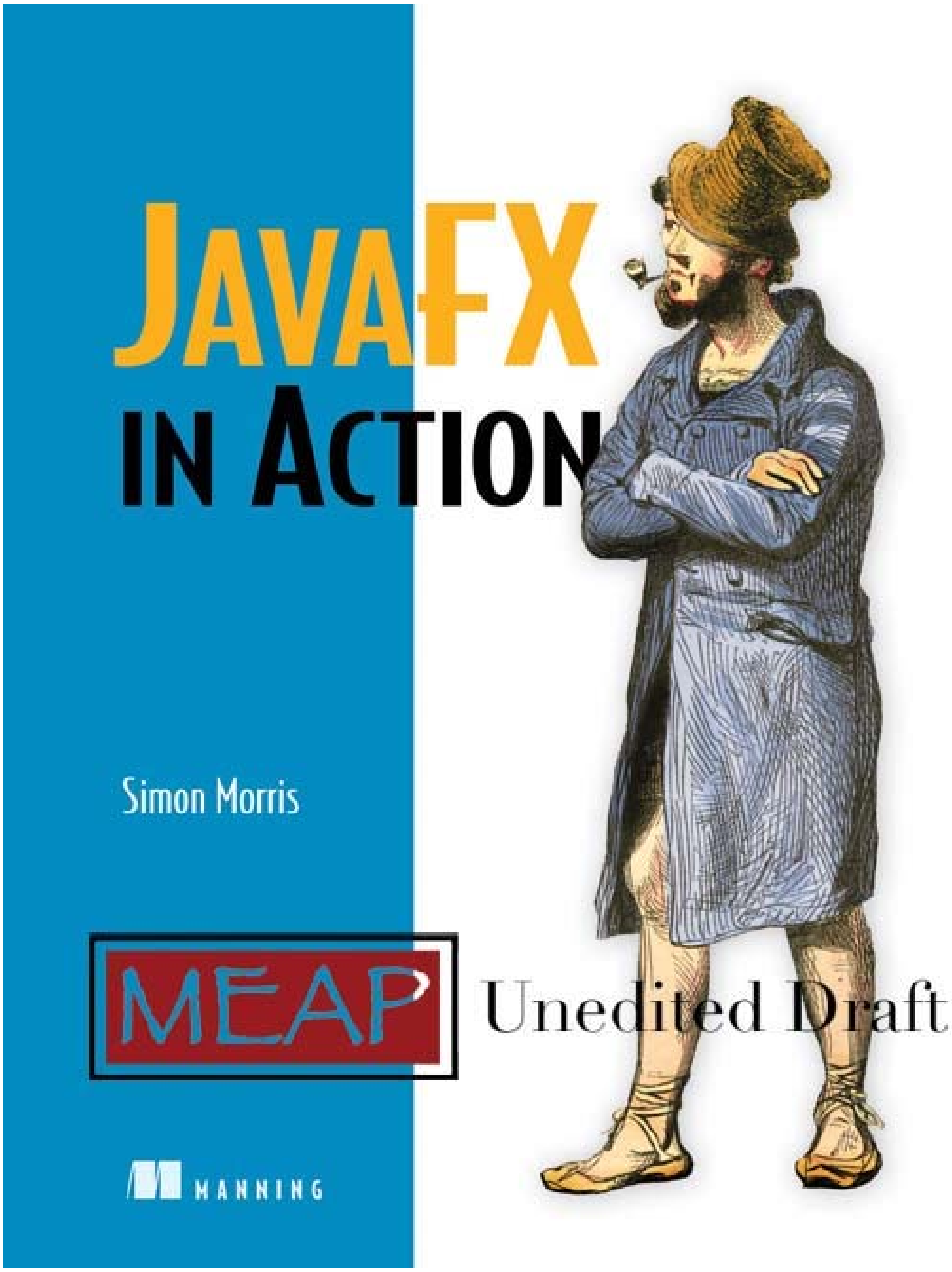

JAVAFX
IN ACTION
Simon Morris
MEAP
Unedited Draft
MANNING

MEAP Edition
Manning Early Access Program

Table of Contents

1 Welcome to the future: introducing JavaFX

Graphics programming used to be fun!

Early personal computer software sported predominantly character based user interfaces. Bitmap displays were too expensive, although some computers offered the luxury of hardware sprites – for the programmer, the simple act of poking values into RAM gave instant visual gratification.

These days things are a lot more complicated, we have layers of abstraction separating us from the hardware. Sure, they give us the wonders of scrollbars, rich text editors and tabbed panes, but they also constrain us. The World Wide Web raised the bar, users now expect glossier visuals, yet the tools that we programmers use to create desktop software are in essence little evolved from the days of the first Macintosh or Amiga.

The web also heralded a shift in the way we use our software. Increasingly data is moving away from the hard disk and onto the internet. Our tools are also starting to move that way, yet the fledgling attempts to build on-line applications using HTML and Ajax have resulted in nothing more than pale imitations of their desktop cousins.

If only there was a purpose-built tool for writing the next generation of internet software, one which could deliver the same rich functionality as a desktop application, yet with drop-dead gorgeous visuals and rich media content within easy reach, delivered to whatever device – PC or smart phone – we wanted to work from today.

Sounds too good to be true? Let me introduce you to JavaFX!

JavaFX is a family of complementary technologies for writing a new generation of internet applications, spanning the desktop, the web, and mobile devices. These so-called *Rich Internet Applications* have the power of desktop software, but the omnipresence of a web page. Oh, and they offer eye-popping visuals too!

At the heart of JavaFX is *JavaFX Script*, an innovative, compiled, *domain specific language* (DSL) for creating visually rich user interfaces. Unlike other languages, JavaFX Script boasts a declarative syntax: the code structure mirrors the structure of the interface, meaning associated UI pieces are kept in one efficient bundle, not strewn across multiple locations. Meanwhile simple language constructs smooth the pain of updating and animating the interface, reducing code complexity while increasing productivity.

The JavaFX platform, as its name suggests, sits atop the existing Java Standard and Micro Editions, with current Java packages readily accessible from the JavaFX environment. JavaFX specific libraries are also available for sophisticated presentation and animation, taking advantage of JavaFX Script's unique language features.

In short: JavaFX makes user interface programming fun again!

1.1 Why do we need JavaFX?

A very good question – why **do** we need yet another language? The World is full of programming language, wouldn't one of the existing language do? Perhaps JavaScript, or Python, or what about Scala? Indeed, what's wrong with Java?

Certainly JavaFX makes writing slick graphical applications easier, but is there more to it than that?

In a broader sense JavaFX attempts to respond to a change in the eco-system of user facing software over the last twenty years, and particularly since the mid to late 1990s when the World Wide Web began to rise in prominence. To understand why JavaFX is the answer we would be well served to spell out precisely what the question is. Technology moves so quickly and in so many different directions these days that it is vital we understand the exact scope of the problem JFX is attempting to solve. So, let us first consider what makes the JavaFX Script language special for graphics programming, then provide a little context by examining the shifting trends in the software market since the arrival of the World Wide Web, before finally rounding off by seeing how JavaFX addresses the problems of this new environment.

1.1.1 If I had a hammer

"*When the only tool you have is a hammer, every problem looks like a nail*", the popular saying goes.

Language advocacy is a popular pastime with many programmers, but what many fail to realize, be they 'fanboys' of Java or C# or Python or COBOL (do we have any octogenarian programmers reading?) is that programming languages are like tools: each is good at some things, and next to useless at others.

Java, inspired as it was on prior art like C and Smalltalk, sports a solid general purpose syntax which gets the job done with the minimum of fuss in the majority of cases. Unfortunately there will always be those areas which, by their very nature, demand something a little more specialized. Graphics programming has typically been one such area, with pretty much every popular language developed over the decades struggling to provide anything more than stodgy, cumbersome, support.

What makes graphics programming such an ill fit for modern programming languages? I'm sure if we asked a dozen experts we'd get thirteen opinions, but let me (your humble author) risk having a stab at defining a couple of likely candidate problems.

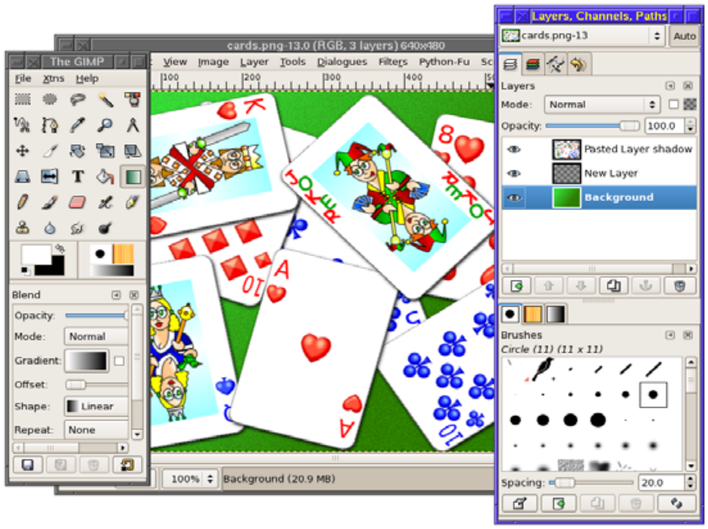


Figure 1.1 A complex GUI typical of modern desktop applications. Two windows host scrolling control palettes, while another holds an editable image and rulers.

Firstly, graphics generally require quite large nested data structures: trees of elements, each element providing a baffling array of configurable options and behaviors. Procedural languages like to work in clearly delineated steps, but this linear pattern conflicts with the tree pattern inherent in most GUIs.

Secondly, graphics code tends to rely heavily on concurrency – processes running in parallel. Modern UI fashions have amplified this requirement, with several transition effects often running within a single interface simultaneously. Unfortunately the heavy boilerplate code demanded by many languages to spawn and manage new threads makes these kind of animations cumbersome to construct.

Perhaps you can think of other problems, but the above two (at least in my experience) seem to create more than enough trouble between them. In the next chapter, and continuing throughout this book, we'll learn how JavaFX goes about defeating them.

But we've only considered part of the story – JavaFX isn't just about slick visuals, it's also an important weapon in the arms race for the Rich Internet Application market. But what is a Rich Internet Application, and why is it important? To answer those questions we need to take a brief trip back in time.

1.1.2 Back to the future

Douglas Adams once wrote "*I suppose the best way to find out where you come from is to find out where you're going, and then work backwards.*"

The internet has become such a dominant cultural force, our grandchildren will likely never believe us when we regale them with exotic tales of a life without a permanent broadband connection. "*But how did your computers **ever** work, Grandpa?*"

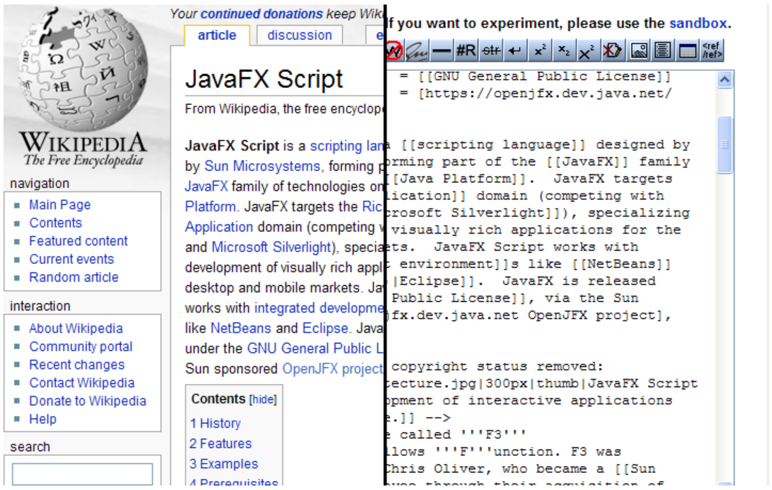
In the pre-internet age software was installed straight onto the hard drive. If, at your local bar, you were overcome by a sudden urge to share with a friend your latest poetic masterpiece, you needed to hand both the poem file itself and the software to open it. Chances are neither would be available.

Although your friend might be eternally grateful for this fact, clearly it was a problem which needed a solution.

And a solution began to take form with the sudden explosion into the popular consciousness of an exciting new technology with the remarkable bizarre name of “The World Wide Web”. Initially allowing no interactivity between user and server beyond the clicking of a hyper-link, the web rapidly evolved to accommodate the sending of basic form data with each page request. This meant pages were no longer static, but could be dynamically generated in response to parameters passed in by the visitor.

Initial web applications confined themselves to humble database lookups: what time is the last bus?, who played the three wise monkeys in the movie Planet of the Apes?, which pages have the words “Pamela”, “Anderson” and “pictures” on them? But bit by bit novel ways were being found to utilize the simple query/response mechanism. It started innocently enough with the likes of web based e-mail, then eventually spread to calendars, personal photo galleries, word processor and spreadsheets. Pretty soon the web was awash with clever new applications.

This marked a fundamental shift in the relationship between site and visitor. Previously the site held content which the visitor browsed or queried; but now, with the advent of web email and the like, the site provided no content itself but relied entirely on users to populate it. The role of the site had moved from information source, to storage depot, and the role of the visitor from passive consumer to active producer.



JavaFX Script

From Wikipedia, the free encyclop

JavaFX Script is a scripting lan
by Sun Microsystems, forming p
JavaFX family of technologies on
Platform. JavaFX targets the Ric
Application domain (competing w
and Microsoft Silverlight), specia
development of visually rich appl
desktop and mobile markets. Ja
works with integrated developme
like NetBeans and Eclipse. Java
under the GNU General Public L
Sun sponsored OpenJFX project

Contents [hide]
1 History
2 Features
3 Examples
4 Prerequisites

```
= [[GNU General Public License]]
= [https://openjfx.dev.java.net/

 [[scripting language]] designed by
rming part of the [[JavaFX]] family
[Java Platform]].  JavaFX targets
ication]] domain (competing with
rosoft Silverlight]]), specializing
visually rich applications for the
ts.  JavaFX Script works with
 environment]]s like [[NetBeans]]
|Eclipse]].  JavaFX is released
Public License]], via the Sun
fx.dev.java.net OpenJFX project],

copyright status removed:
ecture.jpg|300px|thumb|JavaFX Script
pment of interactive applications
.]] -->
 called '''F3'''
lows '''F'''unction. F3 was
hris Oliver, who became a [[Sun
```

Figure 1.2 Wikipedia is an on-line encyclopedia constructed entirely by contributions from its users. The “Edit this page” link above each article allows the reader to change, in real time, the page's content.

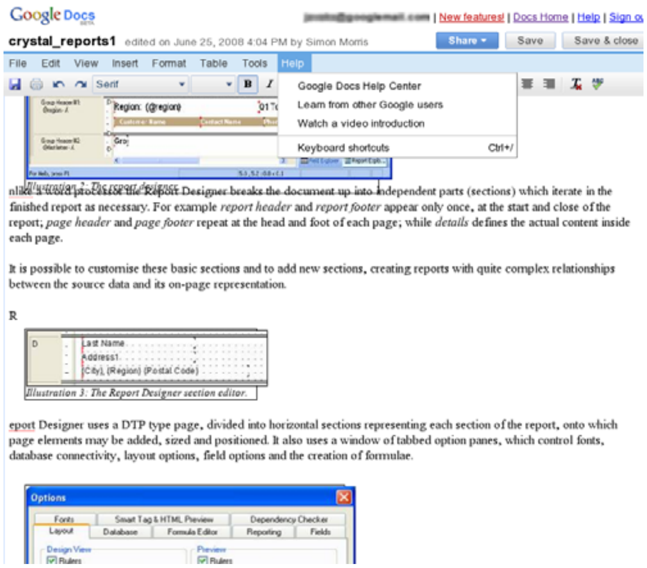
A new meme was emerging: the concept of User Generated Content. Sites would be populated by the users themselves, millions of contributors pooling their resources to create information resources on a spectacular scale. The poster child for this revolution was (and still is, to a large extent) Wikipedia, the on-line encyclopedia in which all of the content is provided by the readers.

Don't think the entry on "The Mighty Boosh" is up to scratch? Just click “EDIT” and change it yourself! No longer would big organizations be able to shape and control information – the inmates were taking over the asylum!

Some were horrified by the prospect – wouldn't this just create a jumble of untrustworthy information from millions of unqualified contributors? – while others began to revel at what they saw as a true democratization of information. Sometime around 2004 publisher and Open Source advocate Tim O'Reilly coined the term "*Web 2.0*" to address the phenomena, and it quickly fell into use (and abuse) as a hot new buzz phrase.

The explosion in web based applications drove demand for more sophisticated user interfaces. This new generation of site, which attempted to ape the look and feel of traditional desktop software, became blessed with the moniker "*Rich Internet Application*" after Macromedia (subsequently purchased by Adobe) coined the term in a 2002 white paper noting the transition of applications from the desktop onto the web, and asking how they might be improved (unsurprisingly the answer involved Macromedia products.)

But despite all the enthusiasm, real progress was slow and frustrating. While the likes of Ajax helped paper over some of the cracks, at its heart the web was designed to show content, not run software. Web content is 'poured' into the window, left to right down the page echoing the technology's publishing origins, while input is predominantly restricted to basic form components. Mimicking the look and functionality of a powerful application-centric technology inside a document-centric environment was not easy, as numerous web developers soon discovered. Yet still they tried...



crystal_reports1 edited on June 25, 2008 4:04 PM by Simon Morris

Illustration 2: The report designer

nlike a word processor the Report Designer breaks the document up into independent parts (sections) which iterate in the finished report as necessary. For example *report header* and *report footer* appear only once, at the start and close of the report; *page header* and *page footer* repeat at the head and foot of each page; while *details* defines the actual content inside each page.

It is possible to customise these basic sections and to add new sections, creating reports with quite complex relationships between the source data and its on-page representation.

R

Illustration 3: The Report Designer section editor.

eport Designer uses a DTP type page, divided into horizontal sections representing each section of the report, onto which page elements may be added, sized and positioned. It also uses a window of tabbed option panes, which control fonts, database connectivity, layout options, field options and the creation of formulae.

Figure 1.3 Google Docs runs inside a browser and has a much simpler GUI than Microsoft Office or OpenOffice.org, even struggling to render a simple Word file correctly.

At the bleeding edges of the software development world some programmers dared to commit heresy; they asked whether the web browser was really the best platform for creating Rich Internet Applications?

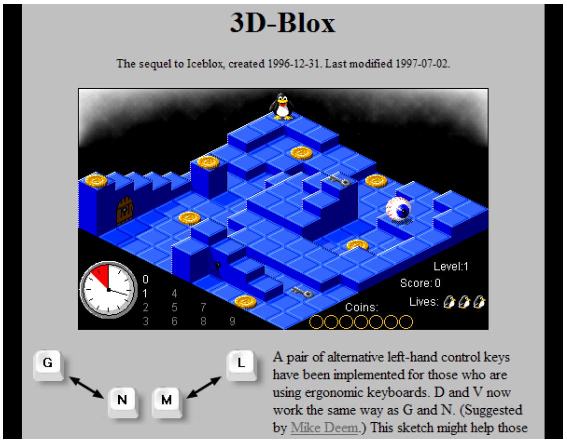
They looked back to see where they had come from, and saw a wealth of elegant desktop software with high fidelity user interfaces and sophisticated user interactivity. But this software was written using the old desktop toolkits which bound themselves firmly to their OS. Not only would software written using the Mac UI libraries fail miserably on Windows (and vice-versa) but the desktop libraries themselves were written back in an age when all software was run from a local disk – huge monolithic beasts which occupied dozens (sometimes hundreds) of megabytes of disk space.

The web had none of these problems, and that is precisely why it had flourished. Web applications were nimble, yet they lacked any capacity for sophistication.

If only there was a middle way, uniting the power of the desktop UI with the nimbleness and visual impact of the web! If only desktop applications could behave like they lived on the web! If only the web, the desktop and the mobile spaces could come together under one unified UI technology! And so the heretics began to cast about for a solution...

1.1.3 Form follows function

From its first release in 1996 Java had featured a powerful technology for deploying rich applications within a web page. So called *Java Applets* could be placed on any page, just like an image, and ran inside a secure environment which prevented unauthorized tampering with the underlying computer or operating system. While applets boosted the visibility of the Java brand, the idea initially met with mixed success. The applet was a hard-core programming technology in a world dominated by artists and designers, and while many page authors drooled over Java's power, few understood how to install one onto their own site, let alone how to create an applet from scratch.



3D-Blox

The sequel to Iceblox, created 1996-12-31. Last modified 1997-07-02.

A pair of alternative left-hand control keys have been implemented for those who are using ergonomic keyboards. D and V now work the same way as G and N. (Suggested by Mike Deem.) This sketch might help those

Figure 1.4 An applet (the game 3D-Blox) runs inside a web page, living along side other web content like text and images.

Java's main rival in the applet space was Macromedia Flash, an animation and presentation tool which boasted a considerably more designer-friendly development experience. Once Macromedia's plugin entered the ring the writing was on the wall for Java applets, and they were quickly defeated by a knockout to leave Flash as the undisputed World (Wide Web) champion. But by then focus of the Java community was already elsewhere; Java had secured an audience on the server side, where it powered blue-chip web sites, and in the mobile market, where it powered phone games and applications. And so applets were forgotten, receiving little or no attention in the updates which followed Java's debut release.

And this is where they might have languished, had the web not taken a collaborative path…

Ten years on (thereabouts) from the death of the applet, and the buzz was once again about on-line applications. Not Java applets this time, but the new breed of Rich Internet Applications which many saw as holding much promise. Yet RIAs were finding it hard to deliver an experience comparable to the desktop, and it wasn't immediately apparent from where a solution might come. Certainly Ajax and HTML were struggling to provide the kind of refined user interaction many now wanted, and Flash's strengths lay more in animation than solid, traditional, 'functional' GUIs and data manipulation.

Was Java about to be given a second chance?

Java had already proved itself as an effective software platform away from the desktop, amassing many followers in the software community and a vast archive of third party libraries in the process. Yet Java still had one major handicap – as far as the desktop was concerned it remained a tool for cola swigging, black t-shirt wearing, code junkies, not trendy cappuccino sipping, goatee stroking, artists. If Java was to be the answer to the RIA dilemma it needed to be less techie and more sympathetic of UI design and aesthetics.

In 2005 Sun Microsystems had acquired a company known as SeeBeyond Technology Corporation, and in the process picked up a talented software engineer by the name of Chris Oliver. Oliver's experimental F3 programming language (Form Follows Function) sought to make GUI programming easier by designing the language around the specific needs of user interface programming. As they pondered how best to exploit the emerging Rich Internet App market, the folks at Sun could surely not have failed to note the potential of combining the existing Java platform and Chris Oliver's new graphics power tool. And so in 2007, at the JavaOne Conference (the community's most important annual gathering), F3 was given center stage as Java's beachhead into the new Rich Internet Application market.

And as if to demonstrate its importance, F3 was blessed with a sexy new name: "*JavaFX*"!

Figure 1.5 The StudioMoto demo, one of the original JavaFX examples, shows off a glossy user interface with animation, movement and rotating elements all responding to the user's interaction.

JavaFX would marry the power of the network-centric Java platform with an efficient graphics-centric language, creating the perfect environment for both programmer and designer to push their skills to the limits on the desktop, the web and on smart phones.

At least, that's the plan...

1.2 JavaFX and the Java platform

JavaFX is not Java, but it rests in a sea of tools and technologies designed and created to support Java. As such it shares the unusual dualistic characteristic of being *of* Java the platform but not Java the language!

Given JavaFX's intentions it's a reasonable expectation to assume a minority of readers may have been drawn to it (and thereby this book) without first having come through Java. They would no doubt benefit from a little scene setting, explaining Java and placing it in context. Meanwhile Java-savy readers will no doubt be keen to hear how the new language and the familiar platform cooperate. So, for wet behind the ears novices we'll take a terse look at the Java platform, and for the salty old Java sea dogs we'll examine how JavaFX fits into the existing Java environment.

1.2.1 How not to go native

The Java is a software platform which seeks to fulfill the mantra "write once, run anywhere". Software is compiled to bytecode files in the form of machine code instructions runnable on a virtual machine called a JVM (Java Virtual Machine). The virtual machine provides a layer of abstraction, allowing the program to be run on many devices without needing to be specifically compiled to the machine code of the underlying hardware.

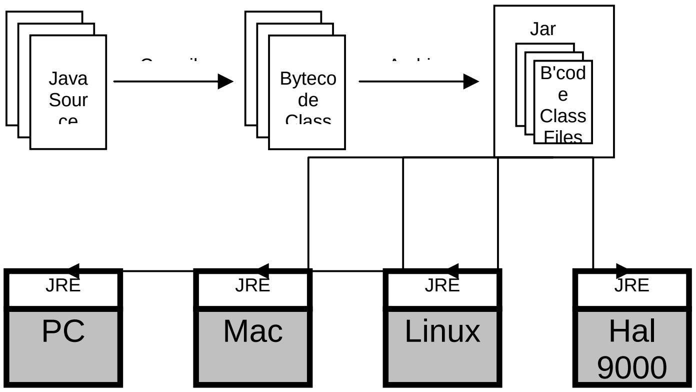


Figure 1.6 From compile-time to run-time: the life cycle of a typical Java application.

Once the source code files are compiled into bytecode class files they can be bundled into a single archive known as a Jar (Java ARchive) for easy distribution. Jars can contain runnable applications, or support libraries. There are numerous ways of getting the software onto the end-user computer, from the traditional (ship it on a CD-ROM) to the modern (embed it in a web page.)

1.2.2 Three editions, two audiences

There are three main Java markets: *Standard Edition* targets regular desktop users, *Micro Edition* is an extension to the Standard Edition adding tools and libraries for small devices like cell phones, and *Enterprise Edition* enhances the Standard Edition for writing web applications and web services.

JavaFX targets both the Standard and Micro editions. (While applets are a web technology, they actually run on the client side. This is why they're part of the Standard Edition, not the Enterprise Edition.)

There exist two basic audiences for Java: the regular user who merely wants to run the software, and the programmer who wants to create the software. These are personified by the *Java Runtime Environment*, and the *Java Development Kit*. The former has the tools necessary to run a Java program and is often shipped pre-installed on desktop computers, while the latter bolsters the JRE with extra tools to create and debug Java software.

JavaFX has its own compiler, and therefore it is theoretically possible to develop JFX software without installing a JDK. However the JDK contains more than just a Java language compiler – it also includes support tools for creating and signing code archives (Jars), profiling your code, and much more. Unless you like writing code with one arm tied behind your back you're strongly recommended to install a full JDK.

1.2.3 A rose by any other name

Now the confusing part for anyone new to Java: release names. Since 1995 Java has been through many revisions. In the beginning it was simple; the first commonly used version of Java was 1.02, which was succeeded by 1.1 shortly thereafter. Someone then apparently decided 1.2 didn't sound important enough, so 'Java' acquired the nom-de-plume 'Java2'. Thus we got *Java2 Standard Edition version 1.2*, more commonly written as *J2SE v1.2*. This schizophrenic naming convention continued until version 1.5, when it was replaced by a new schizophrenic naming convention; 'Java2' reverted to 'Java' once more, and 1.5 became 5.0, however due to technical issues the 1.5 label continued to be used in some places.

(It is left as an exercise for the reader to guess what medication the people who came up with the above may have been on.)

Java SE 6 Update 10 (or later) is the version of the Java platform currently recommended for writing and running JavaFX software. The tenth update overhauls the applet plugin mechanics and provides a much smoother way to get Java onto the end user's machine – both of which are greatly beneficial to JFX. At the time of writing Update 10 is in early access beta only; by the time you read this book it should have made it to a full release.

1.3 Sneak peek: "Hello JavaFX!"

So far we've discussed what JavaFX is, why it's needed, and how it fits into the current Java environment. We've yet to see a single line of JFX code. So let's remedy that by porting an old friend to JavaFX Script:

```
import javafx.ext.swing.*;
import javafx.scene.*;
```

```
Frame
{   title: "Hello World JavaFX"
    content: Label
    {   text: "Hello World!"
        font: Font { size: 30 }
        horizontalAlignment:
            HorizontalAlignment.CENTER
        verticalAlignment:
            VerticalAlignment.CENTER
    }
    width: 400
    height: 100
    visible: true
}
```

The above is a simple JavaFX Script program. Don't panic if you don't understand it yet, it's merely there as an appetizer to give a taste of how JavaFX Script code looks. The program opens a new frame on the desktop with "*Hello World JavaFX*" in the title bar, and the legend "*Hello World!*" as the window contents. Perhaps your untrained eye can already decipher a few clues as to how it works?

```
import javax.swing.*;
class HelloWorldJava
{   public static void main(String[] args)
    {   Runnable r = new Runnable()
        {   public void run()
            {   JLabel l = new JLabel("Hello World!",JLabel.CENTER);
                l.setFont(l.getFont().deriveFont(30f));
                JFrame f = new JFrame("Hello World Java");
                f.getContentPane().add(l);
                f.setSize(400,100);
                f.setVisible(true);
            }
        };
        SwingUtilities.invokeLater(r);
    }
}
```

By way of a comparison the Java equivalent is presented below. It certainly looks busier, although the truth is it's been stripped back to a bare minimum, almost to the point of being crude.The Java version is typical of how GUIs are programmed under popular languages like Java, C++, or BASIC. The frame and the label holding the "*Hello World*" legend are constructed and combined in separate discrete steps. The order of these steps does not necessarily tally with the structure of the user interface they build; above, the label is created before its parent frame is created, but added after.

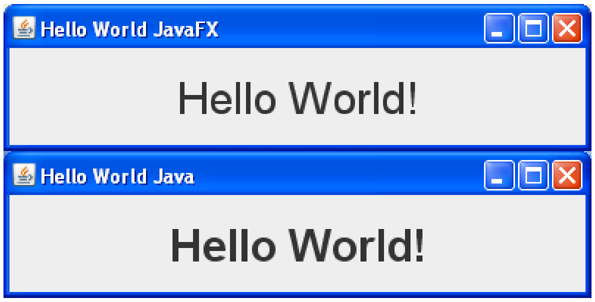


Figure 1.7 Separated at birth... “Hello World!” as a JavaFX and Java application.

As the scale of the GUI increases Java's verbose syntax and disjointed structure (compared to the structure of the GUI) quickly becomes a handful. JavaFX Script, a bit like the Energizer Bunny, can keep on going for far longer thanks to its declarative syntax.

1.4 How do I get what I need?

By now, hopefully, you're just about excited enough to want to dive in and start creating your own software. But you need to know what to install first. Fortunately everything you need is free, and readily available for download off of the internet.

1.4.1 The essentials

This section contains web addresses to software you need to get started. The next section houses links for software, such as the phone toolkits, which you will need if you want to follow the some of the later projects, plus optional extras like Integrated Development Environments.

For each piece of software read the licenses (if any) and requirements carefully, and follow the download and installation instructions.

Existing Java programmers will only need to install the JavaFX SDK itself, assuming you're current Java Development Kit is up to date, plus an optional IDE plugin if required. Non-Java programmers are recommended to install the latest JDK first (as well as an IDE if desired) then install the JavaFX SDK.

`http://java.sun.com/javase/downloads/index.jsp` **`(Java SE Downloads)`**

I'd like the page URLs/titles to stand apart from the text so they're more 'browse-able'. Ideally fixed width font for URLs, reducing likelihood of transcription errors.

Firstly you need to visit the Java Standard Edition downloads page to fetch and install a copy of the Java SE JDK, if you don't already have the latest version. Sun provides SE implementations for Microsoft Windows and Linux, including 64 bit builds. You're recommended to get a Java 6 (or later) JDK, with Update 10 (or later) favored due to a radical enhancement of the platform mechanics regarding applet deployment and start up times.

```
http://developer.apple.com/java/  (Java)
```

The above is the equivalent home page for Apple's Mac OS X Java JDK and JRE implementations.

```
https://openjfx.dev.java.net/  (OpenJFX Home)
http://openjfx.java.sun.com/  (JavaFX Build Server)
```

It should come as no surprise you'll also need to download and install the JavaFX Software Developers Kit. This provides the extra JFX components, such as the compiler, for creating JavaFX software to target the Java SE platform. The first URL above is the home page of the OpenJFX project, which is the Open Source initiative (sponsored by Sun Microsystems) to create and distribute a JavaFX SDK. The second URL leads directly to a menu of links for downloads, documentation, and examples. From here you can get the latest stable and weekly builds of the OpenJFX project.

1.4.2 The extras

The above URLs are essential, while the following are either optional, or not needed until we cover more advanced topics later in the book.

```
http://java.sun.com/javame/downloads/index.jsp  (Java ME Downloads)
```

In later chapters, when we cover phone applications, you will need to install an add-on to Java SE which includes the tools and APIs for addressing mobile devices. The Java Wireless Toolkit (WTK) is available from Sun for Microsoft Windows and Linux.

```
http://code.google.com/android/  (Android)
```

Google has a rival platform for creating mobile software called *Android*, which is also Java based (although it does not use standard Java bytecode.) If you want to try running your mobile JavaFX applications on a Google phone you should consider downloading the Google Android SDK.

```
http://www.netbeans.org/  (Welcome to NetNeans)
http://www.eclipse.org/  (Eclipes.org home)
```

You may want to download an Integrated Development Environment, if you don't have one already. NetBeans amd Eclipse are popular free IDEs for Java, although others are available. IDEs are optional – you may instead opt to work with a regular text editor and the command line tools. This book does not favor one method of working over another.

http://developers.sun.com/downloads/
(Sun Microsystems - Sun Developer Network (SDN))

Various other *official* APIs and Java tools can be found at the Sun Java site. You won't need them for this book, but you may want to experiment with some of them in your own projects. The above page is a handy menu giving quick access to noteworthy tools and APIs.

http://java.sun.com/
(Developer Resources for Java Technology)
http://www.java.net/
(java.net - THe Source for Java Technology Collaboration)

Finally, the above addresses are the Sun Microsystems Java site and the Java community's official Open Source project site. These URLs should prove useful if any of the previous addresses transform themselves into a cursed "*404 Not found*" (as sub page s have wont to do over time.)

1.5 Summary

So, you know all about JavaFX now, why it was created, what it can do, how it fits into the Java platform, and how to get your mitts on the stuff you need to write your own cool JFX apps! What are we waiting for? Let's dive in and start writing coding!

In the next chapter we'll take a whistle stop tour through the JavaFX Script language – it's really quite simple – and in subsequent chapters we'll start making some real mini applications to demonstrate the power of JavaFX.

Are you excited? I certainly hope so! Then let the fun begin...