ORACLE®

JavaOne™

ORACLE®

Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

HiDPI in Java and JavaFX

Adapting to new displays

Jim Graham
2D Graphics Guru
Java2D and JavaFX groups
October 26, 2015

JavaOne ORACLE

Safe Harbor Statement

Program Agenda

Program Agenda

Basic terms

- PPI
  - Concrete definition
  - Hardware term used for devices that have discrete pixels
- DPI
  - Hardware term usually associated with half-toning printers
  - Sometimes used as PPI synonym
  - Also API term
- Resolution
  - Sometimes used for PPI/DPI
  - Sometimes for raw pixel count

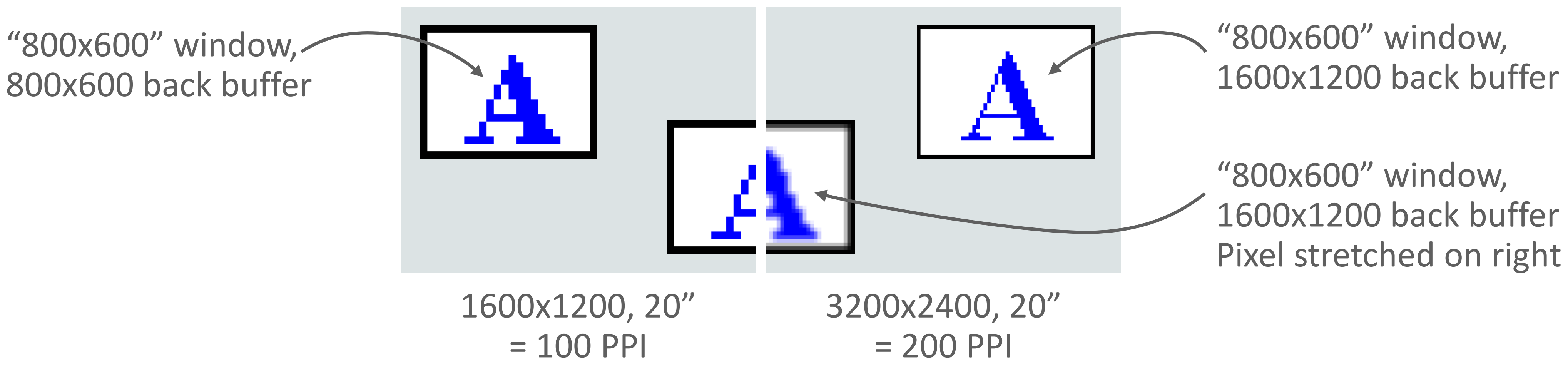
Some general approaches

- System scaling
  - Automatic for all applications
  - Can be coordinate scaling or pixel scaling
  - Least amount of work for applications
  - Pixel scaling will usually cause visible artifacts, but even coordinate scaling can have issue
- Application scaling
  - Requires buy-in from the application to various degrees
  - Usually accompanied by system information provided only in pixels
  - Application in complete control over artifacts
- CSS
  - Semi-automatic, but only for well-behaved apps that use virtual units

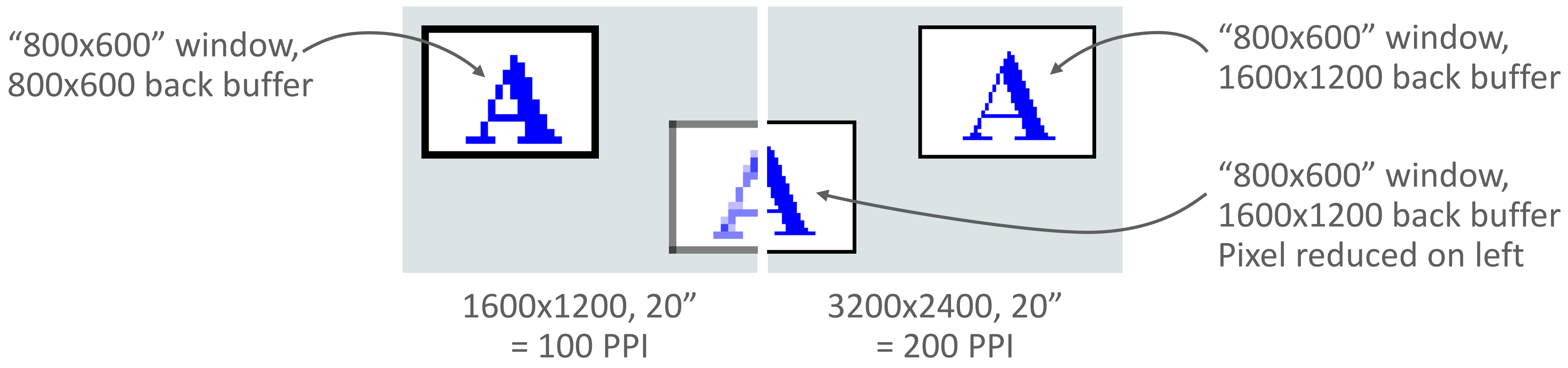
Screen layout approaches

- Screens in virtual space
  - Measurements on all screens similar in size
  - Windows usually remain consistent in size as they move between screens
  - Events (and most system APIs) should be floating point
  - Minority of APIs need pixel support – e.g. screen capture, system utilities

Virtual Screen Space Visualized (moving left to right)



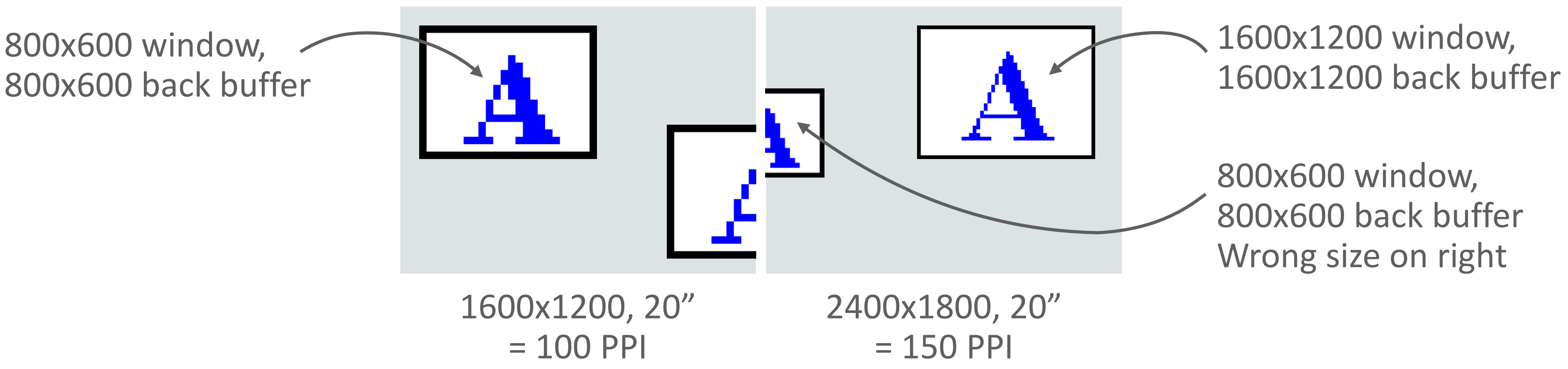
Virtual Screen Space Visualized (moving right to left)



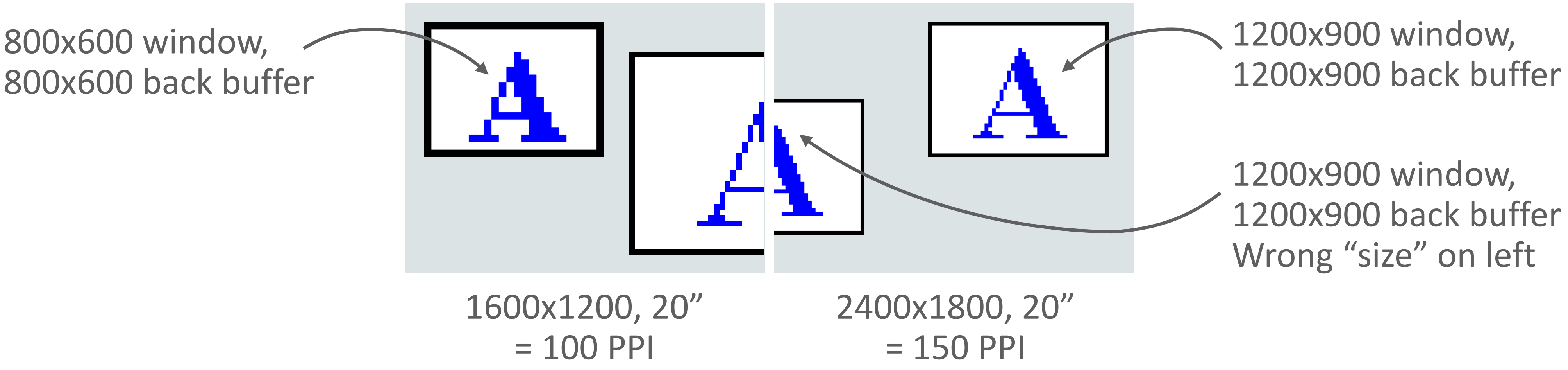
Screen layout approaches (continued)

- Screens in pixel space
  - Measurements relative to a given screen
  - Windows usually dramatically change size as they switch screens
  - Events (and most system APIs) usually integers
  - Every application is essentially a low-level system utility in terms of how it views the hardware
  - Some behaviors impossible or require significant work by the application

Pixel Screen Space Visualized (moving left to right)



Pixel Screen Space Visualized (moving right to left)

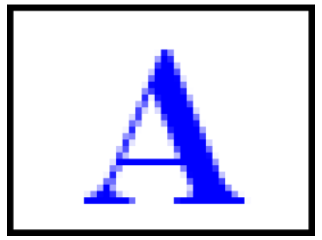


Back Buffer (Render) Scale vs. UI Scale

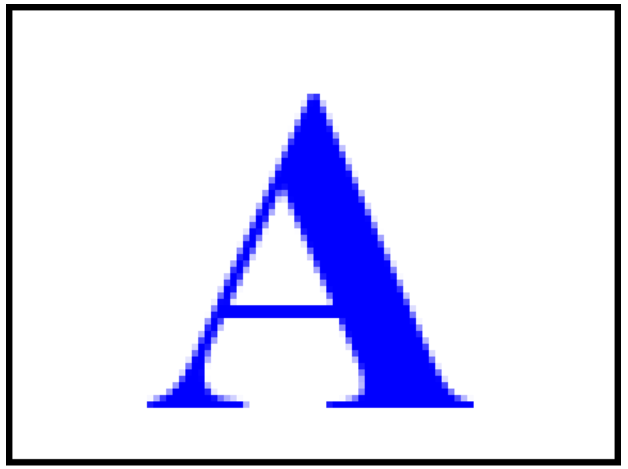
- Platform screen compositing
  - Some platforms render windows to buffers and then composite them to screen
  - Typically the screen compositing is at 1:1 scale, but they can play games
    - Mac screen scaling
    - Windows DPI virtualization
- Double buffering
  - Typically a scene is rendered to a buffer at 1:1 scale with the destination
  - But you can opt to render at a different scale
    - FSAA (Full Scene AntiAliasing)
    - Some code prefers integer coordinates/scales e.g. copyArea scrolling

Back Buffer (Render) Scale vs. UI Scale Visualization

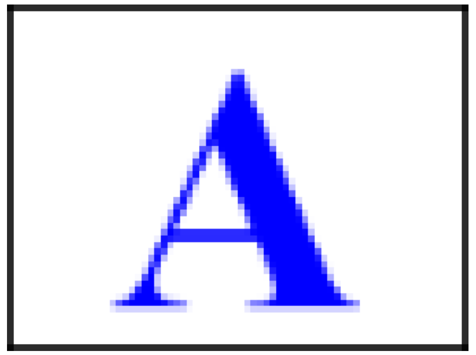
- Rendering a scene to 150% screen using 200% rendering buffer



Scene at 100%



Rendering at 200%
(back buffer)



On screen at 150%

Program Agenda

1. HiDPI terminology and approaches
2. **HiDPI by platform**
3. HiDPI in AWT, Java2D, Swing
4. HiDPI in JavaFX
5. Demos and Examples
6. Issues to be aware of

Mac: History of the retina screen

- June 2010 – iPhone 4
  - 960x640 pixels in 3.5” for 326 PPI
  - Replaced 3GS with 480x320 in 3.5” for 163 PPI
  - Automatic pixel stretching for legacy apps
- June 2012 – retina MacBook Pro
  - 2880x1800 pixels in 15.6” for 220 PPI
  - Replaced regular MBP with 1440x900 pixels in 15.6” for 110 PPI
  - Automatic pixel stretching for legacy applications
  - User-specified scaling factor in Control Panel with “Best for Retina” default

Mac: Retina APIs

- NSScreen.backingScaleFactor
  - 2.0 for retina screens, 1.0 otherwise
    - If user specifies a scale factor other than “Best for Retina” it is still 2.0
    - Thus, scale factor is applied in all cases other than “Best for Retina”
- CALayer.contentsScale
  - Acknowledge 2.0 for retina screens to opt in
- NSImage naming conventions
  - Regular image stored as foo.png, retina version stored as foo@2x.png
  - Both are loaded automatically and used dynamically depending on scale

Mac: HiDPI behaviors and philosophy

- Screens, windows, and events are in virtual coordinates
  - Bounds are reported relative to the scale factor (whether or not you “buy in”)
  - “Buying in” requires you to provide a more detailed back buffer image
  - Windows remain the same perceived size even when moving between monitors
- HiDPI image choosing
  - If high resolution images are provided, system loads them automatically
  - The best size image is used for each rendering request automatically

Windows: History of DPI settings and APIs

- Windows XP
  - LOGPIXELSX/Y provided to indicate screen resolution
    - One value for whole system, never changes until reboot/logout
  - No pixel scaling for any app
- Windows Vista
  - No new DPI information – LOGPIXELSX/Y still used
  - ::SetProcessDPIAware() call provided to indicate DPI awareness
  - Pixel scaling provided for apps that do not indicate awareness
  - Primitive scaling preferences added to Displays Control Panel

Windows: History of DPI settings and APIs (part 2)

- Windows 8.1
  - New DPI-agnostic scaling preferences in Displays Control Panel
  - ::GetDPIForMonitor() – new API for getting monitor resolutions
    - Effective DPI (user preference) and Raw DPI (hardware measurement)
    - Monitor specific and can change dynamically if fully "Per_Monitor" DPI aware
  - ::SetProcessDPIAwareness() – new API for 3 levels of awareness
    - DPI_Unaware (naive), System_DPI_Aware (Vista-style), Per_Monitor_DPI_Aware (new)
  - WM_DPICHANGED event for dynamic changes of scale
  - 3 levels of "virtualization" depending on DPI awareness level
  - No automatic high resolution image loading, but suggested guidelines

Windows: HiDPI Behaviors and Philosophy

- Screens laid out in pixels, Window and event coordinates in pixels
  - Bounds are reported in pixels whether or not you “buy in”
  - “Buying in” requires you to ask for larger windows and draw things larger
  - Window size changes as you change screens of different DPI
    - On Windows 8.1 parts of windows that hang off onto another screen are pixel scaled
    - Problem – as a window leaves one window and changes size, it may revert back
    - Solution – Windows platform uses hysteresis to delay DPI change events
- HiDPI image choosing
  - No support provided, developer must manage media

Linux: DPI

- GTK
  - GDK_SCALE (system-wide) specifies scale to be applied to your window
  - GDK_DPI_SCALE specifies un-scale (<1.0) to be applied to fonts
- Other conventions
  - https://wiki.archlinux.org/index.php/HiDPI

Program Agenda

Platform support in JDK (AWT/Swing)

- Apple retina displays
  - Apple runtimes have supported since Java 6
  - OpenJDK/Oracle runtimes have supported since 7u40
- Windows HiDPI
  - LOGPIXELSX/Y reported as screen resolution
  - Font size scaled by system DPI settings
  - Windows 8.1 style DPI scaling coming soon (JDK-8073320 in final webrevs)
- Linux HiDPI
  - GDK_SCALE support coming soon (JDK-8137571 in final webrevs)

HiDPI Image Support in JDK (AWT/Swing)

- Automatic loading of “@2x” images
  - Supported since 8u20
  - Automatic loading and use based on graphics scale
- High-Resolution image control APIs
  - MultiResolutionImage API in JDK9
  - RenderingHints.KEY_RESOLUTION_VARIANT hints in JDK9

JDK APIs affected by HiDPI

- Follows Virtual Screen model
  - Automatic scaling without application buy-in
  - Windows scaling will not (currently) use intermediate integer render scales
- Default transforms can now be non-identity
  - `AffineTransform GraphicsConfiguration.getDefaultTransform()`
- Screens may now have virtual sizes
  - `Rectangle GraphicsConfiguration.getBounds()`
- Volatile images will be have scaled internal buffers
  - `Image GraphicsConfiguration.createCompatibleVolatileImage(w, h)`

New Multi-Resolution Image interface

- Primary method chooses an image based on a rendered size
  - `Image getResolutionVariant(double destWidth, double destHeight)`
- Convenience method to return all image variants
  - `List<Image> getResolutionVariants()`
- Convenience implementation classes provided
  - `AbstractMultiResolutionImage`
    - Overrides primary methods from the Image interface
      - Developer provides `Image getBaseImage()` method
  - `BaseMultiResolutionImage`
    - Provides a simple implementation based on a list of `Image` objects

New RenderingHints

- `RenderingHints.KEY_RESOLUTION_VARIANT`
  - `RenderingHints.VALUE_RESOLUTION_VARIANT_DEFAULT`
    - May be platform specific
  - `RenderingHints.VALUE_RESOLUTION_VARIANT_BASE`
    - Always use base image
  - `RenderingHints.VALUE_RESOLUTION_VARIANT_SIZE_FIT`
    - Choose based on current rendering transform and parameters
  - `RenderingHints.VALUE_RESOLUTION_VARIANT_DPI_FIT`
    - Choose based on DPI of current screen
- Used by Graphics object when calling `MRI.getResolutionVariant()`

Program Agenda

Platform Support in JavaFX

- Apple retina displays
  - Supported since 7u40
- Windows HiDPI
  - LOGPIXELSX/Y reported as screen resolution
  - Font size scaled by system DPI settings
  - CSS “em” size scales by font size which scales by system DPI settings
  - Windows 8.1 support added in 8u60
    - (Runtime forced into System_DPI_Aware mode by java.exe manifest)
- Linux (TBD: JDK-8137050)

Image support in JavaFX

- Image loading of “@2x” images
  - Supported since JDK8
  - Automatic loading of either regular or “@2x” version of image
    - Only one or the other is loaded
    - Based on the deepest screen attached at time of loading
  - Dynamic loading of multiple resolutions (TBD)
- High resolution image control APIs (TBD)

HiDPI settings available in JavaFX

- Virtual Screen layouts presented to developers
  - Coordinates for Window/Stage bounds and root Node parent coordinates are virtual
- All platforms
  - `-Dprism.allowhidpi={true|false}`
- Windows-specific
  - `-Dglass.win.minHiDPI=N.M` (scale factor, default=1.5)
  - `-Dglass.win.uiScale={N.M|N%|Ndpi}` (no default)
  - `-Dglass.win.renderScale={N.M|N%|Ndpi}` (no default)
  - `-Dglass.win.forceIntegerRenderScale={true|false}` (default=true)
- JavaFX in 8u60 defaulted to integer intermediate render scales

Future API support in JavaFX

- Detection of/adaptation to scaling
  - Region provides `snap*()` methods which assume 1:1 scaling
    - Existing `snap*()` methods are compatible with integer scaling as well
    - As a result, current Windows scaling provides only integer render scales
  - New API being added to Window and Screen
    - `Screen.getOutputScaleX/Y()`
    - `Window.getOutputScaleX/Y()`
    - `Window.getRenderScaleX/Y()`
    - All `snap*()` methods updated for non-integer scales, `snap*X()` and `snap*Y()` methods added
    - Assumption: all parents of a control have been snapped to a pixel location
- JavaFX in JDK9 will most likely do away with integer render scales

Program Agenda

1. HiDPI terminology and approaches
2. HiDPI by platform
3. HiDPI in AWT, Java2D, Swing
4. HiDPI in JavaFX
5. **Demos and Examples**
6. Issues to be aware of

Multi-Resolution Image Example

```
public class MultiResImageExample extends JPanel {
    public static final int IMG_W = 175;
    public static final int IMG_H = 100;
    public static final int PANEL_W = 600;
    public static final int PANEL_H = 500;

    public static Image createImage(Color bg, double scale) {
        int w = (int) Math.ceil(IMG_W * scale);
        int h = (int) Math.ceil(IMG_H * scale);
        BufferedImage bimg = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = bimg.createGraphics();
        g2d.setColor(bg);
        g2d.fillRect(0, 0, w, h);
        g2d.scale(scale, scale);
        g2d.setFont(new Font(Font.SERIF, Font.BOLD, 90));
        g2d.setColor(Color.BLACK);
        g2d.drawString(Double.toString(scale), 10, 90);
        return bimg;
    }
```

Multi-Resolution Image Example (continued)

```
public static Image createMRI(Color bg, double... scales) {
    Image base = createImage(bg, 1.0);
    if (scales == null) return base;
    Image variants[] = new Image[scales.length+1];
    variants[0] = base;
    for (int i = 0; i < scales.length; i++) {
        variants[i+1] = createImage(bg, scales[i]);
    }
    return new BaseMultiResolutionImage(variants);
}

static final double DEF_SCALES[] = { 2.0, 3.0, 4.0, 5.0 };
…
    MultiResImageExample mrie1 =
        new MultiResImageExample(createMRI(Color.GREEN, DEF_SCALES));
    MultiResImageExample mrie2 =
        new MultiResImageExample(createMRI(Color.YELLOW, null));
    mrie1.setPreferredSize(new Dimension(PANEL_W, PANEL_H));
    mrie2.setPreferredSize(new Dimension(PANEL_W, PANEL_H));
```

Multi-Resolution Image Example (continued)

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g.create();
    g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
                         RenderingHints.VALUE_INTERPOLATION_BILINEAR);
    g2d.translate(getWidth()/2.0, getHeight()/2.0);
    g2d.scale(scale, scale);
    g2d.translate(-IMG_W/2.0, -IMG_H/2.0);
    g2d.drawImage(img, 0, 0, this);
    g2d.dispose();
    g2d = (Graphics2D) g.create();
    g2d.setFont(new Font(Font.DIALOG, Font.BOLD, 24));
    g2d.drawString("Render scale = "+scale, 20, getHeight()-10);
    String str = img instanceof MultiResolutionImage
            ? "MultiRes Image" : "Normal Image";
    g2d.drawString(str, 20, 30);
    g2d.dispose();
}
```

Show demo

Custom Multi-Resolution Image Example

```
public static class MyMRI extends AbstractMultiResolutionImage {
    Image variants[];
    int baseWidth, baseHeight;
    public MyMRI(Image... variants) {
        this.variants = variants;
        this.baseWidth = variants[0].getWidth(null);
        this.baseHeight = variants[0].getHeight(null);
    }

    public Image getBaseImage() { return variants[0]; }

    public Image getResolutionVariant(double destWidth, double destHeight) {
        return variants[(int) (Math.random() * variants.length)];
    }

    public List<Image> getResolutionVariants() {
        return Arrays.asList(variants);
    }
}
```

Show demo

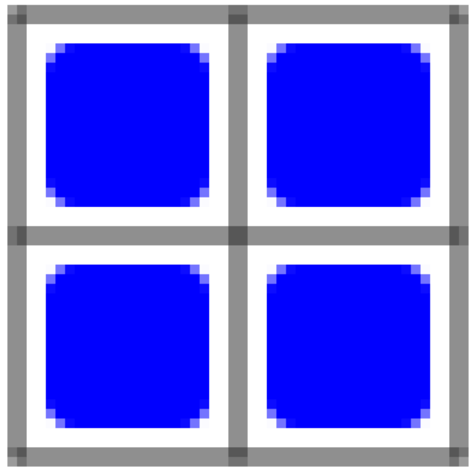
Snap to Pixel examples

```
public class SnapToPixelExample extends Application {
    public Node makeChild() {
        Pane p = new Pane();
        p.setPrefSize(17, 17);
        p.setStyle("-fx-background-color: BLUE; -fx-background-radius: 3");
        GridPane.setMargin(p, new Insets(3));
        return p;
    }

    @Override
    public void start(Stage stage) {
        GridPane gp = new GridPane();
        gp.setGridLinesVisible(true);
        gp.setPadding(new Insets(5));
        gp.add(makeChild(), 0, 0);
        gp.add(makeChild(), 0, 1);
        gp.add(makeChild(), 1, 0);
        gp.add(makeChild(), 1, 1);
```

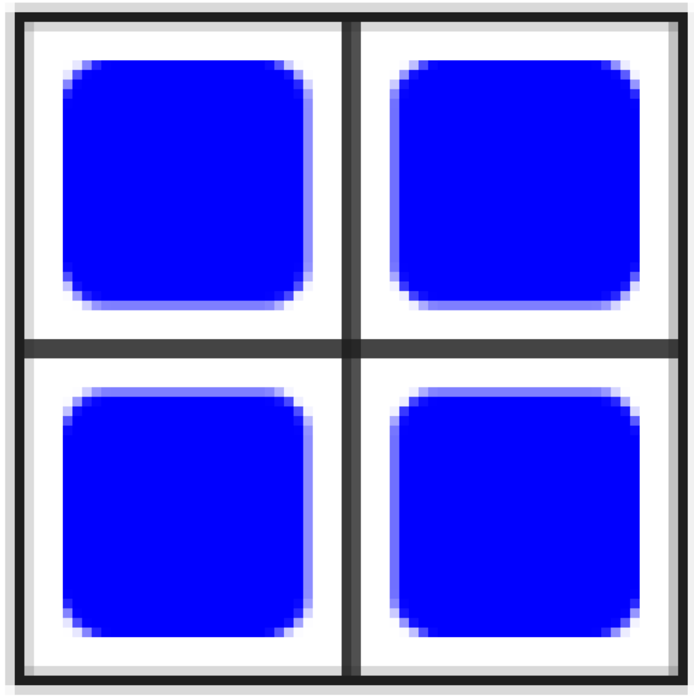
Snap to Pixel Example running on JDK8

- Using uiScale=100%
- Using render scale=100%



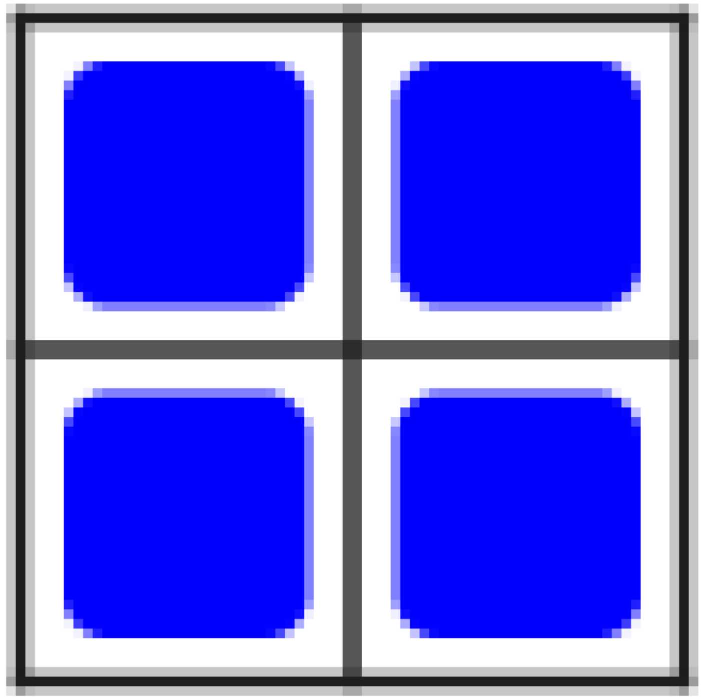
Snap to Pixel Example running on JDK8

- Using uiScale=150%
- Using integer render scale=200%



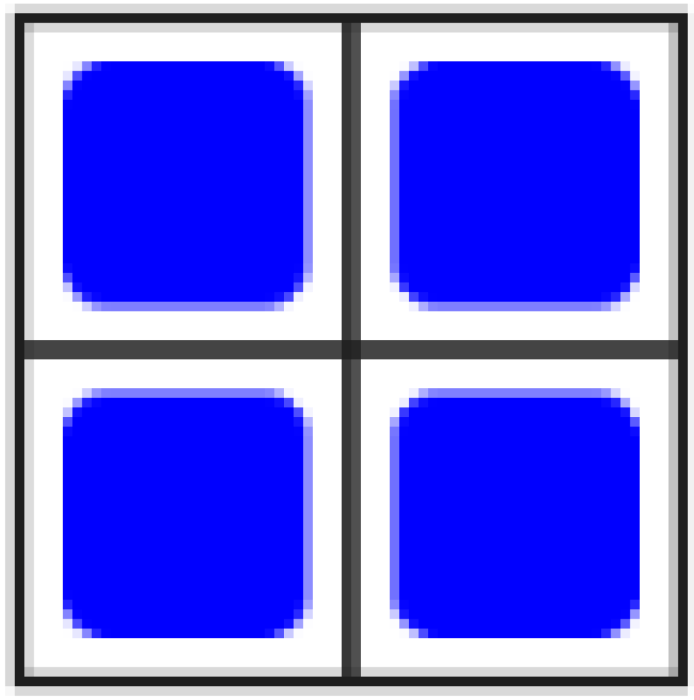
Snap to Pixel Example running on JDK8

- Using uiScale=150%
- Using renderScale=150%



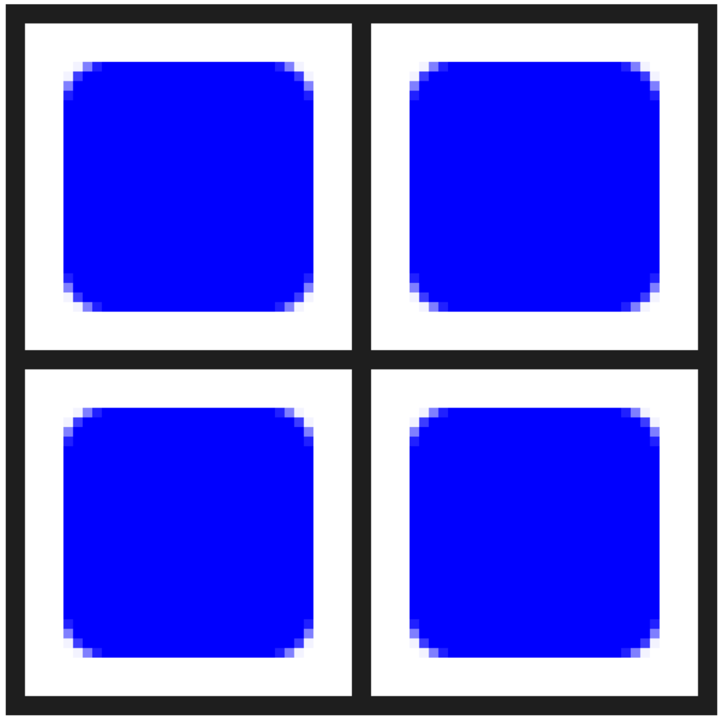
Snap to Pixel Example running on JDK9

- Using uiScale=150%
- Using integer render scale=200%



Snap to Pixel Example running on JDK9

- Using uiScale=150%
- Using renderScale=150%



Program Agenda

1. HiDPI terminology and approaches
2. HiDPI by platform
3. HiDPI in AWT, Java2D, Swing
4. HiDPI in JavaFX
5. Demos and Examples
6. Issues to be aware of

Integer coordinates

- APIs that have only integer values
  - AWT component locations and sizes
  - AWT event locations
  - Swing component locations and sizes
  - Graphics object methods

Integer coordinates (continued)

- Programmer assumptions about pixel locations
  - I hard-code integers, therefore I'll line up with pixels, right?
  - JavaFX code that rolls its own "snapToPixel" APIs
    - `Math.round()` becomes `(Math.round(v * scale) / scale)`
    - `Math.ceil()` becomes `(Math.ceil(v * scale) / scale)`
  - Lines drawn at pixel edges with odd stroke widths
  - Lines drawn with Java2D `STROKE_CONTROL` hint

Default transforms

- `Graphics2D`
  - Do not use `setTransform()` except to restore after `getTransform()`
- `GraphicsConfiguration`
  - `getDefaultTransform()` used to always return Identity, not any more…
- JavaFX new APIs to get output and/or render scale
- This is all similar to the behavior when printing

Performance

- Rendering more pixels
  - Should be mitigated by newer displays being paired with newer GPUs
- Optimizations that punt when scales are present
  - JavaFX scrolling optimizations

Who to contact

- JDK issues/help
  - swing-dev@openjdk.java.net
- JavaFX issues/help
  - openjfx-dev@openjdk.java.net
- JDK9 Early Access
  - https://jdk9.java.net/download/
- Bugs
  - JDK Bug System for any of AWT/Swing/JavaFX
  - https://bugs.openjdk.java.net/

Safe Harbor Statement

JavaOne™

ORACLE®

ORACLE®