1 Welcome to the future: introducing JavaFX

This chapter covers

- Reviewing the history of the internet-based application
- Asking what promise DSLs hold for UIs
- Looking at JavaFX Script examples
- Comparing JavaFX to its main rivals

"If the only tool you have is a hammer, you tend to see every problem as a nail," American psychologist Abraham Maslow once observed.

Language advocacy is a popular pastime with many programmers, but what many fail to realize is that programming languages are like tools: each is good at some things and next to useless at others. Java, inspired as it was by prior art like C and Smalltalk, sports a solid general-purpose syntax that gets the job done with the minimum of fuss in the majority of cases. Unfortunately, there will always be those areas that, by their very nature, demand something a little more specialized. Graphics programming has typically been one such area.

Graphics programming used to be fun! Early personal computer software sported predominantly character-based UIs. 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 graphical toolkits used to create desktop software are little evolved from the days of the first Macintosh or Amiga.

But it's not just the look of software that has been changed by the web. 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 online applications using HTML and Ajax have resulted in nothing more than pale imitations of their desktop cousins. At the same time, consumer devices like phones and TV set top boxes are getting increasingly sophisticated in terms of their UI, and faster wireless networks are reaching out to these devices, allowing applications to run in places previously unheard of.

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

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

1.1 Introducing JavaFX

JavaFX is the name of a family of technologies for developing visually rich applications across a variety of devices. Version 1.0 was launched in December 2008, focusing on the desktop and web applets. Version 1.1 arrived a couple of months later, adding phone support to the mix, and by summer 2009 version 1.2 was available, sporting a modern UI toolkit. Later editions promise to expand the platform's reach even further, onto TV devices, Blu-ray disc players, and possibly even personal video recorders, plus further enhance its desktop support with more next-gen UI controls.

The JavaFX APIs have a radically different way of handling graphics, known as *retained mode*, shifting focus away from the pixel-pushing *immediate mode* (à la the Java2D library used by Swing), toward a more structured approach that makes animation cleaner and easier. At JavaFX's center is a major new programming language, *JavaFX Script*, built from the ground up for modeling and animating multimedia applications. JavaFX Script is compiled and object oriented, with a syntax independent of Java but capable of working with Java class files. Together JavaFX Script (the language) and JavaFX (the APIs and tools) create a modern, powerful, and convenient way to create software.

1.1.1 Why do we need JavaFX Script? The power of a DSL

A very good question: why *do* we need yet another language? The world is full of programming languages—wouldn't one of the existing languages do? Perhaps JavaScript, or Python, or Scala? Indeed, what's wrong with Java? Certainly JavaFX Script makes writing slick graphical applications easier, but is there more to it than that?

What makes graphics programming such an ill fit for modern programming languages? There are many problems; ask a dozen experts and you'll get

thirteen answers, but let me (your humble author) risk suggesting a couple of prime suspects:

- UIs generally require quite large nested data structures: trees of elements, each providing a baffling array of configurable options and behaviors. Figure 1.1 demonstrates the hierarchy within a typical desktop application: controls laid out within panels, panels nested within other panels (tabbed panes, for example), ultimately held within windows. Procedural languages like to work in clearly delineated steps, but this linear pattern conflicts with the tree pattern inherent in most GUIs.
- 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. The boilerplate code demanded by many languages to create and manage these animations is verbose and cumbersome.

Perhaps you can think of other problems, but the above two I mentioned (at least in my experience) seem to create more than enough trouble between them. It's deep, fundamental problems like these that a *domain-specific language* can best address.

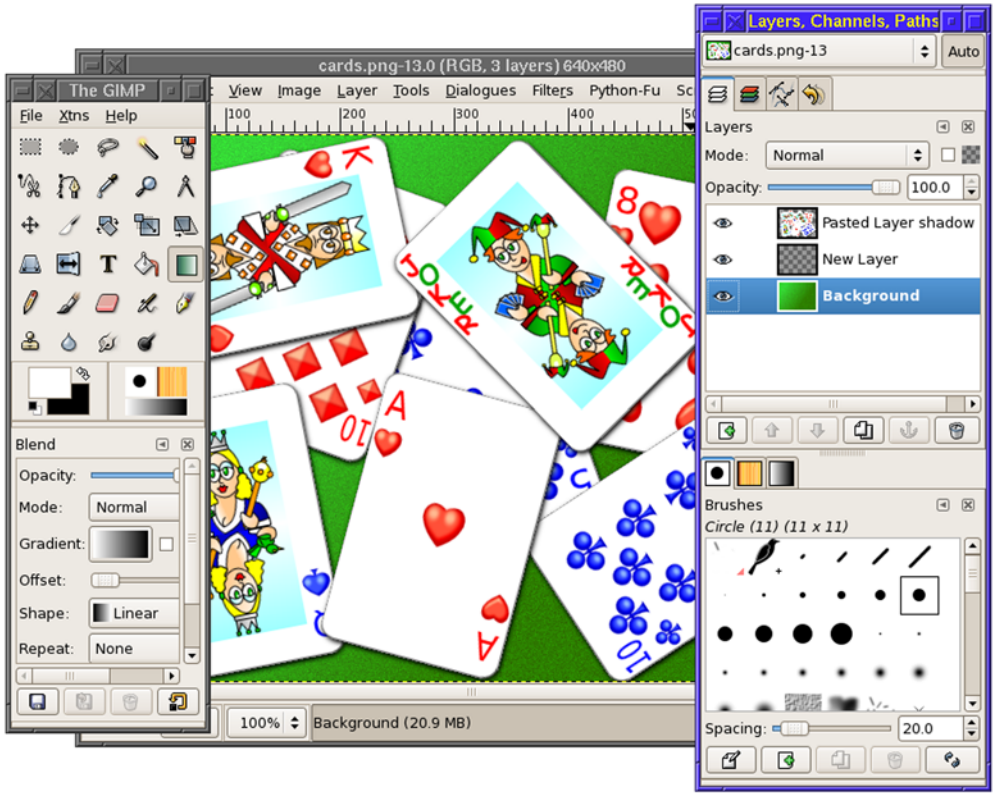

Figure 1.1 A complex GUI typical of modern desktop applications. Two windows host scrolling control palettes, while another holds an editable image and rulers.

A domain-specific language (DSL) is a programming language designed from the ground up to meet a particular set of challenges and solve a specific type of problem. The language at the heart of JavaFX, JavaFX Script, is an innovative DSL for creating visually rich UIs. It boasts a declarative syntax, meaning the code structure mirrors the structure of the interface. Associated UI components are kept in one place, not strewn across multiple locations. Simple language constructs soothe the pain of updating and animating the interface, reducing code complexity while increasing productivity. The language syntax is also heavily expression-based, allowing tight integration between *object models* and the code that controls them.

In layperson's terms, JavaFX Script is a tool custom made for UI programming.

But JavaFX isn't just about slick visuals; it's also an important weapon in the arms race for the emerging Rich Internet Application (RIA) market. But what is an RIA?

1.1.2 Back to the future: the rise of the cloud

Douglas Adams 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."

Sometimes we become so engrossed in the here and now, we forget to stop and consider how we arrived at where we are. We know where we want to go, but can our past better help us get there?

In the pre-internet age, software was installed straight onto the hard drive. If suddenly overcome by an urge to share with friends your latest poetic masterpiece, you needed at your disposal both the document file and the software to open it. Chances are neither would be available. Your friends might be grateful, but clearly this was a problem needing a solution.

The World Wide Web was a small step toward that solution. Initially, *applications* were nothing more than query/response database lookups, but web mail changed all that (figure 1.2). Web mail marked a fundamental shift in the relationship between

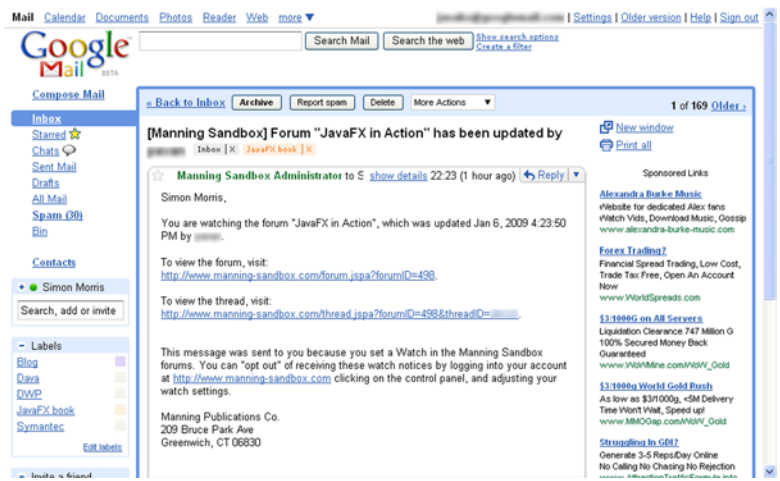

Figure 1.2 Google's Gmail is an example of a website application that attempts to mimic the look and function of a desktop application.

site and visitor. Previously the site held content that the visitor browsed or queried, but web mail sites supplied no content themselves, relying instead on content from (or for) the user. 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.

A new generation of websites attempted to ape the look and feel of traditional desktop software, earning 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. By late 2007 the term *cloud computing* was in common use to describe the anticipated move from the hard disk to the network for storing personal data such as word processor documents, music files, or photos.

Despite the enthusiasm, progress was slow and frustrating. Ajax helped paper over some of the cracks, but at its heart the web was designed to show page-based 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 layout and functionality of a desktop application inside a document-centric environment was not easy, as numerous web developers soon discovered (figure 1.3).

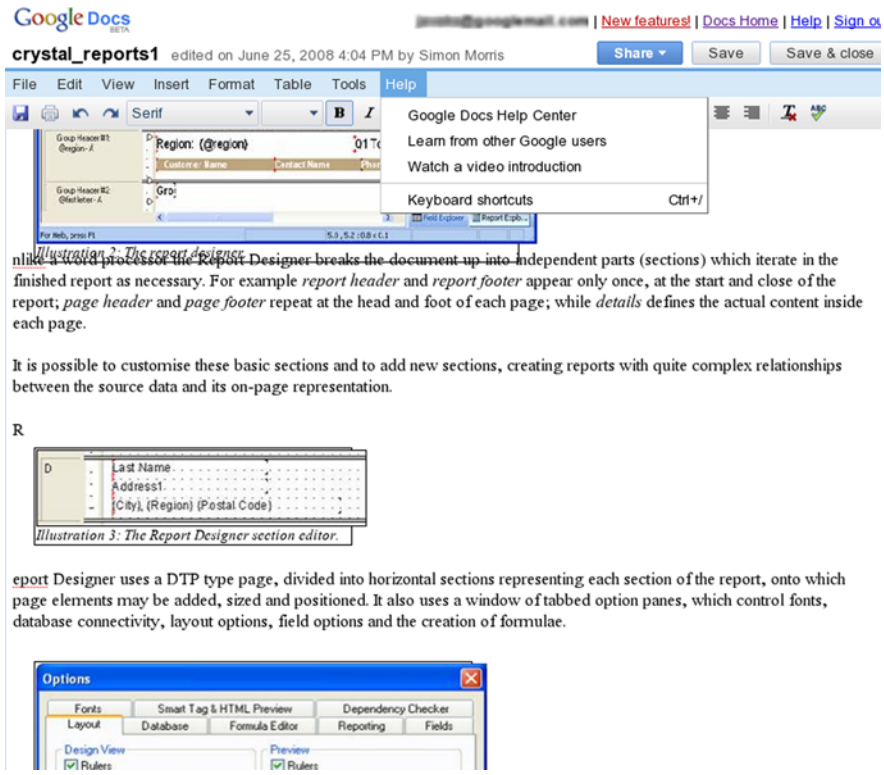

Figure 1.3 Google Docs runs inside a browser and has a much simpler GUI than Microsoft Office or OpenOffice.org. (Google Docs shown.)

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 RIAs. Looking back they saw a wealth of old desktop software with high-fidelity UIs and sophisticated interactivity. But this software used old desktop toolkits, bound firmly to one hardware and OS platform. Web pages could be loaded dynamically from the internet on any type of computer; web RIAs were nimble, yet they lacked any capacity for sophistication.

1.1.3 Form follows function: the fall and rebirth of desktop Java

From its first release in 1995 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 that prevented unauthorized tampering with the underlying operating system (figure 1.4). 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 an applet onto their own site, let alone how to create one from scratch.

Java applet's main rival was Macromedia Flash, an animation and presentation tool boasting a more designer-friendly development experience. Once Macromedia's plug-in began to gain ground, the writing was on the wall for the humble Java applet. Already Sun was starting to ignore user-facing Java in favor of big back-end systems running enterprise web applications. The Java applet vanished almost as quickly as it arrived.

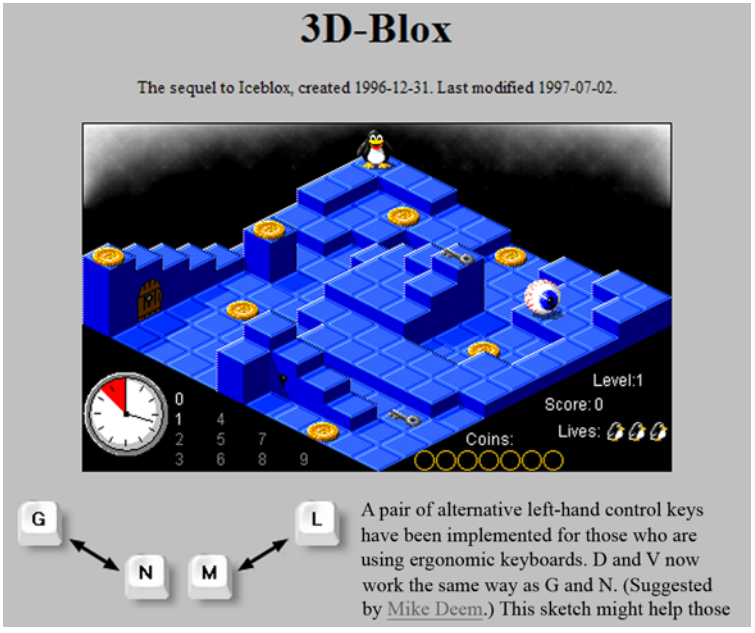

Figure 1.4
An applet (the game 3D-Blox) runs inside a web page, living alongside other web content like text and images.

Fast forward 10 years and the buzz was once again about online applications: RIAs and cloud computing. Yet Ajax and HTML were struggling to provide the kind of refined UI many now wanted, and Flash's strengths lay more in animation than solid *functional* GUIs and data manipulation.

Could Java be given a second chance?

Java had proved itself in the enterprise space, amassing many followers in the software community and a vast archive of third-party libraries. Yet Java still had one major handicap—on the desktop 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 more Leonardo da Vinci and less Bill Gates (figure 1.5).

In 2005 Sun Microsystems acquired SeeBeyond Technology Corporation, and in the process it picked up a talented software engineer by the name of Chris Oliver. Oliver's experimental F3 (Form Follows Function) programming language sought to make GUI programming easier by designing the syntax around the specific needs of UI programming. As they pondered how best to exploit the emerging RIA market, the folks at Sun could surely not have failed to note the potential of combining the existing Java platform with Oliver's new graphics power tool. 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 RIA 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 UI with animation, movement, and rotating elements all responding to the user's interaction.

1.2 Minimum effort, maximum impact: a quick shot of JavaFX

It's hard to visualize the difference a new technology will make to your working life from a description alone. What's often needed is a short but powerful example of what's possible. A picture is worth a thousand words, and so in lieu of a few pages of text, I give you figure 1.6.

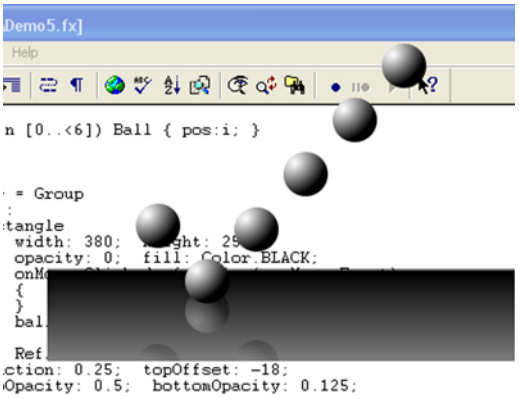

Figure 1.6 The bouncing balls demo, with color shading, reflection effect, and a shaped window (that's a text editor behind, with source code loaded, demonstrating the app's transparency).

Six shaded balls bounce smoothly up and down onto a reflective shaded surface, as the desktop is exposed behind the balls. The window has no title bar (the title bar you see belongs to the text editor behind), but clicking inside its boundary will close the window and exit the bouncing ball application.

Now, the sixty-four-thousand-dollar question: how many lines of code does it take to construct an application like this? Consider what's involved: multiple objects moving independently, circular shading on each ball, linear shading on the ground, a reflection effect, transparency against the desktop, and a click event handler. If you said "less than 70," then you'd be right! Indeed, the whole source file is only 1.4k in size and weighs in at a mere 69 lines. Don't believe me? Take a look at listing 1.1.

Listing 1.1 The bouncing ball demo

```
import javafx.animation.*;
import javafx.stage.*;
import javafx.scene.*;
import javafx.scene.effect.*;
import javafx.scene.shape.*;
import javafx.scene.input.*;
import javafx.scene.paint.*;

var balls = for(i in [0..<6]) {
    var c = Circle {
        translateX: (i*40)+90;  translateY: 30;
        radius: 18;
        fill: RadialGradient {
            focusX: 0.25;  focusY:0.25;
            proportional: true;
            stops: [
                Stop { offset: 0; color: Color.WHITE; } ,
                Stop { offset: 1; color: Color.BLACK; }
            ]
        };
    }
}
```

```
Stage {
    scene: Scene {
        content: Group {
            content: [
                Rectangle {
                    width: 380;  height: 250;
                    opacity: 0.01;
                    onMouseClicked:
                        function(ev:MouseEvent) { FX.exit(); }
                } , balls
            ]
            effect: Reflection {
                fraction: 0.25;  topOffset: -18;
                topOpacity: 0.5;  bottomOpacity: 0.125;
            }
        }
        fill: LinearGradient {
            endX: 0;  endY: 1;  proportional: true;
            stops: [
                Stop { offset: 0.74;  color: Color.TRANSPARENT; } ,
                Stop { offset: 0.75;  color: Color.BLACK } ,
                Stop { offset: 1;  color: Color.GRAY }
            ]
        }
    };
    style: StageStyle.TRANSPARENT
};

Timeline {
    keyFrames: for(i in [0..<sizeof balls]) {
        KeyFrame {
            time: i*200ms;
            action: function() {
                Timeline {
                    repeatCount: Timeline.INDEFINITE;
                    autoReverse: true;
                    keyFrames: [
                        at (0s) { balls[i].translateY => 30 } ,
                        at (1s) { balls[i].translateY => 230
                            tween Interpolator.EASEIN }
                    ]
                }.play();
            }
        }
    }
}.play();
```

Since this is an introductory chapter, I'm not going to go into detail about how each part of the code works. Besides, by the time you've finished this book you won't need an explanation of its mysteries, because you'll already be writing cool demos of your own. Suffice to say although the code may look cryptic now, it's all pretty straightforward once you know the few simple rules that govern the language syntax. Make a mental note, if you want, to check back with listing 1.1 as you read the first half of this book; you'll be surprised at how quickly its secrets are revealed.

1.3 Comparing Java and JavaFX Script: "Hello JavaFX!"

So far we've discussed what JavaFX is and why it's needed. We've looked at an example of JavaFX Script and seen that it's very different from Java, but just how different? For a true side-by-side comparison to demonstrate the benefits of JavaFX Script over Java, we need to code the same program in both languages. Listings 1.2 and 1.3 do just that, and figure 1.7 compares them visually.

Listing 1.2 Hello World as JavaFX Script

```
import javafx.scene.Scene;
import javafx.scene.text.*;
import javafx.stage.Stage;
Stage {
    title: "Hello World JavaFX"
    scene: Scene {
        content: Text {
            content: "Hello World!"
            font: Font { size: 30 }
            layoutX: 114
            layoutY: 45
        }
    }
    width:400  height:100
}
```

Listing 1.2 is a simple JavaFX Script program. Don't panic if you don't understand it yet–this isn't a tutorial; we're merely contrasting the two languages. 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 you can already decipher a few clues as to how it works.

Listing 1.3 Hello World as Java

```
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);
    }
}
```

The Java equivalent is presented in listing 1.3. It certainly looks busier, although actually it has been stripped back, almost to the point of becoming crude. The Java code is typical of GUIs 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 UI they build; the label is created before its parent frame is created but added after.

Figure 1.7 Separated at birth: "Hello World!" as a JavaFX application and as a Java application

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

For readers unfamiliar with the Java platform, appendix D provides an overview, including how the "write once, run anywhere" promise is achieved, the different editions of Java, and the versions and revision names over the years. Although JavaFX Script is independent of Java as a language, it's reliance on the Java runtime platform means background knowledge of Java is useful.

1.4 Comparing JavaFX with Adobe AIR, GWT, and Silverlight

JavaFX is not the only technology competing to become king of the RIA space: Adobe, Google, and Microsoft are all chasing the prize too. But how do their offerings compare to JavaFX? Now that we've explored some of the concepts behind JavaFX, we're in a better position to contrast the platform against its alleged rivals.

Comparing technologies is always fraught with danger. Each technology is a multifaceted beast, and it's impossible to sum up all the nuanced arguments in just a few paragraphs. Readers are encouraged to seek second opinions in deciding which technology to adopt.

1.4.1 Adobe AIR and Flex

Flex is a toolkit adding application-centric features to Flash movies, making it easier to write serious web apps alongside games and animations. AIR (Adobe Integrated Runtime, originally codenamed Apollo) is an attempt to allow Flex web applications to become first-class citizens on the desktop. AIR programs can be installed just like regular desktop programs on a PC, Mac, or Linux computer, assuming the appropriate AIR runtime has been installed beforehand. Using WebKit (the open source HTML/JavaScript component), AIR provides a web-page-like shell in which HTML, JavaScript, Flex, Flash, and PDF content can interact. AIR has made it possible to transfer web programming skills directly onto the desktop, and Adobe plans to extend this concept to allow AIR programmers to target mobile devices as well.

1.4.2 Google Web Toolkit

Google Web Toolkit (GWT) is an open source attempt to smooth over the bumps in HTML/Ajax application development with a consistent cross-browser JavaScript library of desktop-inspired widgets and functions. It's said that GWT started as an internal

Google project to help write sites like Gmail and Google Calendar (although which Google sites actually use GWT is unknown). GWT applications are coded in Java, compiled to JavaScript, and run entirely within the web browser. They can make use of optionally installed plug-ins, such as Gears, to provide offline support.

1.4.3 Microsoft Silverlight

With Silverlight, Microsoft is seeking to shift its desktop software prowess inside the browser. Silverlight is a proprietary browser plug-in for recent editions of Windows and Mac OS X. Linux is also covered via an open source project and a deal with Novell (licensing difficulties may exist for non-Novell Linux customers). Silverlight supports rich vector-based UIs, coded in .NET languages (like C#) and a UI markup language called XAML (Extensible Application Markup Language). Microsoft worked hard to create a fluid video/multimedia environment, with solid support for all the formats supported by its Windows Media framework.

1.4.4 And by comparison, JavaFX

While other RIA technologies blur the line between desktop and browser, JavaFX removes the distinction entirely. A single JavaFX application can move seamlessly (quite literally, by being dragged from the browser window) from one environment to the other. Desktop, applets, and smart phones can already be targeted, while Blu-ray and other TV devices are expected to join this list at a later date. With a common core across all environments, complemented by device-specific extensions, JavaFX lets us target every device or exploit the full power of a particular device.

While other RIA technologies recycle existing languages, JavaFX Script is built from the ground up specifically for creating sophisticated UIs and animation. Studying common working methods found in UI software, the JavaFX team created a language around those patterns. The declarative syntax permits code and structure to be interwoven with a degree of ease not found in the bilingual approach of its rivals. Direct relationships can be defined between an object and the data or functions it depends on; the heavy lifting of model/view/controller is done for you. And because JavaFX Script is compatible with Java classes, it has access to over a decade of libraries and open source projects.

It's true that the need to learn a new language may discourage some, but the reward is a much more powerful tool, shaped specifically for the job at hand. Picking the best tool can often mean the difference between success and failure, while holding onto our familiar tools for too long can sometimes put us at a disadvantage. The skill is in knowing when to embrace a new technology, and hopefully this section has helped clarify whether JavaFX is the right technology for you!

1.5 But why should I buy this book?

Good question—indeed, why buy a book at all? The APIs are documented online, and there are blogs aplenty guiding coders through that tricky first application.

This book specifically seeks not to regurgitate existing documentation, like so many programming tomes tend to do. You won't find laborious enumerations of every variation of every shade of every nuance of every class. This book assumes you're intelligent enough to read the documentation for yourself, once pointed in the right direction; you don't need it reprinted here. So what *is* in this book?

The early chapters give a quick and entertaining (yet comprehensive) guide to the JavaFX Script language; then it's straight into the projects! Each project chapter houses a self-contained miniapplication requiring specific skills and works from initial goals toward a solution in JavaFX. Successive projects reinforce acquired skills and add new ones. Concepts are demonstrated and explained in real-world scenarios; it's an approach centered on common practices, solutions, and patterns, rather than merely ticking off every variation of, for example, a scene graph node or animated transition included in the API.

The code in each chapter seeks to be ideas-rich but compact and fresh. What's the point of page upon page of stuff the reader already saw in previous chapters? Although functional, each completed project leaves room for readers to experiment further, practicing newfound skills by adding features or polishing the UI with extra color blends and animations.

For better or worse, the text attempts to remain agnostic of any particular IDE or tool, other than those shipped with the standard JavaFX SDK. Illustrated click-by-click guides for each IDE would be page hungry and offer little over the online tutorials already provided with (or for) each plug-in. Again, it's about complementing available documentation, not reproducing it, leaving more room for JavaFX examples and advice, not IDE-specific tutorials. (This is, after all, *JavaFX in Action* not *NetBeans in Action*!) Relevant plug-in/IDE links are provided in the appendices.

So, is this book for you? If you're merely looking for a hard copy of the API documentation, perhaps not. But if you want something that goes deeper, exploring JavaFX through *real-world* code, solving *real-world* problems, I hope you'll find what you're looking for in the pages to come.

1.6 Summary

This chapter has been an introduction to the world of JavaFX and JavaFX Script. We started by considering the power of domain-specific languages, designed specifically to meet the needs of particular tasks. Then we considered the rise of the RIA and the challenges in developing such applications using current browser-based technologies. We revisited Java's disappointing track record on the desktop, particularly with lightweight internet applications like applets, but saw how this could change with the introduction of JavaFX to address a new generation of internet applications. We saw an example of JavaFX Script doing modestly impressive things in only a few dozen lines of code, and we reviewed side-by-side the differences in styles and size of Java and JavaFX Script source code. Finally, we considered how JavaFX stacks up against the apparent opposition.

I hope this has been enough to grab your attention and fire your imagination, because in the next chapter we leave the theory behind and dive straight into the detail.

Over the next couple of chapters we'll tour the JavaFX Script language, with, I hope, plenty of nice surprises along the way. This will get us ready to tackle subsequent chapters, where we use practical miniprojects to demonstrate different aspects of JavaFX. (For those expert Java programmers who would prefer more of a whistle-stop tour of the new language, appendix B acts as both a flash-card tutorial and an aide-mémoire).

Before we move on, you will almost certainly want to take a detour to appendix A, which acts as a setup guide for downloading and installing JavaFX, plus getting your code to build. It also features some very useful JavaFX links for help and further reading. Also, if you're not a Java programmer, let me draw your attention to the crash course in object-oriented programming in appendix C and the introduction to the Java platform (and how JavaFX fits into it) in appendix D.

So that's the introduction out of the way. Are you excited? Well, I certainly hope so! Let the fun begin.