Applause from Michelé Clarke, Nicholas Pinn, and 15 others

Adrian D. Finlay
@thewiprogrammer. Writer @hackernoon. Code, LOTS of it. Mangos, LOVE THEM! Barbering. Health. Travel. Business. & more! Network w/ me @ adriandavid.me/network
Sep 30, 2017 · 6 min read

What's New in Java FX—Java 9 Updates

JavaFX Logo [2]

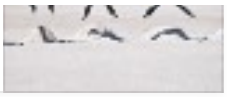
JavaFX is a 3rd Generation Java GUI platform (after AWT, Swing) for the development of Desktop and Rich Internet Applications. As of JDK8, It is a part of Java SE. JavaFX provides resources to build thick client applications that interact with Java SE, ME, and EE. JavaFX supports Windows Vista—Current, Mac OSX—macOS, and various Linux operating systems. There are also ports for various mobile platforms. The current version of Java FX mirrors the JDK numbering as **Java FX 9**. **An overview of the updates to the Java FX Platform of JDK release 9 will be the focus of this article**.

Not Caught up to Java 9? Learn more below:

New Language Features in Java 9

Java 9 is here!

medium.com



Cool Java 9 Features You Might've Missed

While the predominant features of Java SE 9 were the modularity efforts of Project Jigsaw, Java 9...

medium.com

JavaFX happens to be one of my favourite technologies :) I love Desktop Apps. I love Java. And I also like what Danny Coward & the Java Client Group did with JavaFX; It is a refreshing breath in the right direction, learning the lessons from it's predecessors: AWT & Swing.

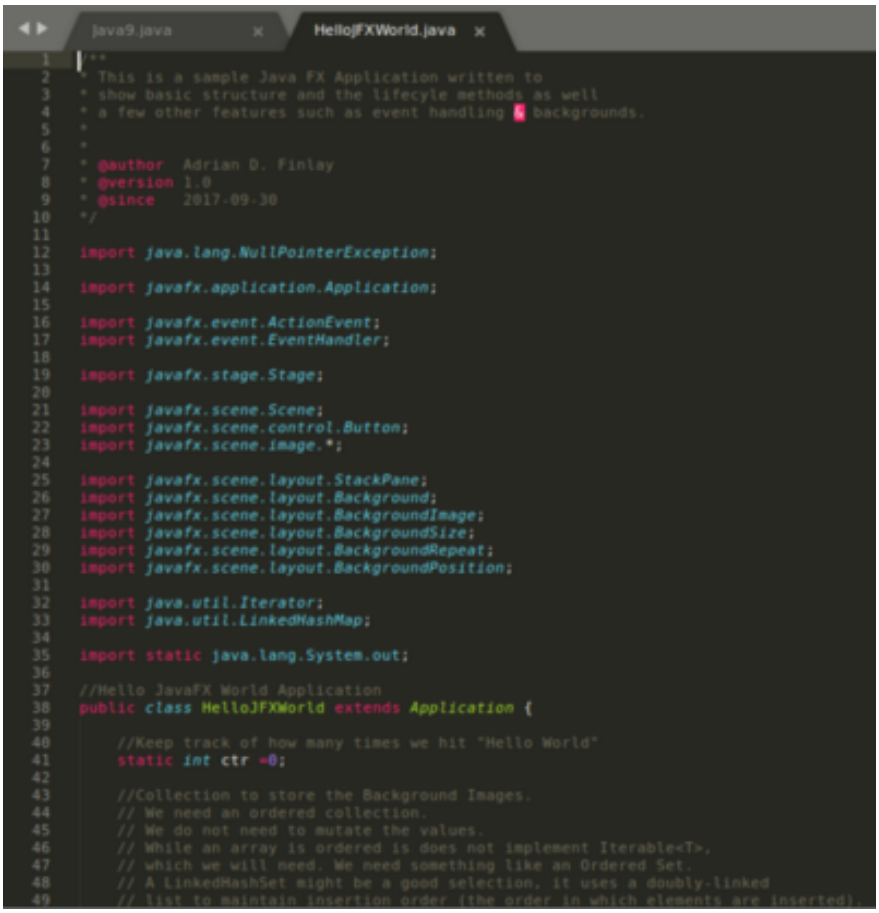
Because of the heavy focus on Project Jigsaw (Modularity), JDK 9 included few upgrades to java language features as well as JavaFX APIs. **The highlights of Java FX updates in JDK 9 are the Skin/CSS APIs and Jigsaw modularization**.

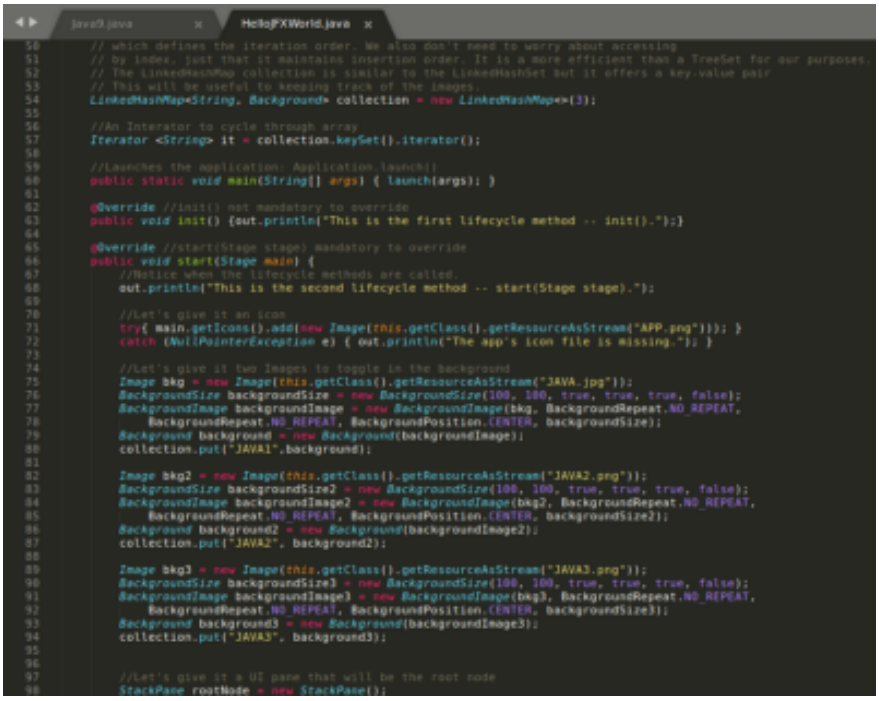
> *JavaFX License*

JavaFX is dual licensed under the **GPL v2 with Classpath Exception** and **Oracle Binary Code License for the Java SE Platform**[5] with a commitment to go fully open source in the foreseeable future[3]. openJFX is the community effort behind JavaFX.

A Brief Java FX Primer—Hello World

While this article will not examine the JavaFX APIs in depth, if you've never seen a JavaFX Application before, below are some screenshots of a Hello World application.


Java9.java
HelloJFXWorld.java
/**
* This is a sample Java FX Application written to
* show basic structure and the lifecyle methods as well
* a few other features such as event handling & backgrounds.
*
*
* @author Adrian D. Finlay
* @version 1.0
* @since   2017-09-30
*/
import java.lang.NullPointerException;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.*;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundImage;
import javafx.scene.layout.BackgroundSize;
import javafx.scene.layout.BackgroundRepeat;
import javafx.scene.layout.BackgroundPosition;
import java.util.Iterator;
import java.util.LinkedHashMap;
import static java.lang.System.out;
//Hello JavaFX World Application
public class HelloJFXWorld extends Application {
//Keep track of how many times we hit "Hello World"
static int ctr =0;
//Collection to store the Background Images.
// We need an ordered collection.
// We do not need to mutate the values.
// While an array is ordered is does not implement Iterable<T>,
// which we will need. We need something like an Ordered Set.
// A LinkedHashSet might be a good selection, it uses a doubly-linked
// list to maintain insertion order (the order in which elements are inserted),


Java9.java
HelloJFXWorld.java
// which defines the iteration order. We also don't need to worry about accessing
// by index, just that it maintains insertion order. It is a more efficient than a TreeSet for our purposes
// The LinkedHashMap collection is similar to the LinkedHashSet but it offers a key-value pair
// This will be useful to keeping track of the images
LinkedHashMap<String, Background> collection = new LinkedHashMap<>(3);
//An Interator to cycle through array
Iterator <String> it = collection.keySet().iterator();
//Launches the application. Application.launch()
public static void main(String[] args) { launch(args); }
@Override //init() not mandatory to override
public void init() {out.println("This is the first lifecycle method -- init().");}
@Override //start(Stage stage) mandatory to override
public void start(Stage main) {
//Notice when the lifecycle methods are called.
out.println("This is the second lifecycle method -- start(Stage stage).");
//Let's give it an icon
try{ main.getIcons().add(new Image(this.getClass().getResourceAsStream("APP.png"))); }
catch (NullPointerException e) { out.println("The app's icon file is missing."); }
//Let's give it two Images to toggle in the background.
Image bkg = new Image(this.getClass().getResourceAsStream("JAVA.jpg"));
BackgroundSize backgroundSize = new BackgroundSize(100, 100, true, true, true, false);
BackgroundImage backgroundImage = new BackgroundImage(bkg, BackgroundRepeat.NO_REPEAT,
BackgroundRepeat.NO_REPEAT, BackgroundPosition.CENTER, backgroundSize);
Background background = new Background(backgroundImage);
collection.put("JAVA1",background);
Image bkg2 = new Image(this.getClass().getResourceAsStream("JAVA2.png"));
BackgroundSize backgroundSize2 = new BackgroundSize(100, 100, true, true, true, false);
BackgroundImage backgroundImage2 = new BackgroundImage(bkg2, BackgroundRepeat.NO_REPEAT,
BackgroundRepeat.NO_REPEAT, BackgroundPosition.CENTER, backgroundSize2);
Background background2 = new Background(backgroundImage2);
collection.put("JAVA2", background2);
Image bkg3 = new Image(this.getClass().getResourceAsStream("JAVA3.png"));
BackgroundSize backgroundSize3 = new BackgroundSize(100, 100, true, true, true, false);
BackgroundImage backgroundImage3 = new BackgroundImage(bkg3, BackgroundRepeat.NO_REPEAT,
BackgroundRepeat.NO_REPEAT, BackgroundPosition.CENTER, backgroundSize3);
Background background3 = new Background(backgroundImage3);
collection.put("JAVA3", background3);
//Let's give it a UI pane that will be the root node
StackPane rootNode = new StackPane();

Some screenshots of the execution. The background toggles on the button event's (clicking off the Hello JavaFX World! button) firing.

Want the source? Grab it here.

afinlay5/JavaFX9

Gradle source code repository for the Java FX 9 source code example posted on my personal blo...

github.com

So What's New?

#1—Modularization (Project Jigsaw)

The first thing to notice is that JavaFX is now broken up into **seven modules [7]:**

- **Base APIs for JavaFX UI Toolkit**—javafx.base
- **JavaFX APIs for UI Controls**—javafx.controls
- **JavaFX APIs for FXML**—javafx.fxml
- **JavaFX APIs for various Graphics Tools**— javafx.graphics
- **JavaFX APIs for Multimedia**—javafx.media
- **JavaFX APIs for Swing-JavaFX Interoperability**—javafx.swing
- **JavaFX APIs for WebView Functionality**—javafx.web

The advantages of modularity include improved scalability, ability to test software components independently without breaking code other code or having to recompile an entire system, among others.

The javapackager can now give you an application with only the modules you need. All modular JavaFX applications require the javafx.base & javafx.graphics modules. All modular Java FX UI Applications require the javafx.controls module. Packages may only belong to one module. Modular applications will need to list dependencies in module-info.java.

JEP 253 & **JEP 200** were the community efforts towards modularizing the JDK and the JavaFX UI Controls & CSS APIs. They came about by the overall push towards a modular JDK and was incentivized by the advantage of providing public availability for functionality found only in internal APIs.

#2—Improved Encapsulation

Java FX 9 brings stricter encapsulation on Java FX API internals. This provides some enforcement for better compliance of avoiding the use of the Java FX internal APIs. Only public types of explicitly exported packages are available[3]. While there is a work-around, I won't mention it, because you should resist the urge to use it ;). Think within the API.

The APIs: com.sun* packages and impl__ methods have been either deprecated, replaced, or made public, now that JDK 9 modularity hides non-public packages. Node builders should be avoided. Class.getResource should be used instead of ClassLoader.getResource(). Classpath relative URL strings should be avoided.

#3—New API

Commonly, developers were dipping into com.sun.javafx.scene.control.skin to develop Java FX skins. This means that developers were relying on private API. Often enough, the grounds for dipping into JavaFX internal APIs could be argued by the need for functionality. In JDK 9 a new API A new API, javafx.scene.control.skin was released to address this need.

A concomitant consequence of the push towards JDK modularity led to the opportunity to provide public implementation for a lot of very useful private API, hence JEP 253. Beginning in June of 2015, members of the Java Client Group and openJFX began polling the community for the most useful private API. They found the following API to be the most wanting of examination: UI Control Skins, CSS APIs, Toolkit / Platform APIs, Robot, Performance Tracker[3]. All but the Robot and Performance Tracker APIs were addressed in JDK 9.

56 new skin API classes and 11 CSS classes were added in JDK 9[6].

#4—Graphical Improvements

In JDK 9, Java FX added High-DPI support and improvements for Linux and Windows[3].

GTK3 Linux Support—
At present, Ubuntu 16.04 LTS & SWT 4.6 and other platforms have migrated to GTK3. JDK 9 may outlive GTK 2 support. To ensure continuing functionality, JDK 9 added GTK3 support. The community effort behind this was JEP 283.

#5—Updates to GStreamer, WebKit

The GStreamer platform version was upgraded to the latest version and WebKit was updated with performance enhancements and bug fixes [3].

#6—Miscellaneous API Enhancements and Bug Fixes

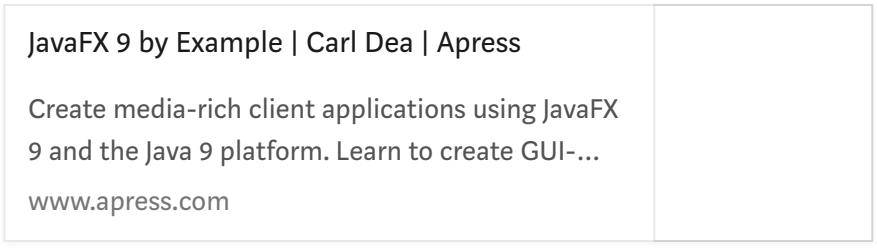
At least 102 small enhancements in the Tooltip, ComboxBox, Spinner, Node, Font, Collections, & FXPermission APIs were added and more than 756 bug fixes have been made[3].

Where to learn JavaFX?

Pro JavaFX 9

In my opinion, this is the defacto reference text on JavaFX 9. Written by the experts, this is the most expansive text on JavaFX 9 to date and should be familiar reading for **every** serious JavaFX developer.

JavaFX 9 by Example



Another stellar text from Apress press. This is an excellent primer for JavaFX newbies that want to get a solid, rigorous introduction to the platform. Experts can learn from this text as well. I learned about the JavaFX printing API from this text.

Looney Tunes Ending [8]

Works Cited

[1]—https://en.wikipedia.org/wiki/JavaFX

[2]—https://blog.idrsolutions.com/2013/09/java-articles-index-understanding-the-world-of-java-and-java-fx/

[3]—JavaOne Talk—JavaFX: New & Noteworthy, Kevin Rushforth & Jonathan Giles, Java Client Group, September 2016

[4]—http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html

[5]—Binary Code License Agreement for the Java SE Platform Products and JavaFX

[6] —Oracle Official Java FX 9 API Documentation

[7]—JEP 253: Prepare JavaFX UI Controls & CSS APIs for Modularization

[8]—https://www.youtube.com/watch?v=0FHEeG_uq5Y