.org/wiki/Accessibility

Accessible Software

- Accessible software aims to address :
  - Visual Impairments (*)
  - Auditory Impairments
  - Motor/Mobility Issues
  - Seizures
  - Cognitive/Intellectual Impairments
- Assistive technologies are often built into the operating system
  - Screen readers (*), high contrast themes (*), zoom capability etc.

Screen Readers

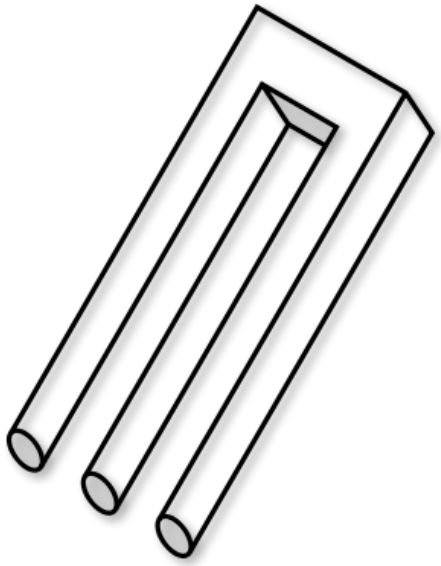
- Speak the contents of a control
- Provide their own concept of focus and traversal
  - Independent (but related to) normal focus and traversal
- Interact with programs using platform specific keys
- Examples:
  - VoiceOver, from Apple, for the Mac
  - Narrator, from Microsoft, for Windows
  - JAWS, from Freedom Scientific, for Windows

Accessibility and JavaFX

- Accessibility API came late to JavaFX
  - Focus of the team was on implementing the controls
  - There was a lot of engineering turn over on the team
  - Prototypes were started but none were completed
- Windows accessibility API was incomplete (until UIA)
  - Non-native solutions such as IA2 available (augments MSAA API)
  - Microsoft UIA only became available with Windows 7

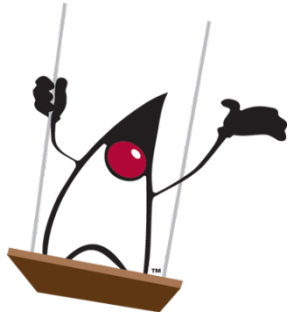
Why is Accessibility hard to implement?

- Requires a common portable JavaFX API
- Must support many different kinds of controls
  - Many different controls causes explosion of API and concepts
- Lots of different platform specific native code
  - Lack of documentation and native code examples
  - Need to determine behavior by experimentation
- Implementation cross cuts different layers in the JavaFX
  - Implementers needs to be familiar with Glass, Controls, Skins …



Other Accessibility Implementations

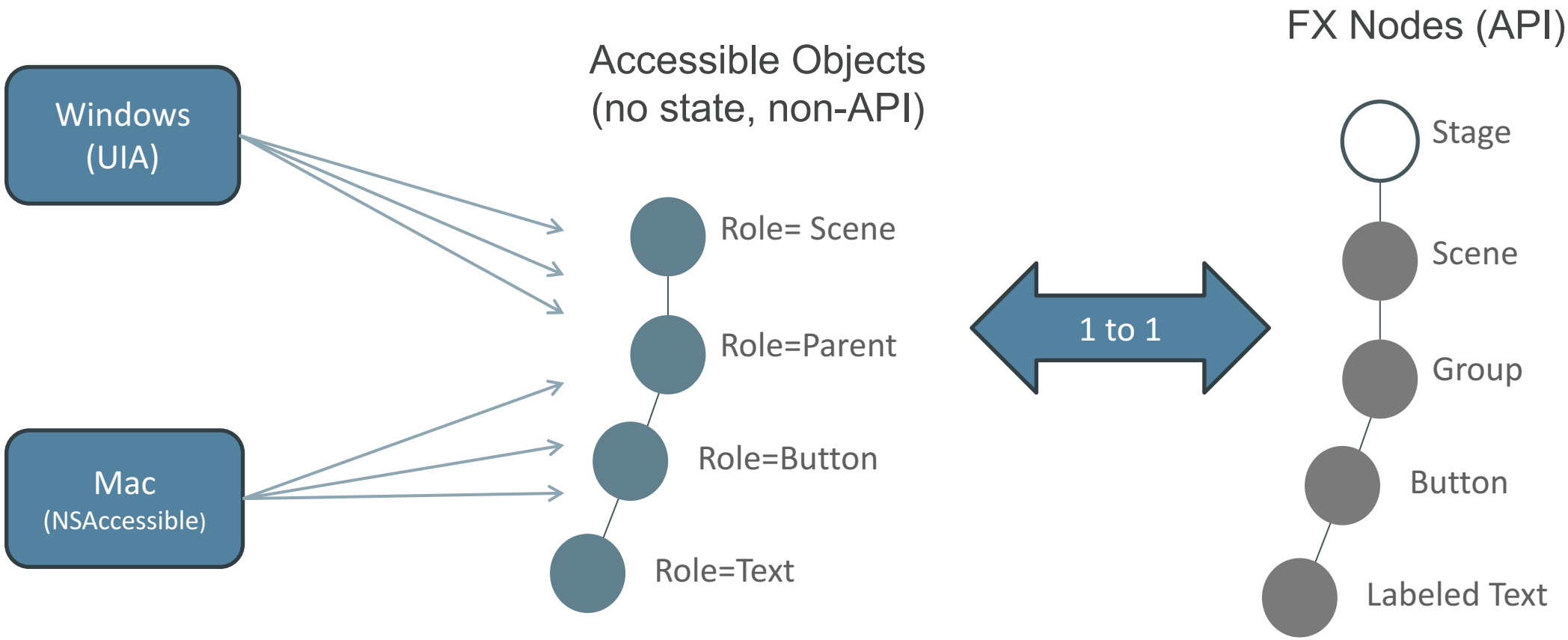
- AWT/Swing
  - Relies on access bridge shared library and vendor support
  - Incomplete, considered by some to be buggy
- Eclipse SWT
  - Uses native controls so gets native accessibility “for free”
  - Uses IA2 on Windows (not supported by Microsoft)
  - Some bugs and problems in the API and implementation



Design Goals for Java FX Accessibility

- Provide a complete, minimal and unobtrusive API
  - Support native behavior and platform specific mechanisms
  - Use existing Java FX conventions and standards
  - Ensure the implementation has a low overhead
- Implement accessibility for all built-in FX controls
  - Prove that the API is complete and full featured
- Allow developers to make their own controls accessible

JavaFX Accessibility: A Lightweight Implementation



Accessibility Design Goals: Realised!

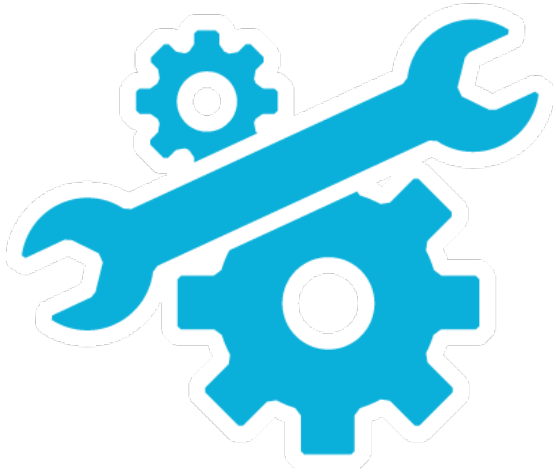
- JavaFX scenegraph is used to contain accessibility info
  - Avoid shadow hierarchies (decrease complexity, footprint)
- Use FX properties, methods and subclassing
  - Ensures that FX Accessibility is CSS and FXML friendly
- Low overhead implementation
  - Run code and create objects only when Screen Reader is active
  - No Screen Reader == No Accessible Code Executed
  - Screen Reader – No data duplication as accessibility data is represented by scenegraph

Fundamentals of Accessibility

The JavaFX Accessibility API

The Fundamental Accessibility API

A simple set of properties to solve the most common cases



- `Node#accessibleRoleProperty()`
- `Node#accessibleRoleDescriptionProperty()`
- `Node#accessibleTextProperty()`
- `Node#accessibleHelpProperty()`
- `Label#labelForProperty()` (not a new property)

The Accessible Role Property

- AccessibleRole is an enumeration property on Node
- The Role tells the Screen Reader the “kind of control”
  - Screen reader normally speaks the role (ie. says “Button”)
  - A Node can have at most one accessible role
- Some values the role might have:
  - `AccessibleRole.HYPERLINK` (used by javafx.scene.control.Hyperlink)
  - `AccessibleRole.TABLE_VIEW` (used by javafx.scene.control.TableView)

Setting the Role (Once, During Creation)

```
public Button(String text, Node graphic) {
        super(text, graphic);
        initialize();
}
private void initialize() {
        getStyleClass().setAll(DEFAULT_STYLE_CLASS);
        setAccessibleRole(AccessibleRole.BUTTON);
}
```

When should I set the role of a Node?

- The role always set appropriately for built-in controls
- The default role is `AccessibleRole.NODE` (or `PARENT`)
  - Currently both of these roles are ignored by the screen reader (nothing spoken)
- Set the role for Nodes that are not a Control
  - Choose the closest matching role from the set of enums
  - Very often `AccessibleRole.BUTTON` is a good choice

“But my control is not a button, it’s a bouncing ball!”

The Accessible Role Description Property

- Role Description is a string property on Node
- The property value is always null for built-in Controls
  - When null, the Screen Reader speaks native description
- Together, role and role description allow customisation
  - There are no user defined roles (`AccessibleRole` is an enum), but the role description allows the Screen Reader to say “Ball” instead of “Button”

“What about speaking the contents of my control?”

The Accessible Text Property

- Accessible Text is a Node Property, type is String (default is null)
- Tells the Screen Reader how to speak the contents of a control
- Always null by default for built-in Controls
  - Controls speak the appropriate contents for the control
  - A button will speak the button text and the role (ie. “OK”)
  - A text field speaks the text contents or the grayed out prompt text

The Accessible Help Property

- Accessible Help a Node property, type is String (default is null)
- Used to provide a longer more detailed description of a control
  - Always null by default for built-in controls
  - When null, the tooltip text is provided to the Screen Reader
- Extra help is provided when requested
  - Makes sense to default to the same value as the tool tip

The “Label For” Property

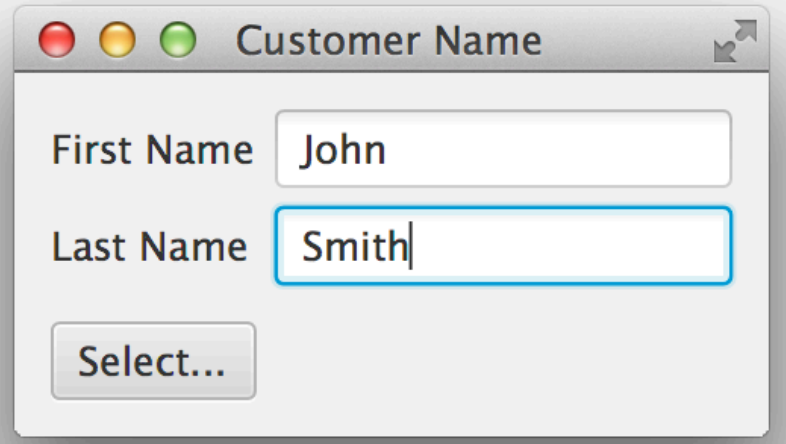
- “Label For” is a Label Property, type is Node (default is null)
- The labelFor property was previously used for mnemonics only
- Used to augment the description of TextFields and ComboBoxes
  - Often used for text fields that appear in configuration dialogs
- Used to provide description for other kinds of controls
  - Most common controls are ImageView, Slider, ProgressIndicator

A Text Field that uses “Label For”

```
TextField field = new TextField("Smith");
Label label = new Label("Last Name");
label.setLabelFor(field);
```

When focused, the screen reader says:

“Last name, editable text, Smith”

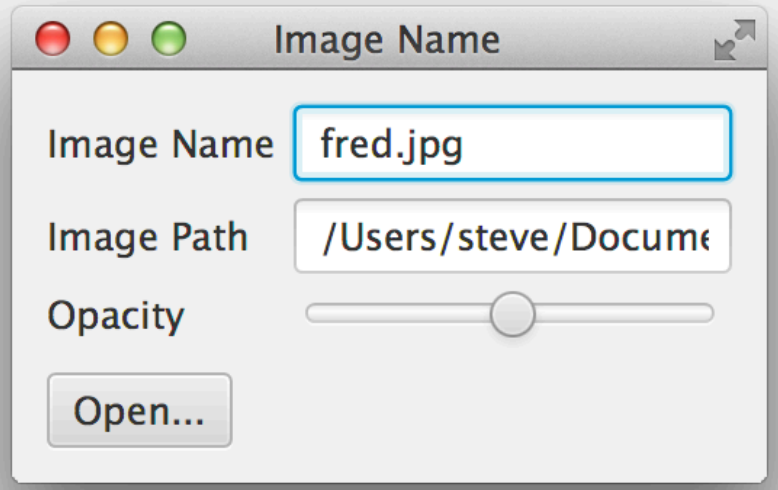


A Slider that uses “Label For”

```
Slider slider = new Slider(0, 100, 50);
Label label = new Label("Opacity");
label.setLabelFor(slider);
```

When focused, the screen reader says:

“Slider Opacity, 50%”



Advanced Accessibility API

The JavaFX Accessibility API

The Advanced Accessibility API

- Supports direct interaction with the Screen Reader
  - Return a value to the Screen Reader
  - Perform an action on behalf of the Screen Reader
  - Notify the Screen Reader that a value has changed
- Exposes the “Accessibility running” state
  - Determine whether accessibility is active

Accessible Methods: Called or Overridden

- `@override Object Node#queryAccessibleAttribute(AccessibleAttribute, Object...)`
- `@override void Node#executeAccessibleAction(AccessibleAction, Object...)`
- `void Node#notifyAccessibleAttributeChanged(AccessibleAttribute)`

Accessible Enums: Arguments to the Methods

- `AccessibleRole` (already seen)
  - ***`BUTTON, CHECK_BOX, LIST_VIEW,`*** `...`
- `AccessibleAttribute`
  - ***`PARENT, SELECTED, TEXT, ITEM_AT_INDEX`***`, ...`
- `AccessibleAction`
  - ***`FIRE, EXPAND, COLLAPSE, SET_TEXT`***`, ...`

Return a Value to the Screen Reader

- The Screen Reader requests the current state of a control
  - Different state is requested depending on the role of the control
- Example:
  - Return the number of rows in a list control
  - Return the selected items in a list control
- Some attributes the that might be queried:
  - `AccessibleAttribute.ROW_COUNT`
  - `AccessibleAttribute.SELECTED_ITEMS`

Overriding queryAccessibleAttribute()

```
@Override public Object queryAccessibleAttribute(
    AccessibleAttribute attribute, Object... parameters) {
      switch (attribute) {
        case ROW_COUNT: return getRowCount();
        default:
           super.queryAccessibleAttribute(attribute, parameters);
      }
}
```

IMPORTANT: Always call the super!

The Super: Node#queryAccessibleAttribute()

```
@Override public Object queryAccessibleAttribute(
    AccessibleAttribute attribute, Object... p) {
        switch (attribute) {
            case ROLE: return getAccessibleRole();
            case ROLE_DESCRIPTION: return getAccessibleRoleDescription();
            case TEXT: return getAccessibleText();
            case PARENT: return getParent();
            case BOUNDS: return localToScreen(getBoundsInLocal());
            case DISABLED: return isDisabled();
            case FOCUSED: return isFocused();
            default: return null;
        }
    }
}
```

Perform an Action on behalf of the Screen Reader

- The Screen Reader requests an action be performed
  - Actions are platform specific request (might be voice activated)
- Example:
  - Activate a button or traverse a hyper link
  - Expand a tree item or a title pane
- Some actions that might be requested:
  - `AccessibleAction.FIRE`
  - `AccessibleAction.EXPAND`

Overriding executeAccessibleAction()

```
@Override public void executeAccessibleAction(
    AccessibleAction action, Object... parameters) {
        switch (action) {
            case FIRE:
                fire();
                break;
            default: super.executeAccessibleAction(action, parameters);
        }
}
```

Notify the Screen Reader an Attribute has Changed

- The application changes the value of an attribute
  - The screen reader must be informed (it cannot know)
- Example:
  - Focus moves to a ListView, but the focus node is an item in the list
  - The user clicks on the (+) expansion indicator of a tree item
- Some attributes that the program changes:
  - `AccessibleAttribute.`***`FOCUS_NODE`***
  - `AccessibleAttribute.`***`EXPANDED`***
  - `AccessibleAttribute.`***`VALUE`***

Calling notifyAccessibleAttributeChanged()

```
public final DoubleProperty valueProperty() {
  if (value == null) {
    value = new SimpleDoubleProperty(this, "value", 0) {
      @Override protected void invalidated() {
        adjustValues();
        notifyAccessibleAttributeChanged(AccessibleAttribute.VALUE);
      }
    };
  }
  return value;
}
```

Determine whether Accessibility is Active

- Accessibility Active is a Platform property
  - type is Boolean (default is false, true when Screen Reader active)
- Used to implemented UI to better suit accessibility
- Examples:
  - Replace a chart or diagram with a table
  - Set additional nodes to be focus traversable

Making a Circle Traversable

```
Circle circle;
...
circle.focusTraversableProperty().bind(
        Platform.accessibilityActiveProperty());
...
```

Making Your Code Accessible

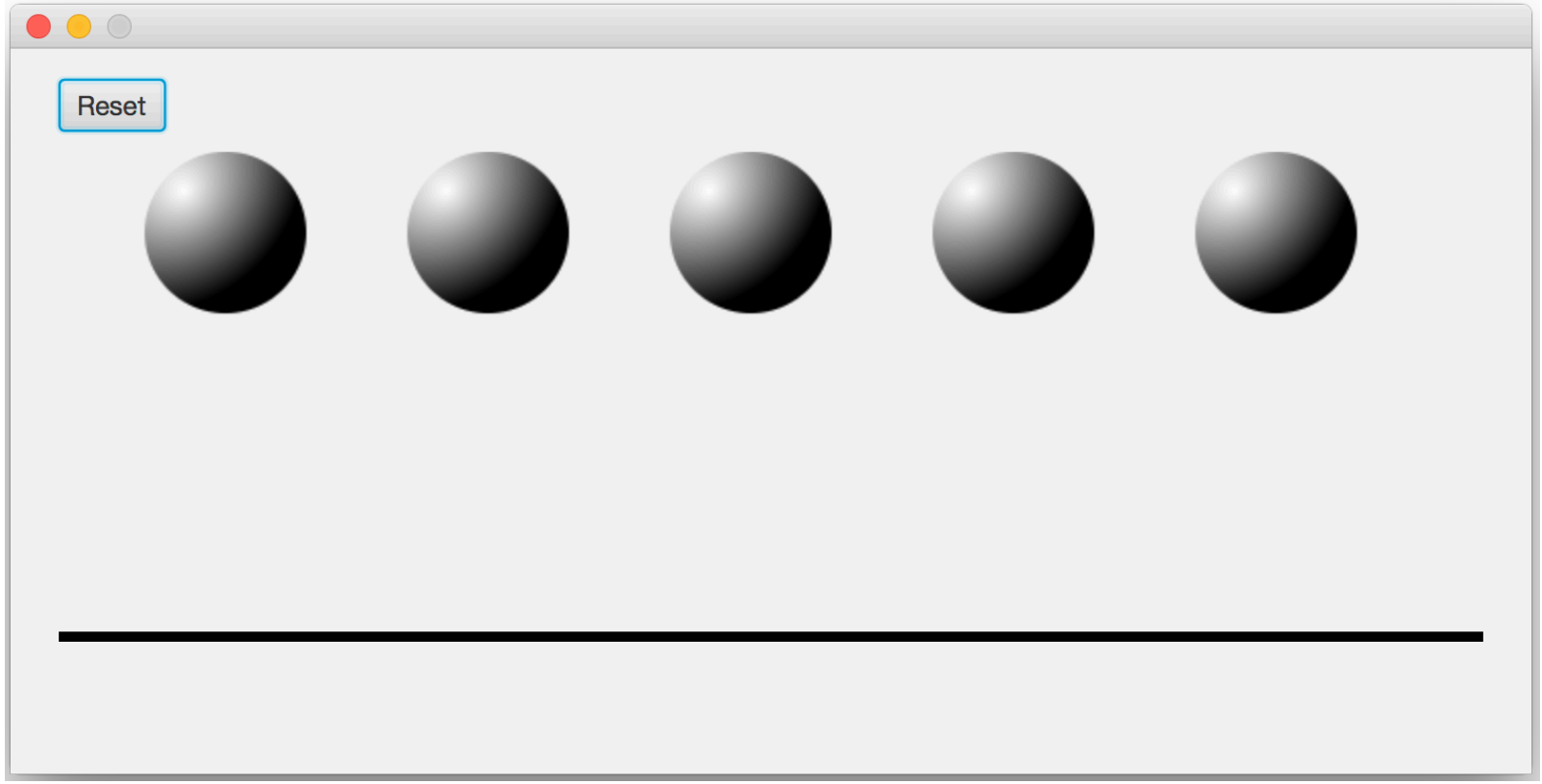
A Quick Accessibility Check List

- Use standard FX Controls as much as possible
  - Standard controls are highly configurable with CSS
  - Provide high contrast versions for custom CSS
- Provide text equivalent for every non-text element
  - Use accessibleText, helpText and/or tool tips
  - Examples: Provide text for ImageView, an animation that shakes etc.
- Use “label For” to connect text fields and combos to labels
- Ensure that important nodes are traversable from the keyboard

A Complete Example: Bouncing Balls

- A Ball is a Circle that draws a radial gradient
  - By default, its role is ***NODE*** so nothing is spoken
  - Need to tell the user that it is a ball and speak its state
  - Balls can be selected in order to stop and start bouncing
- Example Code
  - apps/samples/Ensemble8/src/samples/java/ensemble/samples/graphics2d/bouncingballs/BouncingBallsApp.java

Bouncing Balls



Bouncing Balls: The Unmodified Code

- Only the Reset button is read
- Nothing else is reachable, all other nodes are ignored

Demo: Nothing works

Using the Accessibility API Properties

```
…
ball.setAccessibleRole(AccessibleRole.BUTTON);
ball.setAccessibleRoleDescription("Bouncing Ball");
ball.setAccessibleText(text);
ball.setAccessibleHelp("This is bouncing ball, use
the primary action to start animation");
…
```

Demo: The first cut at accessible bouncing balls

Bouncing Balls: The “Free Native Traversal”

- The balls can be traversed using screen reader:
  - OS X: Using *Control + Option + Arrow*
  - Windows: Using *Capslock + Arrow*
- The Screen Reader will read each ball correctly
  - Ex. "First, Bouncing Ball”
- *Control+Option+Shift+N* (OS X) or *Capslock+F* (Windows) reads the accessible help

Using Accessibility API to Enable FX Traversal

```
...
focusTraversableProperty().bind(
    Platform.accessibilityActiveProperty());
...
```

Demo: Balls can be traversed using the tab key

Using the Accessibility API to Execute an Action

```
@Override public void executeAccessibleAction(
    AccessibleAction action, Object... parameters) {
        switch (action) {
            case FIRE: toggleAnimation(); break;
            default:
                super.executeAccessibleAction(action, parameters);
        }
}
```

Demo: Balls can be activated by Screen Reader

Bouncing Balls: The Final Code

- All text content and functionality is available to the Screen Reader
- Extra FX traversal is enabled only when Accessibility is active
- *Control+Option+Space* (OS X) or *Capslock+Space* (Windows) is used to start the animation for a ball

Summary

Which Accessibility API is Appropriate?

- The fundamental API makes simple nodes accessible
  - Choose a role, set the role description, set the text, etc.

- The advanced API is used for custom controls
  - Full featured: Used by FX to make built-in controls accessible

JavaFX Accessibility is Here

- Supports native accessibility mechanisms
  - No extra shared library or jar is required
- Implements accessibility for all built-in controls
- Provides a minimal but complete API
  - Provides a simple API for the most common cases
  - Provides an advanced API for custom controls

Thanks – Any Questions?

JavaOne ORACLE