



Kim Topley

Updated for
JavaFX 1.3

JavaFX™

Developer's Guide

Developer's Library



JavaFX™ Developer's Guide

Copyright © 2011 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Library of Congress Cataloging-in-Publication Data

Topley, Kim.

JavaFX developer's guide / Kim Topley.
p. cm.

Includes bibliographical references and index.

ISBN 978-0-321-60165-0 (pbk. : alk. paper) 1. Java (Computer program language) 2. JavaFX (Electronic resource) 3. Graphical user interfaces (Computer systems) 4. Application software—Development. 5. Internet programming. I. Title.

QA76.73.J38T693 2010
005.13'3—dc22

2010010696

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.

First Printing, October 2010

ISBN 13: 978-0-321-60165-0

ISBN 10: 0-321-60165-3

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Pearson offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact U.S. Corporate and Government Sales, 1-800-382-3419, corpsales@pearsontechgroup.com.

For sales outside of the United States, please contact: International Sales, international@pearsoned.com.

Visit us on the web: informit.com/aw

For information regarding permissions, write to: Pearson Education, Inc., Rights and Contracts Department, 501 Boylston Street, Suite 900, Boston, MA, 02116;
Fax: (617) 671-3447.

Acquisitions Editor

Greg Doench

Development Editor

Michael Thurston

Managing Editor

John Fuller

Full-Service Production Manager

Julie B. Nahil

Copy Editor

Keith Cline

Indexer

Rebecca Salerno

Proofreader

Apostrophe
Editing Services

Publishing Coordinator

Michelle Housley

Book Designer

Gary Adair

Composition

Jake McFarland

Contents at a Glance

About the Author **xxv**

Preface **xxvi**

I: Introduction to JavaFX

- 1** An Overview of JavaFX **3**
- 2** JavaFX Script Basics **17**
- 3** JavaFX Script Development **33**
- 4** A Simple JavaFX Application **45**

II: The JavaFX Script Language

- 5** Variables and Data Types **89**
- 6** Expressions, Functions, and Object Literals **121**
- 7** Sequences **153**
- 8** Controlling Program Flow **179**
- 9** Binding **195**
- 10** Triggers **221**
- 11** JavaFX Script Classes **239**
- 12** Platform APIs **285**
- 13** Reflection **309**

III: User Interfaces with JavaFX

- 14** User Interface Basics **341**
- 15** Node Variables and Events **375**
- 16** Shapes, Text, and Images **433**
- 17** Coordinates, Transforms, and Layout **503**
- 18** Animation **591**
- 19** Video and Audio **627**

- 20** Effects and Blending **651**
- 21** Importing Graphics **703**
- 22** Cross-Platform Controls **737**
- 23** Style Sheets **811**
- 24** Using Swing Controls **829**
- 25** Building Custom Controls **865**
- 26** Charts **911**

IV: Miscellaneous

- 27** Using External Data Sources **949**
- 28** Packaging and Deployment **1025**
 - A** Using JavaFX Command-Line Tools **1049**
 - B** CSS Properties **1061**
 - Index **1071**

Table of Contents

About the Author xxv

Preface xxvi

I: Introduction to JavaFX

1 An Overview of JavaFX 3

The JavaFX Platform 3

The JavaFX Script Language 5

Variable Declarations 6

Access to Java APIs 6

Object Literals 7

Binding 8

Scripts and Compilation 8

The JavaFX Runtime 9

User Interface Classes 9

Video and Audio 10

Animation 10

Network Access 12

JavaFX Development Tools 13

Deployment 14

Java Platform Dependencies and Installation 14

The Java Plug-In 15

Converting an Applet to an Installed Application 15

2 JavaFX Script Basics 17

Source File Structure 17

Comments 18

Single-Line Comments 18

Multiline Comments 18

Documentation Comments 19

The package Statement 20

The import Statement 20

Import by Name 20

Wildcard Imports 21

Static Imports 21

Automatic Imports 22

Direct Class References	23
Other JavaFX Statements	23
Variable Declarations	23
Value Assignment and Data Manipulation	23
Using Java Methods	24
Binding Expressions	24
Functions	25
Flow-of-Control Statements	26
Class Definitions	26
Triggers	26
JavaFX Keywords and Reserved Words	27
Script Execution and Arguments	28
Predefined Variables	31

3 JavaFX Script Development 33

Compiling and Running JavaFX Code	33
Development Using the NetBeans IDE	34
Development with the Eclipse IDE	39
Documentation in JavaFX Source Code	43
Viewing JavaFX Documentation in NetBeans	43
Viewing JavaFX Documentation in Eclipse	44

4 A Simple JavaFX Application 45

Building the SnowStorm Application	46
Creating the SnowStorm Project	46
Building the User Interface	47
Adding the Animation	58
Counting the Snowflakes	64
SnowStorm on the Web, a Phone, and TV	65
Running SnowStorm Using Java Web Start	65
Running SnowStorm as an Applet	67
Running SnowStorm on the Mobile Emulator	70
Running SnowStorm on the JavaFX TV Emulator	72
Debugging the SnowStorm Application	72
Setting Breakpoints	72
The Call Stack View	73
Inspecting Variables	73

Changing Variable Values	74
Stepping Through Code	75
Disabling and Removing Breakpoints and Resuming Execution	76
Profiling the SnowStorm Application	77
Using the NetBeans Profiler	77
Source Code for the SnowStorm Application	82

II: The JavaFX Script Language

5 Variables and Data Types 89

Variable Declarations	89
Declaring Variables with var	89
Declaring Variables with def	92
Variable Scopes	93
Object Types	94
Creating a JavaFX Object Instance	95
Invoking JavaFX Functions	95
Accessing Variables	96
Using Java Classes in JavaFX Code	97
Basic Data Types	97
Numeric Types	97
The Boolean Type	102
The String Type	103
String Localization	108
Type Inference	117
Visibility of Variables	119

6 Expressions, Functions, and Object Literals 121

Expressions and Operations	121
Numeric Operations	123
Boolean Operations	129
Object and Class Operations	130
JavaFX Functions	134
Declaring Functions	134
Functions and Variables	137
Functions Within Functions	138

Invoking JavaFX Functions	138
Invoking Java Methods	138
Function Variables	139
Anonymous Functions	145
Visibility of Functions	147
Object Literals	147
Initializing a JavaFX Object	148
Variables in Object Literals	149
Functions in Object Literals	150

7 Sequences 153

Sequence Creation	153
The String Form of a Sequence	155
Range Notation	156
Sequence Equality and Copying	157
Querying Sequences	158
Obtaining the Size of a Sequence	158
Obtaining an Element of a Sequence	159
Obtaining Part of a Sequence	160
Querying a Sequence by Condition	160
Modifying Sequences	162
Replacing Elements	162
Inserting Elements	163
Removing Elements	165
Replacing a Range of Elements	166
Operations on Sequences	167
Comparing Sequences	167
Searching for Elements	168
Finding the Largest and Smallest Elements	169
Sorting a Sequence	171
Searching and Updating a Sorted Sequence	172
Shuffling a Sequence	174
Java Arrays	174
Array Variable Declaration	174
Array Variable Initialization	175
Array Operations	176
Array Size	177

8 Controlling Program Flow 179

- The if Statement 179
- The while Statement 181
 - The break Statement 182
 - The continue statement 184
- The for Statement 184
 - Iterating over a Sequence 184
 - The for Statement as an Expression 186
 - Iterating over a Subset of a Sequence 187
 - Iterating over Multiple Sequences 188
 - Iterating over an Iterable 190
 - Iterating over an Array 193
- Exception Handling 193

9 Binding 195

- Binding to Variables and Expressions 195
 - Binding to a Script Variable 195
 - Binding in an Object Literal 196
 - Binding to an Expression 199
 - Binding and the def Statement 201
 - Binding to an Instance Variable 201
 - Binding with a Conditional Expression 202
 - Bidirectional Binding 203
 - Eager and Lazy Binding 206
- Binding and Functions 207
 - Binding and Unbound Functions 207
 - Binding and Bound Functions 209
 - Optimization of Bound Function Evaluation 212
 - Content of a Bound Function 213
- Binding and Sequences 217
 - Binding to a Sequence 217
 - Binding to a Transformed Sequence 218
 - Binding to the Size of a Sequence 219
 - Binding to a Range of Sequence Elements 219

10 Triggers 221

- Triggers on Simple Variables 221
 - Declaring a Trigger 221
 - Getting the Previous Value 223
 - Triggers and Binding 223
 - Using a Trigger to Enforce Constraints 224
- Triggers and Sequences 229
 - Replacing Elements in a Sequence 230
 - Removing Elements from a Sequence 231
 - Inserting Elements into a Sequence 233
 - Example Use of a Sequence Trigger 234
- Triggers and Instance Variables 236

11 JavaFX Script Classes 239

- JavaFX Class Declaration 240
 - An Example JavaFX Class 241
 - Class Visibility 242
 - Instance Variables 243
 - Instance Functions 246
- Subclassing and Abstract Classes 249
 - An Abstract Base Class 249
 - Extending the Base Class 251
 - Function Overloading 253
 - Function Overriding 254
 - Function Selection by Classname 258
- Using Bound Functions 258
- Variable Overrides 259
- Class Initialization 261
 - Initialization Order 261
 - Using the init and postinit Blocks 263
- Classes and Script Files 265
- Mixins 266
 - Implementing Logging with a Mixin 267
 - Mixin Characteristics 272
 - Mixins and Inheritance 273
 - Mixins and Triggers 280
 - Initialization and Mixins 281

12 Platform APIs 285

- Built-In Functions 285
 - Writing to the System Output Stream 286
 - Object Comparison 286
- Arguments and System Properties 287
 - Application Arguments 287
 - Named Arguments 288
 - System Properties 290
- Application Shutdown 292
- Deferring Operations 294
- Functions for Internationalization 295
 - Changing String Localization Defaults 296
- Local Storage 298
 - Reading and Writing Data 299
 - Storage Metadata 301
 - Removing Data 303
 - Resource Names and Access Control 305
- Conditional Features 307

13 Reflection 309

- Context and Classes 309
 - The Reflection Context 309
 - Reflective Access to Classes 310
- Types and Values 314
 - Representation of Types 314
 - Values 317
- Variables and Functions 320
 - Reflecting on Variables 322
 - Reflecting on Functions 323
 - Filtered Searches 326
 - Reflecting on Variables and Functions 327
- Using Reflection 328
 - Creating Class Instances: Part 1 328
 - Reading and Setting Variable Values 328
 - Invoking Functions 330
 - Creating and Accessing Sequences 333
 - Creating Class Instances: Part 2 336

III: User Interfaces with JavaFX

14 User Interface Basics 341

- The Stage Class 342
 - Stage Appearance and State 342
 - Stage Position and Size 348
 - Style and Opacity 354
 - Extensions 357
- The Scene Class 358
- Nodes 360
 - Node Organization 361
 - Events 367
 - Colors 367
 - Effects 368
- Alerts 369
 - Information Alert 369
 - Confirm Alert 370
 - Question Alert 371
- 3D Features 372
 - Cameras and the Z-Axis of the Scene Graph 372

15 Node Variables and Events 375

- Cursors 375
- Colors 378
 - Solid Colors 379
 - Linear Gradients 383
 - Radial Gradients 392
- Events 401
 - Mouse Events 401
 - The Mouse and the MouseEvent Class 403
 - Keyboard Events 425

16 Shapes, Text, and Images 433

- Shapes 433
 - Basic Shapes 434
 - Paths 451
 - SVGPath 456
- Stroking and Filling Shapes 456

Stroking Shapes	456
Shape Filling	463
The Text Class	466
Text Content and Positioning	466
Text Fill and Stroke	470
Text Decoration	471
Fonts	472
Font Characteristics	473
Physical and Logical Fonts	476
The Font Class	477
Listing Fonts and Font Families	479
Selecting Fonts	479
Groups and Custom Nodes	482
Images	486
Loading an Image	486
Displaying an Image	497
17 Coordinates, Transforms, and Layout	503
Transforms	503
Translation	506
Rotation	508
Scaling	511
Shearing	515
Combining Transforms	517
Transform Order	517
Combining Transforms and Node Variable Settings	521
Clipping	523
Coordinates and Bounds	527
Getting Node Bounds	527
Coordinate System Conversions	537
Node Layout	538
Node Sizes and Layout	539
The Flow Container	547
The Stack Container	555
The HBox and VBox Containers	559

The Tile Container	563
The Panel Container	572
The ClipView Node	574
Centering Nodes	577
SnowStorm Revisited	579
Placing the Background Image	581
Placing and Animating the Snow	582
Coordinates and Screens	585
Discovering Screens	585
Using Two Screens	588
Changing Screen Arrangement	589

18 Animation 591

Timelines	591
Time Literals and the Duration Class	595
Key Frames	596
Interpolation and Interpolators	600
Controlling a Timeline	605
Repeating an Animation	605
Automatically Reversing an Animation	607
Pausing, Stopping, and Restarting an Animation	608
Changing the Speed and Direction of an Animation	609
Starting a Timeline from the End	610
Using a Timeline as a Timer	611
Animation Length	613
Transitions	613
The Transition Class	613
TranslateTransition	614
RotateTransition	616
ScaleTransition	617
FadeTransition	618
PathTransition	619
PauseTransition	622
Sequential and Parallel Transitions	622

19 Video and Audio 627

- The Media Class 628
- The MediaPlayer Class 630
 - Controlling Media Playback 630
 - Restricting and Repeating Playback 632
 - Volume Control 633
 - Monitoring Playback 634
 - Synchronizing External Events with Playback 637
- The MediaView Class 639
 - Size and Position of the Video Frame 640
 - The Viewport 644
 - Transforms and Effects 646

20 Effects and Blending 651

- Effects Overview 651
 - Effects Chains 651
 - Effects and Nodes 652
 - Effects and Groups 655
- The JavaFX Effects Classes 656
 - GaussianBlur 656
 - BoxBlur 657
 - MotionBlur 659
 - DropShadow 660
 - InnerShadow 663
 - Shadow 664
 - Bloom 665
 - Glow 666
 - Identity 667
 - Flood 669
 - ColorAdjust 669
 - InvertMask 671
 - Reflection 671
 - SepiaTone 673
 - PerspectiveTransform 673
 - DisplacementMap 679

Blending	686
The Blend Effect	686
The Group Blend Mode	688
Lighting	690
The surfaceScale Variable	692
The Bump Map	693
DistantLight	694
PointLight	697
SpotLight	698

21 Importing Graphics 703

The JavaFX Production Suite	703
Using Adobe Illustrator and Photoshop Graphics	705
Exporting Graphics from Adobe Illustrator	705
Previewing the JavaFX Format Files	708
Loading Graphics into a JavaFX Application	710
Specifying Animation Paths with Illustrator	717
Embedding Fonts	720
Embedding Images	724
Using a Stub Node	725
Creating Multiple Copies of a Graphics Element	731
Importing SVG Graphics	733

22 Cross-Platform Controls 737

Controls Overview	737
The Label Control	738
Label and the Labeled Class	739
Basic Labels	741
Positioning of Text and Graphics	742
Multiline Text	746
Text Overruns and Wrapping	747
Button Controls	749
The Button Class	749
The Hyperlink Class	752
The ToggleButton, RadioButton, and CheckBox Classes	756

The TextBox Control	761
TextBox Width	764
TextBox Height	764
Editability	764
Setting and Getting the Content of the TextBox	766
Selection	769
Copy and Paste	771
The PasswordBox Control	771
The ListView Control	773
Creating a ListView	773
ListView Selection	778
ListView Cell Rendering	782
The ChoiceBox Control	786
The ScrollBar Control	787
ScrollBar Value and Range	789
User Gestures	789
Using the ScrollBar Control	790
The ScrollView Control	794
Scrollbar Display and Values	797
Scrollable Node Size	797
The Slider Control	797
Basic Slider Operation	798
Tick Marks	800
The ProgressIndicator and ProgressBar Controls	804
Using the ProgressBar and ProgressIndicator Controls	804
The Separator Control	807
Tooltips	808

23 Style Sheets 811

Style Sheet Basics	811
Using a Style Sheet	812
Style Sheet Structure	813
Selection by ID	814
Style Sheet Property Specifications	824
Fonts	824

Paints 826

Effects 828

24 Using Swing Controls 829

Swing Component Wrappers 829

SwingComponent Variables 830

SwingComponent as a Node 830

Accessing the Wrapped Swing Component 832

Labels 833

Text and Icon 834

Positioning the Content of SwingLabel 836

Text Input 839

Configuring the SwingTextField Control 840

Handling Input 842

Buttons 843

The SwingAbstractButton and SwingButton
Classes 843

Toggle Buttons 846

Radio Buttons and Check Boxes 848

The SwingList Class 849

Creating a SwingList Control 850

Handling the Selection 852

The SwingScrollPane Class 852

The SwingComboBox Class 854

Using a Noneditable SwingComboBox 855

Using an Editable SwingComboBox 856

The SwingSlider Class 857

Using Other Swing Components 860

Using a Generic JavaFX Wrapper 860

Creating a JavaFX Wrapper Class 862

25 Building Custom Controls 865

Custom Nodes 865

The CoordinateGrid Node 866

Custom Containers 869

A Border Container 869

Using the Panel Class 884

Custom Controls	887
Custom Control Architecture	887
Control Appearance and Styling	890
A Skeleton Custom Control	890
A Media Player Control Bar	895

26 Charts 911

Chart Basics	911
Chart Components	912
Chart Data	914
Chart Interaction	914
Pie Charts	914
Creating a Pie Chart	914
A 3D Pie Chart	916
Customizing a Pie Chart	917
Bar Charts	919
Creating a Bar Chart	919
A 3D Bar Chart	922
Bar Chart Customization	922
Line Charts	926
Creating a Line Chart	926
Line Chart Customization	928
Area Charts	930
Scatter Charts	932
Bubble Charts	934
Chart Interaction	936
Common Customizations	937
Chart Customization	937
XY Chart Customization	939
Axis Customization	941

IV: Miscellaneous

27 Using External Data Sources 949

The HttpRequest Class	950
Basic Use of the HttpRequest Class	950
Lifecycle of an HTTP Request	952
GET Requests	955

PUT Requests	965
POST Requests	971
DELETE Requests	973
Using RESTful Web Services	974
Parsing XML	975
A Twitter Web Service Client	983
A JSON Web Service Client	988
RSS and Atom Feeds	995
Feeds Overview	996
RSS Feeds	997
Atom Feeds	1004
Tasks and Progress Monitoring	1008
Task State Variables	1009
Progress Monitoring	1010
A State Monitoring Example	1011
Asynchronous Operations and Database Access	1014
A Database Access Task	1015
Implementing the Database Access Task	1018

28 Packaging and Deployment 1025

Packaging and Deployment for the Desktop	1026
Creating a Packaged Application and Applet with <code>javafxpackager</code>	1026
Application Deployment	1029
Applet Deployment	1033
Setting and Reading Parameters	1037
Incorporating Libraries	1039
Compressing the JAR Files	1042
Signing Applications and Applets	1043
Packaging and Deployment for Mobile Devices	1045
Creating a Packaged Mobile Application	1046
Deployment	1047

A Using JavaFX Command-Line Tools 1049

Development Using Command-Line Tools	1049
Compiling a JavaFX Application	1050
Running a JavaFX Application	1051
Development Using an Ant Script	1052
Generating Documentation from JavaFX Source	1055

B CSS Properties 1061

Properties Applicable to Nodes	1061
Group Properties	1062
ImageView Properties	1062
Text Properties	1062
Properties Applicable to Shapes	1063
ClipView Properties	1064
Rectangle Properties	1064
Properties Applicable to Containers	1064
Flow Properties	1065
HBox Properties	1065
Stack Properties	1065
Tile Properties	1066
VBox Properties	1066
Properties Applicable to Controls	1067
Properties Applicable to Labeled Nodes	1067
ListView Properties	1068
ScrollBar Properties	1068
ScrollView Properties	1069
Separator Properties	1069
Slider Properties	1069
TextBox and PasswordBox Properties	1070

Index 1071

Preface

The official launch of the Java programming language coincided with huge public interest in the World Wide Web. Home computers were becoming affordable, and large numbers of homes were connected to the Internet, mostly using slow, dial-up lines (remember those?), and Netscape Navigator was by far the most popular web browser. In 1994, a version of this browser was shipped with a plug-in that allowed Java applets to be embedded in HTML web pages. Applets brought motion and dynamic content to what had formerly been a mainly static World Wide Web. So great was the impact that a bouncing-head animation, which was actually a Java applet, was shown on an evening news broadcast in the United Kingdom. It seemed that applets and the Java programming language were set for a bright future.

The 1.0 release of the Java Development Kit (JDK) included the compiler together with a relatively small set of class libraries that provided mainly I/O and networking facilities and a primitive user interface toolkit called AWT (Abstract Window Toolkit). One of the most novel things about Java and AWT was that they allowed the programmer to write an application that would run unchanged on both Microsoft Windows and UNIX. The platform became more robust with the release of JDK 1.1, which for a long time was the standard Java platform and was used to write both applets and free-standing applications, deployed on corporate intranets.

Although Java was born as a desktop technology, it didn't stay that way for very long. Novel though it was, there were problems with the applet programming model as implemented in JDK 1.1. The most obvious was speed—both of delivery and of execution. Before the arrival of Java Archive (JAR) files, an applet's class files were hosted on a website and downloaded individually, on demand. This meant that anything other than a very simple applet was slow to start and would be prone to freezing during execution if a new class file had to be fetched. Furthermore, the early versions of the Java Virtual Machine (JVM) were not well optimized, especially when it came to garbage collection, which would often cause execution to appear to be suspended for a noticeable time.

Neither was AWT a comprehensive toolkit—there were very few components, meaning that an applet or application either had to be very basic or its author had to invest considerable time and effort to write and debug custom components. Unfavorable comparisons were being made with native applications and this continued even with the release of Swing, a more comprehensive user interface toolkit that was an add-on to JDK 1.1. Swing was (and is) very powerful, but it is complex, and it gained a reputation, at least partially justified, for being slow and unwieldy.

Swing was integrated into the next major release of the Java platform, which was given the name Java 2 Standard Edition, to distinguish it from the Enterprise Edition, which was focused on the application programming interfaces (APIs) need to build web applications. The other major desktop feature in the Java 2 SE, or JDK 1.2 as it was also known, was Java2D. Java2D greatly improved the graphical capabilities of the platform, with improved font support, the ability to treat text as a shape, the ability to rotate, shear, and scale shapes, and a host of other features. All of this was implemented by a new *graphics pipeline*, which was powerful but also slow. Swing applications originally implemented on JDK 1.1 ran more slowly on the Java 2 platform. In addition to this, the Java2D APIs were seen as hard to learn and difficult to use. Interest in Java as a desktop platform began to wane as more and more companies turned to web applications to satisfy their needs. The performance problems in Java2D and Swing were addressed with subsequent releases so that by the time JDK 1.4 appeared, desktop applications written with Swing could outperform those running on the now obsolete JDK 1.1, but this was not enough to prompt a serious revival in the fortunes of desktop Java.

Fortunately for Java desktop developers, at the JavaOne conference in May 2007, Sun Microsystems made several announcements that were aimed at reclaiming the desktop from other vendors that had moved in to satisfy the need for a rich client-side platform—specifically Adobe with Flash and Flex and Microsoft with its newly announced Silverlight product. Sun announced a major overhaul of its client-side technologies, beginning with J2SE itself and culminating with a new technology called JavaFX.

What Is JavaFX?

Sun's marketing organization describes JavaFX as a new platform for writing *rich Internet applications* (RIAs), or to those of us who would prefer to skip the marketing hype, it is a new language and runtime environment that lets you write rich client applications and then deploy them to users' desktops, mobile devices, Blu-ray players, and TVs (provided that they have the appropriate supporting environment). For a high-level description of the language and its runtime, see the introduction in Chapter 1, "An Overview of JavaFX."

JavaFX runs on the Java platform. The early releases were exclusively targeted at the Microsoft Windows and Apple Mac OS X platforms, but there are now fully supported versions for Linux and OpenSolaris, and a developer release that runs on mobile devices based on Windows Mobile is also available. By the time you read this book, it is likely that you can buy mobile handsets that include a fully supported JavaFX software stack.

From a developer's point of view, writing JavaFX applications couldn't be easier. If you are familiar with Java or a similar language such as C++, you will find the transition to the syntax of the JavaFX Script language easy to make. Much of what you already know remains true in JavaFX, and there are some new features that, once you use them, you will wonder how you ever managed to work without them. The most obvious of these is *binding*, which enables you to link one piece of application state to another.

There is a whole chapter on binding in this book, but there is a simple introduction to the concept in Chapter 1.

The JavaFX user interface libraries offer a clean and very easy-to-use API. If you are a Swing developer, you will recognize many of the concepts, but you may be disappointed at first because the feature set is smaller than that provided by Swing. This is, of course, only a temporary state of affairs. Over time, the API will be expanded, and it should eventually be possible to do in JavaFX almost anything that Swing allows today.¹ On the plus side, the concepts of the *scene graph* and *nodes* make it easy to build impressive user interfaces. As you'll see in the first few chapters of this book, the features built in to the scene graph APIs let you accomplish in a few lines of JavaFX code things that would have taken many more lines of Java code and also an intimate knowledge of the Java2D APIs. Even better, the resulting code will run (with a few exceptions) on both your desktop and on your mobile phone, so you don't need to work with two different APIs and two different development environments to create a truly portable application.²

For those of us who are developers first and artists second (or, like me, not an artist at all), JavaFX makes it easy to work with professional graphic designers to create a user interface that doesn't look like it was designed by a programmer, to be used by another programmer. You can import graphic elements or an entire user interface that was originally created in Adobe Illustrator or Adobe Photoshop and then animate it in response to user actions or the passage of time. This capability is provided by the JavaFX Production Suite, a separate download that contains plug-ins that export graphics from the Adobe development tools in a format that can be read by the JavaFX runtime.

If you are reading this book because you expect to use JavaFX at your place of work, you probably already have an audience of users waiting for your application, but if you are intending to develop an application as a private enterprise and you would like to make some money from it, you could try posting it in the *Java Warehouse*. JavaFX applications in the Java Warehouse appear in and can be sold from the *Java Store*. The Java Warehouse and Java Store were announced at JavaOne in 2009 and should represent a revenue opportunity for talented Java and JavaFX programmers. You can find the Java Warehouse at <http://java.sun.com/warehouse> and the Java Store at <http://www.java.sun.com/store>.

¹ In the meanwhile, you can always use Java APIs to do almost anything that you can't do directly in JavaFX.

² Of course, there are limitations to this. It is easily possible to write a JavaFX application that works well on the desktop but is completely unusable on a mobile device, primarily because mobile devices have smaller screens and less powerful processors. However, if you are careful, it is certainly possible to write a JavaFX application that works on more than one type of device.

How This Book Is Organized

This book is a complete guide to the JavaFX platform—the language, the user interface libraries, and the tools that you can use to develop JavaFX applications. It is divided into three parts. The first part contains an introduction to the platform and a detailed description of the JavaFX script programming language, the second discusses the user interface libraries, and the third covers the APIs that let you access external systems and the tools provided to let you package and deploy your JavaFX applications. Here is an overview of what you’ll find in each chapter.

- Chapter 1, “An Overview of JavaFX,” contains an overview of the JavaFX platform, the JavaFX script language, and the tools for development and deployment of JavaFX applications.
- Chapter 2, “JavaFX Script Basics,” introduces the JavaFX script language by looking at the structure of a JavaFX script file, how scripts are compiled, and how they are executed.
- Chapter 3, “JavaFX Development,” shows you how to create a new JavaFX project in both the NetBeans and Eclipse integrated development environments (IDEs) and walks you through the coding, compilation, and execution of a simple JavaFX application. You also see how to compile and run JavaFX applications from the command line and how to extract documentation from JavaFX source files.
- Chapter 4, “A Simple JavaFX Application,” builds a more complex JavaFX application and then shows you how to run it as a desktop application, an applet, and on an emulated mobile device. The second half of this chapter shows you how to debug and profile JavaFX code.
- Chapter 5, “Variables and Data Types,” is the first of nine chapters that describe the JavaFX script language in detail, beginning with what you need to know about variables and the data types that the language supports. We also discuss the support that the JavaFX runtime provides for the internationalization of applications that need to support more than one native language.
- Chapter 6, “Expressions, Functions, and Object Literals,” discusses the arithmetic, Boolean, and other operators that the language provides and introduces the two types of functions that exist in JavaFX. JavaFX functions are first-class citizens of the language, which means, among other things, that you can have a variable that refers to a function, and you can pass a function reference as an argument to another function. This chapter also discusses object literals, which are the nearest thing that JavaFX has to a Java constructor.
- Chapter 7, “Sequences,” introduces one of the more powerful features of JavaFX—sequences. Although they are superficially like Java arrays, the ability of sequences to report changes to their content, when used together with either binding or triggers, makes it easy to create user interfaces that display or let the user manipulate lists of objects.

- Chapter 8, “Controlling Program Flow,” covers the JavaFX keywords that enable you to change the flow of control in an application. The `if`, `while`, and `for` statements are, as you might expect, similar to their Java equivalents, but there are some important differences. For example, `if` is an expression, which means it can return a value, and `for` operates on a sequence of values and may return a sequence of derived values.
- Chapter 9, “Binding,” discusses what is probably the single most important feature of JavaFX. The `bind` keyword enables you to create an association between an expression and a variable so that whenever the value of the expression changes, its new value is written to the variable without programmer intervention. As you’ll see in this chapter and throughout the book, this makes it much easier to create user interfaces in JavaFX than it is in Java.
- Chapter 10, “Triggers,” introduces the trigger mechanism, which allows arbitrary code to be run when the value of a variable changes.
- Chapter 11, “JavaFX Script Classes,” shows you how to write your own JavaFX classes. Unlike Java, code in JavaFX does not have to be coded inside an explicit class definition. As a result, it is possible to write complete applications without knowing how to define a JavaFX class. However, if you want to create a library of reusable code, you need to create your own JavaFX classes.
- Chapter 12, “Platform APIs,” covers a subset of the JavaFX runtime that is not part of the user interface libraries, including APIs that allow you to access application parameters and system properties and a couple of classes that allow even untrusted applications to store information on a user’s computer.
- Chapter 13, “Reflection,” discusses the reflective capabilities of the JavaFX platform. Reflection is a feature that can be of great use to developers who write frameworks or software tools where the data types of the objects being manipulated are not always known in advance. This chapter covers all the JavaFX reflection APIs.
- Chapter 14, “User Interface Basics,” is the opening chapter of the part of the book that covers the user interface classes. It introduces the `Stage` and `Scene` classes, which represent the top-level container of an application, and provides a high-level view of the scene graph and the nodes from which it is composed.
- Chapter 15, “Node Variables and Events,” takes a detailed look at the variables that are common to all nodes and the node events that your application registers for to track mouse and keyboard activity. It also discusses colors and color gradients.
- Chapter 16, “Shapes, Text, and Images,” opens with a discussion of the basic node types that are provided by the JavaFX runtime (such as rectangles, circles, ellipses, and so on) and covers font handling and the rendering of text in JavaFX. Finally, you’ll see how to load, display, and, if necessary, resize an image.
- Chapter 17, “Coordinates, Transformations, and Layout,” describes the coordinate system used to place and describe the size of a node and the transformations, such

as rotation, translation, and scaling, that can be applied to a node. The second half of the chapter introduces the `Container` class and a variety of its subclasses that have an embedded node layout policy.

- Chapter 18, “Animation,” describes the animation features of JavaFX. Using just constructs, you can change the position or appearance of a node or group of nodes over a period of time to give the impression of an animation. This chapter covers both the `Timeline` class, which is the basis for JavaFX animation, and a set of higher-level classes called transitions that let you specify an animation in a more abstract fashion.
- Chapter 19, “Video and Audio,” covers the JavaFX classes that let you play video and audio clips. As you’ll see, JavaFX supports several platform-specific media formats (including MP3, WMV, and AVI) and a cross-platform format that can be played on any platform that supports JavaFX.
- Chapter 20, “Effects and Blending,” shows you how to apply a range of graphical effects to a node. The effects supported include shadowing, blurring, and various different lighting effects. Effects are currently supported only on the desktop platform.
- Chapter 21, “Importing Graphics,” describes how you, as a JavaFX developer, can work with a designer to create graphics that you can then import into your application. The JavaFX Production Suite, which is downloaded separately from the JavaFX SDK, provides plug-ins that enable a graphic designer to prepare a graphic in Adobe Photoshop or Adobe Illustrator and export it in a form that is suitable for import into a JavaFX application. You’ll see a couple of examples that illustrate the overall workflow, and another that shows you how to capture graphics created in a graphics tool that creates output in SVG (Scalable Vector Graphics) format.
- Chapter 22, “Cross-Platform Controls,” discusses the node-based controls that allow user input and display lists of data and other information to the user. These controls work on the desktop, on mobile devices, and with Java TV.
- Chapter 23, “Style Sheets,” shows you how to change the appearance of your application without changing a line of code by using a style sheet. The style sheets supported by the JavaFX runtime are similar to the Cascading Style Sheets (CSS) documents used with HTML and provide many of the same features. Style sheets work with the basic node classes, controls, and custom controls.
- Chapter 24, “Using Swing Controls,” shows you how to embed Swing components into your JavaFX application. An embedded Swing component is a node, which means that you can rotate it, scale it, shear it, change its opacity, and so on. The JavaFX runtime contains equivalents for many of the standard Swing components, and those that are not directly supported, including third-party components, can be made available to a JavaFX application through a wrapper class.

- Chapter 25, “Building Custom Nodes,” describes how to create your own custom nodes, controls, and layouts and how to use style sheets to change their appearance or behavior.
- Chapter 26, “Charts,” covers the JavaFX classes that let you render data in the form of a chart. Support is provided for various different types of charts, including pie charts, line charts, area charts, and scatter charts. All these classes are highly customizable, either in code or from a style sheet.
- Chapter 27, “Using External Data Sources,” discusses the support in the JavaFX runtime for retrieving data from a web server or a web service and presenting it to the user. At the lowest level, you can use the `HttpRequest` class to make an asynchronous call to a web server. If the data is returned in Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format, you can use the `PullParser` class to parse it and then display the results in your user interface. At a higher level, JavaFX provides a couple of classes that read an RSS or Atom feed and convert the content to JavaFX objects, thus saving you the trouble of parsing it yourself.
- Chapter 28, “Packaging and Deployment,” covers the `javafxpackager` command, which allows you to package your JavaFX application and shows you how to deploy it for delivery as a desktop application, an applet, or to a mobile device.

This book is *not* intended to be read from front to back. Most chapters have a mixture of basic and advanced material, so you definitely do not need to read everything in a chapter before you can progress to the next. I recommend that you install the software that you need, get the example source code, and then read the first four chapters. At this point, you should have a good grasp of the fundamentals of the language and how to compile and run JavaFX application.

Given that this book is aimed mainly at developers who have experience with Java, the odds are good that you will be able to get a grip on the JavaFX Script language very quickly, and you should only need to skim over the language chapters before diving into the in-depth coverage of the graphical user interface (GUI) libraries in Part III, “User Interfaces with JavaFX.” You’ll find that each of the GUI chapters contains detailed information on the topic that it covers, so feel free to skip from chapter to chapter picking out the information that you need from each. When you need more detail on a language feature, you can return to the relevant chapter in Part II, “The JavaFX Script Language,” and read about it.

Getting the Software

This book includes more than 400 example JavaFX applications. Most of them are small and are focused on a single aspect of the language or the runtime libraries. To get the best from this book, you need to download and install both the example source code and the JavaFX runtime and documentation, as described in the paragraphs that follow. In addition to the software listed here, you need to have a Java Development Kit

installed. The minimum requirement for running JavaFX applications is Java 5, but I recommend that you get the latest release of the platform that is currently available. To use all the features of the JavaFX platform, you need at least Java 6 update 10. You can download the JDK from <http://java.sun.com/javase/downloads/index.jsp>.

The JavaFX SDK

There are two ways to install a JavaFX runtime on your computer: You can get the JavaFX SDK, which allows you to develop applications using an editor of your choice and a set of command line tools; or you can get the JavaFX plug-in for either the NetBeans or Eclipse IDEs. In fact, it is a good idea to get both the SDK and an IDE plug-in, because the API documentation in the SDK is, at least in my opinion, easier to use than the help facility in an IDE, while the IDEs allow you to find your way around the APIs more easily because they provide code completion, automatic management of imports, and an easy way to execute your applications either on the desktop or on a mobile device emulator.

To download the SDK, go to <http://javafx.com>. There, you will find a link to the downloads page where you will find most of the additional software that you need. The `bin` directory of the SDK contains command-line tools that let you run the compiler, package an application for deployment, extract and format JavaFX documentation, and run a compiled application. You'll find documentation for the command-lines tools in the `docs` directory and the API documentation for the platform in the directory `docs/api`. It is worth bookmarking the API documentation in your browser because you will probably refer to it frequently.

The text and examples in this book refer to and have been tested with JavaFX version 1.3, which was released in April 2010.

The NetBeans IDE

If you'd like to use the NetBeans IDE to compile and run the example source code for this book, you can get it from the download page at <http://javafx.com>. Alternatively, you can download it from the NetBeans site at <http://www.netbeans.org/features/javafx>. Be sure to choose a package that contains the JavaFX plug-in.

If you already have the IDE (version 6.9 or higher supports JavaFX 1.3), you can add the JavaFX plug-in by going to Tools, Plugins, opening the Available Plugins tab, and installing the JavaFX Kit and JavaFX SDK plug-ins. If these plug-ins do not appear, click the Reload Catalog button to update the list.

Having installed the IDE, you need to install the plug-ins for web application development, which are required for the examples in Chapter 27, "Using External Data Sources." See the "GlassFish" section, below, for further information.

As new releases of the JavaFX SDK appear, you can update the IDE plug-in from the Installed tab of the Plugins window. This is the recommended approach because it is simple and ensures that you have the latest IDE features for JavaFX. Alternatively, you

can download and install a new SDK and then make it available in the IDE as follows:

1. Go to Tools, Java Platforms, and click Add Platform.
2. Select JavaFX Platform and click Next.
3. Assign a name for the SDK and navigate to its install directory, and then click Finish.

You can assign your new SDK version to a JavaFX project by right-clicking the project node in the Projects window, selecting Properties and the Libraries, and choosing the SDK to be used from the JavaFX Platform combo box.

The Eclipse Plug-In for JavaFX

There are two Eclipse plug-ins for JavaFX, one provided by Sun Microsystems, the other by Exadel (<http://www.exadel.com>). In this book, we use the Sun Microsystems plug-in, which requires Eclipse version 3.4 or later.

Plug-In Installation and Update for Eclipse 3.4

To install the Sun Microsystems plug-in, do the following:

1. On the Eclipse main menu bar, choose Help, Software Updates, and then open the Available Software tab.
2. Click the Add Site button and enter the following URL into the Add Site dialog:
<http://javafx.com/downloads/eclipse-plugin/>.
3. The site will be added to the available software list. Open the site node to show the JavaFX Features node and select it.
4. Click the Install button to install the plug-in.

You can update the plug-in to a later release, or check whether there is an update, as follows:

1. Select Help, Software Updates, and then open the Installed Software tab.
2. Select the row for JavaFX Feature.
3. Click Update.

If an update is available, you will be prompted to accept the license, and then the update will be installed.

Plug-In Installation and Update for Eclipse 3.5

The steps required to install the plug-in for Eclipse 3.5 are as follows:

1. On the main menu bar, select Help, Install New Software.
2. In the Work With field on the Available Software dialog, enter the URL
<http://javafx.com/downloads/eclipse-plugin/>, and then click Add.
3. In the Add Site dialog that appears, give the site a name (for example, JavaFX Plugin) and click OK.

4. After a short pause, an entry for JavaFX Features appears in the Available Software dialog. Select the check box for this entry, and then click Next.
5. The plug-in details will be obtained and displayed. Click Next, and on the next page, review the license. If it is acceptable, select I Accept the Terms of the License Agreement, and then click Finish to install the plug-in.

To update the plug-in to a later release, or to check whether an update is available, select Help. Check for Updates and follow the instructions to install a new version of the plug-in.

GlassFish

You need a web server, such as the one that comes with the GlassFish application server, to run the examples for Chapter 27. You also need to install plug-ins for NetBeans or Eclipse that let you work with GlassFish from within the IDE. In this section, we describe how to work with GlassFish from within NetBeans.

Installing GlassFish

You can get GlassFish from the GlassFish community downloads page at <https://glassfish.dev.java.net/public/downloadsindex.html>. I used GlassFish version 2 when writing this book, but the examples should also work with version 3. During the installation process, you will be asked for the username and password to be used for server administration. You must supply these values again when registering the application server with the NetBeans or Eclipse IDE plug-in.

Installing the NetBeans Plug-Ins

To install the plug-ins required to work with the GlassFish application server from within the NetBeans IDE, do the following:

1. From the main menu, select Tools, Plugins, and open the Available Plugins tab.
2. Select Java Web Applications and click Install. Follow the prompts until the plug-in, and a few other plug-ins that it requires, is installed.
3. You now need to register your GlassFish server with the plug-in, which you can do as follows:
 - a. On the main menu, select Window, Services. This opens the Service view.
 - b. Right-click the Servers node and select Add Server.
 - c. In the Add Server Instance dialog, select the version of GlassFish that you installed and click Next.
 - d. In the Server Location field, enter the installation directory of your GlassFish server and click Next.
 - e. Enter the administrator username and password that you assigned when installing GlassFish and click Finish.

Your GlassFish server should now appear under the Servers node in the Services view. If you expand the Databases node, you should see a JavaDB entry with URL `jdbc:derby://localhost:1527/sample`. Start the database server by right-clicking the sample node and selecting Connect. When the database starts, expand the `APP` node and then the `TABLES` node, and you should see the `CUSTOMER` table from the sample database. This table will be used in Chapter 27.

The JavaFX Production Suite

The JavaFX Production Suite provides plug-ins for Adobe Photoshop and Adobe Illustrator that let you export artwork in a form that can be easily imported into a JavaFX application. You'll find a full description of the Production Suite, together with installation instructions, in Chapter 21, "Importing Graphics."

JavaDB

One of the examples in Chapter 27 uses the JavaDB database. If you use Windows and you have the Java 6 JDK installed, you already have JavaDB on your computer—you'll find it at `C:\Program Files\Sun\JavaDB`. Otherwise, you should download JavaDB from <http://developers.sun.com/javadb/downloads/index.jsp> and install it.

The Example Source Code

A Zip file containing the example source code for this book can be found at <http://www.informit.com/title/9780321601650>. (Click on the Downloads tab.) Extract the content of the file into a directory and then set the value of the `javafx-sdk.dir` property in the `build.properties` file so that it points to the directory in which you installed the JavaFX SDK. You can then import the source code into the NetBeans or Eclipse IDE.

Note

You need to set this property only if you plan to build and run the examples from the command line, which requires that you also download the JavaFX SDK. If you plan to use an IDE to build and run the examples, you do not need to edit this file. You will find instructions for building and running the examples from the command line in Appendix A, "Using JavaFX Command-Line Tools."

Importing the Example Source Code into the NetBeans IDE

Assuming that you have installed all the relevant plug-ins, you can import the example source code into NetBeans as follows:

1. On the main menu, select File, Open Project.
2. In the Open Project dialog, navigate to the directory in which you installed the example source code and then into the `desktop` subdirectory.

3. Select the project file in that directory (it's called JavaFX Book Desktop NetBeans Project) and click Open Project.

You may get a message about a missing reference, which we will resolve shortly. Close the message dialog and the project will be imported.

Repeat this process for the projects in the subdirectories `gui`, `moregui`, `intro`, `language`, and `server/ExternalData`.

To fix the missing reference problem, do the following:

1. Right-click the node for the JavaFX Book Desktop NetBeans Project and select Resolve Reference Problems.
2. In the Resolve Reference Problems dialog, click Resolve to open the Library Manager dialog, and then click New Library.
3. In the New Library dialog, enter the name **JavaDBJavaFXBook** and click OK.
4. In the Library Manager dialog, click Add JAR/Folder and navigate to the directory in which JavaDB is installed. If you run the Java 6 JDK on Windows, you will find JavaDB in the folder `C:\Program Files\Sun\JavaDB`. Otherwise, you should install JavaDB as described in the section “JavaDB,” earlier in this Preface. Navigate to the `lib` directory and select the file `derbyclient.jar`. Click OK, and then click OK to close the Library Manager.

To run any of the examples, right-click the source file and select Run File. You'll find more information on how to run the examples in Chapter 3, “JavaFX Script Development,” and Chapter 4, “A Simple JavaFX Application.”

Importing the Example Source Code into the Eclipse IDE

If you have the JavaFX plug-in installed, you can build and run the example source code in Eclipse.

Warning

Compiling all the example source code requires a lot of Java heap space. To make sure that you don't run out of memory, specify a larger heap when running Eclipse, like this:

```
eclipse.exe -vmargs -Xmx1024M
```

To import the example source code, do the following:

1. On the main menu, select File, Import.
2. In the Import dialog, select General, Existing Projects into Workspace, and then click Next.
3. Enter the directory in which you installed the example source code as the root directory.
4. Select all the projects that appear in the Projects list, and then click Finish.

To run an example, right-click the JavaFX file and select Run As, JavaFX Application. When the Edit Configuration dialog appears, click Run. You'll find more information on how to run the examples in Chapters 3 and 4.

Conventions

`Courier` font is used to indicate JavaFX and Java code, both in code listings and in the code extracts embedded in the text. Lines of code that are of particular interest are highlighted in bold.

Throughout this book, you will find tables that list the accessible variables of JavaFX classes. Each row corresponds to one variable, and the columns contain the variable's name, its type, the permitted modes of access, its default value, and a description. Here's an example:

Variable	Type	Access	Default	Description
focused	Boolean	R	(None)	Whether the Stage is focused
icons	Image[]	RW	Empty	The icons used in the title bar of the top-level container
title	String	RW	Empty string	The title used in the title bar of the top-level container
visible	Boolean	RW	true	The visibility of the Stage

The value in the Access column contains the permitted access modes for application code (more specifically, for code that is not related to the owning class—that is, code that is not in the same package as the class or in a subclass). The possible access modes are as follows.

R	The value can be read.
W	The value can be written at any time.
I	The value can be set, but only when an instance of the class is being created (that is, at initialization time).

Further Information

Even though JavaFX is a recent innovation, there are already many sources that you can refer to for up-to-date information. The most obvious of these is the JavaFX website at <http://javafx.com>, where you can download the latest release, find hints and tips, and browse through a gallery of examples. There is also a set of forums dedicated to JavaFX at <http://forums.sun.com/category.jspa?categoryID=132> and a JavaFX blog at <http://blogs.sun.com/javafx/>.

JavaFX is still a young technology and currently lacks some of the features that you'll find in Swing or in comparable toolkits. If you can't find what you need in the JavaFX runtime, you might find it instead at <http://jfxtras.org/>, a site dedicated to the development, discussion, and extension of the JavaFX platform. Here you will find third-party controls, shapes, layout containers, and lots of sample code to help you get the most out of the platform.

Feedback

Although this book has been reviewed for technical accuracy, it is inevitable that some errors remain. If you find something that you think needs to be corrected or that could be improved, or if there is something that you think could usefully be added in future editions of this book, please contact the author by e-mail at kimtopley@gmail.com.

Acknowledgments

This is the eighth time that I have been through the process of writing a book. Two of those books didn't see the light of day, but the six that did (one of them being a second edition) have been interesting, character-building experiences. There is no doubt that writing a book involves a major commitment of time and effort. This book was conceived when the first version of JavaFX was officially announced at JavaOne in 2007. By the time I finished the first draft, I had been working on it in my spare time for more than two years. Naturally, I blame the time overrun on Sun Microsystems, who caused me to discard at least 300 pages of text by throwing away virtually all the language and libraries between the announcement in 2007 and the next JavaOne in May 2008, but in so doing creating a much better product.

I am, once more, indebted to my editor, Greg Doench, who accepted my proposal for this book based on a very sketchy outline and shepherded the whole project to completion despite my attempts to frustrate the process by repeatedly adding "just one more" section to an ever-growing table of contents. Thanks also to Michael Thurston, Keith Cline, Julie Nahil, and the rest of the Addison-Wesley team who converted this book from a raw manuscript to the final result that you now hold in your hands. Along the way, reviewers Peter Pilgrim and Joe Bowbeer provided insightful technical feedback for which I'm very grateful.

I was lucky enough to receive help from members of the JavaFX team at Sun Microsystems while writing this book. My thanks go, in no particular order, to Brian Goetz, Amy Fowler, Richard Bair, Robert Field, and Per Bothner, who took the time to answer my questions even when they were busy trying to meet deadlines of their own.

Effects and Blending

In this chapter, you see how to use the classes in the `javafx.scene.effects` and `javafx.scene.effects.lighting` packages, which implement graphical effects that you can use to enhance the appearance of your application. After discussing effects in general, the first part of this chapter describes 15 different effects that you can use to create blurs, shadows, warps, and various lighting effects. The second part describes the blending effect, which provides 19 different ways to combine two inputs, such as a node and another effect, to produce an output. The same 19 blending modes can also be applied to a group (and therefore also to a container) to control how the pixels for intersecting nodes are combined. The last part of this chapter looks at the ways in which you can light a scene by using the `Lighting` effect.

Effects are a feature of the desktop profile—they do not work on mobile devices—so the example source code for this chapter can all be found in the `javafxeffects` package in the JavaFX Book Desktop project. You can use the conditional feature API described in Chapter 12, “Platform API,” to determine at runtime whether effects are available to your application.

Effects Overview

An effect is a graphical filter that accepts an input (and in some cases more than one input) and modifies it in some way to produce an output. The output is either rendered as part of a scene or becomes the input for another effect. The combination of a node and one or more effects is referred to here as an *effects chain*.

Effects Chains

Figure 20-1 shows two common effects chains. An effects chain contains, at minimum, one node and one effect.

In the first chain, at the top of the figure, a single effect is applied to a node, and the result is drawn onto the scene. The second chain contains two effects. In this case, the first effect is applied to the node, which results in an output image that becomes the input for the second effect. It is the output of the second effect that will be drawn onto the scene.

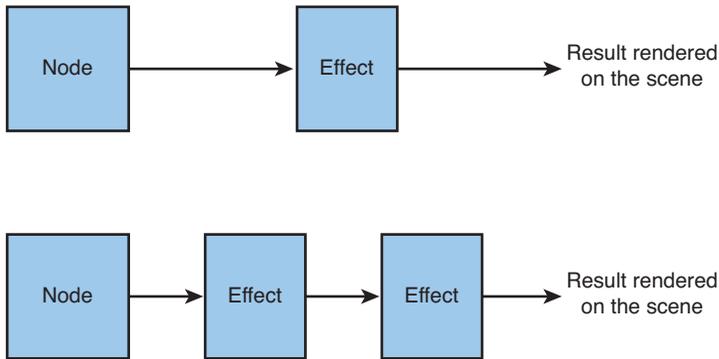


Figure 20-1 Relationship between effects and nodes

Effects and Nodes

When an effect is applied to a node, the output of the effects chain logically replaces the node itself on the screen. In general, an effect will change the bounds of a node. For example, adding a shadow to a node by using the `DropShadow` effect will typically make it wider and taller. The node’s local and parent bounds are adjusted based on the result of the effects chain, but its layout bounds are not affected. When a node has both effects and transformations, the effect is applied before the transformations. This means, for example, that adding a shadow and then scaling the node will also scale up the shadow.

An effect is linked to a node via its `effect` variable.¹ The code in Listing 20-1, which you will find in the file `javafxeffects/Effects1.fx`, shows how simple it is to add an effect to a node. In this case, a drop shadow is added by the three lines of code starting on line 19. Figure 20-2 shows the result.

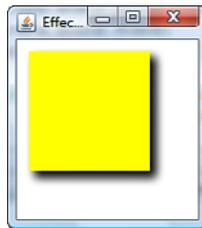


Figure 20-2 A rectangle with a drop shadow effect

¹ This variable does not exist in the mobile profile. If you get compilation errors when trying to run the examples for this chapter, the reason is most likely that you are trying to build for the mobile emulator.

Listing 20-1 Adding an Effect to a Node

```
1 package javafxeffects;
2
3 import javafx.scene.effect.DropShadow;
4 import javafx.scene.paint.Color;
5 import javafx.scene.Scene;
6 import javafx.scene.shape.Rectangle;
7 import javafx.stage.Stage;
8
9 var rect: Rectangle;
10 Stage {
11     title: "Effects #1"
12     scene: Scene {
13         width: 150 height: 150
14         content: [
15             rect = Rectangle {
16                 x: 10 y: 10
17                 width: 100 height: 100
18                 fill: Color.YELLOW
19                 effect: DropShadow {
20                     offsetX: 5 offsetY: 5
21                 }
22             }
23         ]
24     }
25 }
26 println("Layout bounds: {rect.layoutBounds}");
27 println("Parent bounds: {rect.boundsInParent}");
```

The last two lines of Listing 20-1 print the layout bounds and parent bounds of the rectangle. Here's the result:

```
Layout bounds: BoundingBox [minX = 10.0, minY=10.0, maxX=110.0, maxY=110.0,
width=100.0, height=100.0]
Parent bounds: BoundingBox [minX = 6.0, minY=6.0, maxX=124.0,
maxY=124.0,width=118.0, height=118.0]
```

As you can see, the rectangle's layout bounds correspond to its specified width and height (because the layout bounds do not include the results of the effect), but width and height of the parent bounds have both increased from 100 to 118 because of the space occupied by the drop shadow.

Applying more than one effect is simply a matter of linking one effect to another. The following code (which you'll find in the file `javafxeffects/Effects2.fx`) adds a reflection to the drop shadow, giving the result shown in Figure 20-3:

```
Rectangle {
    x: 10
    y: 10
```

```

width: 100
height: 100
fill: Color.YELLOW
effect: Reflection {
    input: DropShadow {
        offsetX: 5 offsetY: 5
    }
}
}

```

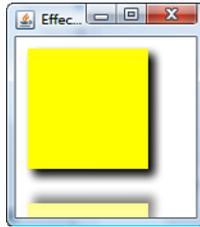


Figure 20-3 Applying two effects to the same rectangle

The linkage between the effects is made through the `input` variable of the reflection effect—the drop shadow is applied first, and the result of this becomes the input to the reflection effect. When no input is specified, the node itself is used as the input, as in the case of the drop shadow effect.

Not all effects have an `input` variable. Those that don't can only appear as the first (or only) entry in the effects chain. The `DropShadow` class is an example of this.² Other effects can have more than one input, such as the `Blend` effect that you'll see in the second part of this chapter.

As noted earlier, transformations are applied after any effects, so they apply to the effects, too. The following code, from the file `javafxeffects/Effects3.fx`, adds a rotation to the two effects that are applied to the `Rectangle`, as shown in Figure 20-4.

```

Rectangle {
    x: 10 y: 10
    width: 100 height: 100
    fill: Color.YELLOW
    rotate: -45
    effect: Reflection {
        input: DropShadow {

```

² The lack of an `input` variable in the `DropShadow` and other effects classes may be a temporary state of affairs. An issue has been filed at <http://javafx-jira.kenai.com> that may result in this being changed.

```

    offsetX: 5 offsetY: 5
  }
}

```

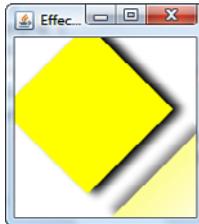


Figure 20-4 Using effects and transformations together.

Effects and Groups

A particularly powerful feature of effects is that they can be applied to a group. An effect that is applied to a group operates on the group as a whole. This is particularly useful if you want to create an effect that is uniform across the scene, such as the direction of lighting.

The following code, from the file `javafxeffects/Effects4.fx`, applies a `DropShadow` effect to a group that contains a rectangle and a circle; as you can see in Figure 20-5, this gives both of the nodes a `DropShadow` effect:

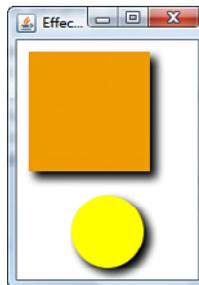


Figure 20-5 Applying an effect to a **Group**

```

Group {
  effect: DropShadow {
    offsetX: 5 offsetY: 5
  }
  content: [
    Rectangle {
      x: 10 y: 10
      width: 100 height: 100
    }
  ]
}

```

```

        fill: Color.ORANGE
    }
    Circle {
        centerX: 75 centerY: 160 radius: 30
        fill: Color.YELLOW
    }
}
]
}
```

The JavaFX Effects Classes

The JavaFX SDK provides 17 different effects that can be applied to any node. This section describes and illustrates all the effects, with the exception of `Blend` and `Lighting`, which have sections of their own at the end of the chapter. Each effect has a set of variables that you can use to customize it. As we examine each effect, we’ll take a look at the variables available and roughly consider what each of them does. There are too many combinations to illustrate them all in this chapter, so in most cases we limit ourselves to some typical examples. It is easy to experiment with these effects—all you need to do is modify the example source code. You can also use the Effects Playground application that you’ll find among the samples at <http://javafx.com>.

GaussianBlur

The `GaussianBlur` effect produces a blurred version of its input. The “Gaussian” part of the name refers to the fact that the output pixels are calculated by applying a Gaussian function to the source pixel and a group of pixels surrounding it. If you are interested in the details, you’ll find them at http://en.wikipedia.org/wiki/Gaussian_blur. The size of the group of adjacent pixels that are used to calculate the result is controlled by the `radius` variable (see Table 20-1). The larger the value of the `radius` variable, the greater the blur effect will be. When the value of this variable is 0, there is no blur at all.

Table 20-1 Variables of the `GaussianBlur` Class

Variable	Type	Access	Default	Description
<code>input</code>	<code>Effect</code>	RW	<code>null</code>	The input to this effect
<code>radius</code>	<code>Number</code>	RW	<code>10.0</code>	The radius of the area containing the source pixels used to create each target pixel, in the range 0 to 63, inclusive

Two example of the `GaussianBlur` effect applied to the image used in the previous two sections are shown in Figure 20-6. Here’s the code used to create this effect, which you’ll find in the file `javafxeffects/GaussianBlur1.fx`:

```

ImageView {
    image: Image { url: "{__DIR__}image1.jpg" }
    effect: GaussianBlur {
        radius: bind radiusSlider.value
    }
}

```

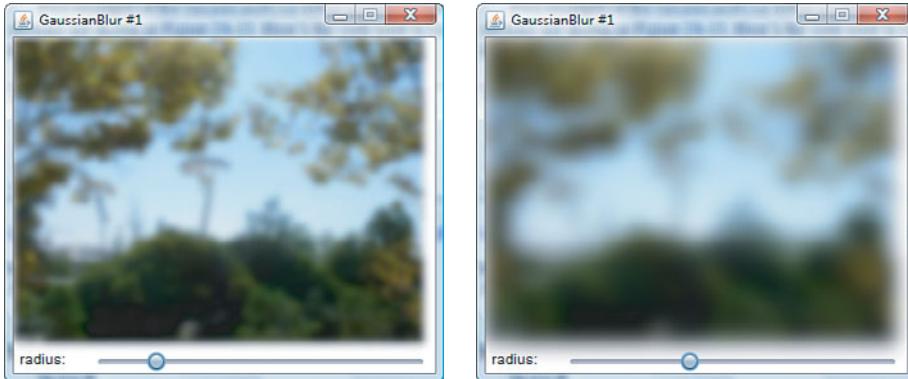


Figure 20-6 The `GaussianBlur` effect

The image on the left has a blur radius of 10, while the one on the right has radius 40.

BoxBlur

`GaussianBlur` is a high-quality effect, but it is also a relatively expensive one. The `BoxBlur` effect is a cheaper way to produce a blur, albeit one of lower quality. The variables that you can use to control the `BoxBlur` effect are listed in Table 20-2.

Table 20-2 Variables of the `BoxBlur` Class

Variable	Type	Access	Default	Description
<code>input</code>	Effect	RW	<code>null</code>	The input to this effect.
<code>height</code>	Number	RW	5.0	The vertical size of the box used to create the blur, in the range 0 to 255, inclusive.
<code>width</code>	Number	RW	5.0	The horizontal size of the box used to create the blur, in the range 0 to 255, inclusive.
<code>iterations</code>	Number	RW	1	The number of averaging iterations, in the range 0 to 3, inclusive. Higher values produce a smoother blur effect.

This effect works by replacing each pixel of the input by the result of averaging its value with those of its neighboring pixels. The pixels that take part in the operation are those in a rectangular area surrounding the source pixel, the dimensions of which are given by the `width` and `height` variables. You can see the effects of changing these variables by running the code in the file `javafxeffects/BoxBlur1.fx`. This example applies the `BoxBlur` effect to the same image as we used to illustrate `GaussianBlur`. The `width`, `height`, and `iterations` variables are set from three sliders that allow you to test the full ranges of values for each variable. Here's how the `BoxBlur` is applied:

```
ImageView {  
    image: Image { url: "{_DIR_}image1.jpg" }  
    effect: BoxBlur {  
        height: bind heightSlider.value  
        width: bind widthSlider.value  
        iterations: bind iterationSlider.value  
    }  
}
```

Increasing the value of the `height` variable produces a vertical blur, as shown on the left of Figure 20-7. Similarly, the `width` variable controls the extent of the blur in the horizontal direction.

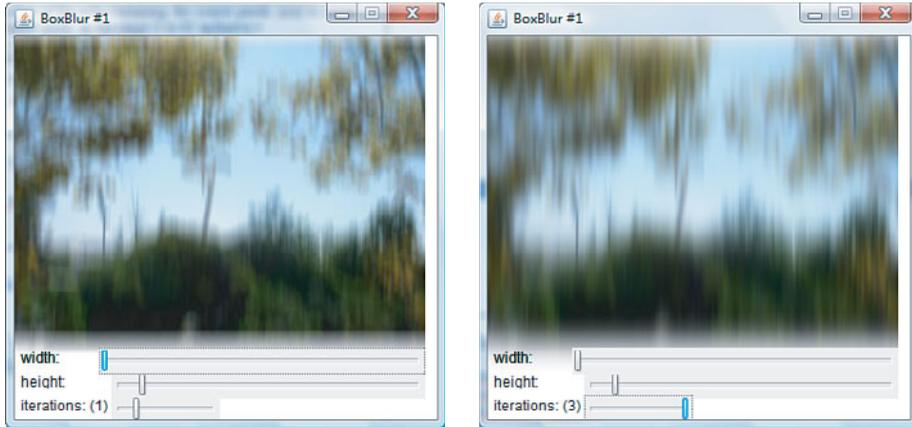


Figure 20-7 The Box Blur effect

You can use the `iterations` variable to increase the quality of the blur at the expense of greater CPU utilization. When this variable has the value 2 or 3, the averaging operation is repeated the specified number of times. On the second iteration, the averaged pix-

els are averaged against each other, which tends to smooth out any sharp differences that might exist near to edges in the input source. A third iteration produces an even smoother result. You can see the result of applying three iterations to a horizontal blur of the input image on the right of Figure 20-7. A `BoxBlur` with three iterations produces a result that is close to that of a `GaussianBlur`, but at a slightly lower cost.

MotionBlur

`MotionBlur` creates the effect that you would see if you look out of the window of a fast-moving vehicle. Like `GaussianBlur`, it has a `radius` variable that determines how much of a blur is to be applied. It also has an `angle` variable that lets you specify the direction of the motion. These variables are described in Table 20-3.

Table 20-3 Variables of the `MotionBlur` Class

Variable	Type	Access	Default	Description
<code>input</code>	<code>Effect</code>	RW	<code>null</code>	The input to this effect
<code>angle</code>	<code>Number</code>	RW	<code>0</code>	The angle of the motion blur
<code>radius</code>	<code>Number</code>	RW	<code>10.0</code>	The radius of the area containing the source pixels used to create each target pixel, in the range 0 to 63 inclusive

There are no restrictions on the value of the `angle` variable, but values greater than 360 are treated modulo 360, while negative values are first reduced modulo 360 and then have 180 added to them, so that -90 is the same as 270. The following extract shows how to apply a `MotionBlur` to a node.

```
image: Image { url: "{__DIR__}image1.jpg" }
effect: MotionBlur {
    angle: bind angleSlider.value
    radius: bind radiusSlider.value
}
}
```

If you run the code in the file `javafxeffects/MotionBlur1.fx`, you can experiment with the effects of different `radius` and `angle` values. Two examples with different angles are shown in Figure 20-8. The angle slider lets you vary the value of this variable from -180 when the thumb is at the far left to +180 at the far right. In the example on the left of the figure, the `angle` variable is 0, which gives a horizontal blur. In the example on the right, the `angle` variable has the value 90, and the result is a vertical blur. As is the case

elsewhere in the JavaFX API, angles are measured with 0 at the 3 o'clock position and increase as you move in a clockwise direction.

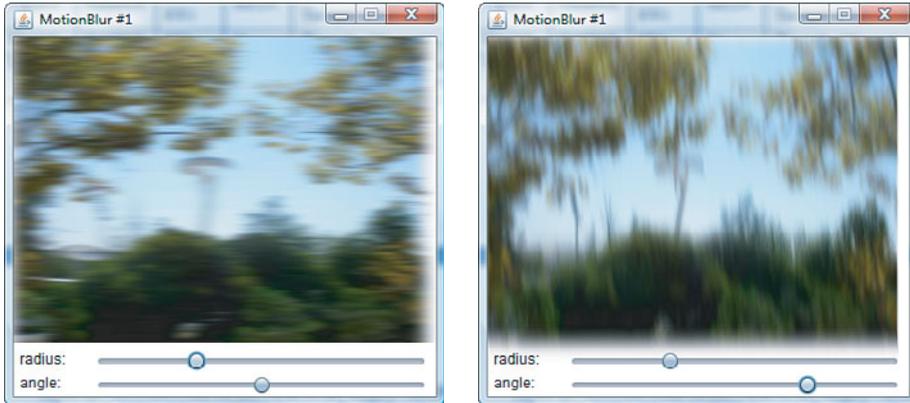


Figure 20-8 The `MotionBlur` effect

DropShadow

As you have already seen, the `DropShadow` effect draws a shadow that appears to be behind and partly obscured by the node to which it is applied. By using this effect with the appropriate variable settings, you can give the impression that the node is floating above a nearby surface or one slightly farther away. You can also change the nature of the shadow to indicate whether the light source is close to or a long way from the node. The variables that you can use to configure the `DropShadow` class are listed in Table 20-4.

Table 20-4 Variables of the `DropShadow` Class

Variable	Type	Access	Default	Description
<code>blurType</code>	Blur Type	RW	<code>THREE_PASS_BOX</code>	The type of blur to be used
<code>color</code>	Color	RW	<code>Color.BLACK</code>	The color to be used for the shadow
<code>offsetX</code>	Number	RW	0.0	The x-offset of the shadow
<code>offsetY</code>	Number	RW	0.0	The y-offset of the shadow
<code>radius</code>	Number	RW	10	The radius of the blur effect if a <code>GaussianBlur</code> is used
<code>width</code>	Number	RW	21	The width of the blur if <code>BoxBlur</code> is used
<code>height</code>	Number	RW	21	The height of the blur if <code>BoxBlur</code> is used
<code>spread</code>	Number	RW	0.0	The proportion of the radius (or box for <code>BoxBlur</code>) over which the shadow is fully opaque (see text)

The variables that you will most commonly set are `color`, `offsetX`, and `offsetY`. The `color` variable simply determines the color of the solid part of the shadow, which will generally be slightly darker than the background behind the node. By default, the shadow is black. The `offsetX` and `offsetY` variables control the displacement of the shadow relative to the node itself.

The `blurType` variable controls which of the supported types of blur is used at the edges of the shadow. This variable is of type `javafx.scene.effects.BlurType`, which has the following possible values:

- `BlurType.GAUSSIAN`: A `GaussianBlur`
- `BlurType.ONE_PASS_BOX`: A `BoxBlur` with one iteration
- `BlurType.TWO_PASS_BOX`: A `BoxBlur` with two iterations
- `BlurType.THREE_PASS_BOX`: A `BoxBlur` with three iteration

The code in the file `javafxeffects/DropShadow1.fx` creates a scene containing a rectangle with a `DropShadow` effect and a `GaussianBlur`. There are four sliders that let you control some of the variables listed in Table 20-5. You can use this program to experiment with various settings to see how they work. Figure 20-9 shows a typical example.

Table 20-5 Variables of the Shadow Class

Variable	Type	Access	Default	Description
<code>blurType</code>	<code>BlurType</code>	RW	<code>THREE_PASS_BOX</code>	The type of blur to be used
<code>input</code>	<code>Effect</code>	RW	<code>null</code>	The input to this effect
<code>color</code>	<code>Color</code>	RW	<code>Color.BLACK</code>	The color to be used for the shadow
<code>radius</code>	<code>Number</code>	RW	<code>10.0</code>	The radius of the blur effect if a <code>GaussianBlur</code> is used
<code>width</code>	<code>Number</code>	RW	<code>21</code>	The width of the blur if a <code>BoxBlur</code> is used
<code>height</code>	<code>Number</code>	RW	<code>21</code>	The height if the blur if a <code>BoxBlur</code> is used

The size of the shadow is determined by the values of the `offsetX`, `offsetY`, and `radius` variables. When the `radius` is 0, the shadow has a sharp edge as shown on the left of Figure 20-10. In this case, the `offsetX` and `offsetY` values are both 15, so the shadow is offset by 15 pixels to the right of and below the top-left corner of the node, which gives the impression of a light source that is to the left of and above the top of the node. Negative values for the `offsetX` variable would be used for a light source to the right of the node, and negative `offsetY` values for a light source that is below the node.

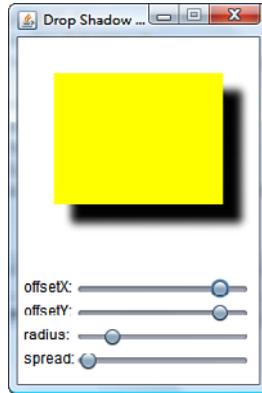


Figure 20-9 Configuring a `DropShadow` effect

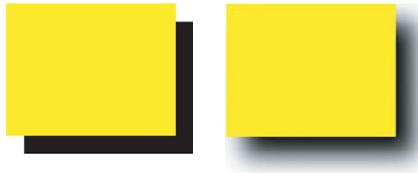


Figure 20-10 Effects of the `offsetX`, `offsetY`, and `radius` variable of the `DropShadow` effect

When the `radius` value is non-0, the edge of the shadow is blurred by blending pixels of the shadow color with those of the background color. The `radius` determines the size of this blurred area. Increasing the `radius` value makes the blurred region, and the size of the shadow, larger, as shown on the right of Figure 20-10. As you can see, the blurring fades out with increasing distance from the original shadow area. The `radius` value can be anywhere between 0 and 63, inclusive.

By default, the blurred area starts with the shadow color on its inside edge and progresses to the background color on its outside edge. If you want, you can arrange for a larger part of the blurred area to have the shadow color, resulting in a larger, darker shadow. You do this by setting the `spread` value, which ranges from 0.0, the default, to 1.0. This value represents the proportion of the blurred area into which the shadow color creeps. On the right side of Figure 20-10, the `spread` variable has value 0, and you can see that the shadow gets lighter very rapidly as you move your eyes away from the edge of the rectangle. On the left side of Figure 20-11, the `spread` variable has been set to 0.5. Now you can see that the darker region of the shadow has increased in size as it encroaches into the blurred area. On the right of Figure 20-11, the `spread` is at of 0.9, and you can see that almost all the blurred area ... has been taken over by the shadow color.

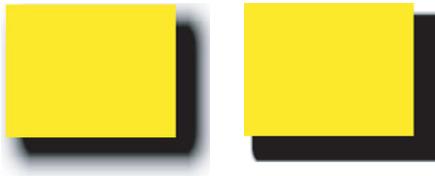


Figure 20-11 The effect of the spread variable

The idea of the `spread` variable is to allow a proper emulation of what would happen if you moved a light source quite close up to the node. A light source nearby would cause a wide shadow, corresponding to a larger blurred area, but it would also cause the darker part of the shadow to increase in size. You simulate the former effect by increasing the blur `radius` and the latter by increasing the `spread`.

InnerShadow

`InnerShadow` is very similar to `DropShadow`, the difference being that the shadow is *inside* the boundaries of the node to which it is applied, rather than outside. This gives the impression of depth within the node, because it appears to have built-up sides. The variables of this class, which are listed in Table 20-6, are almost the same as those of `DropShadow`.

You can see an example of this effect in Figure 20-12. This screenshot shows the result of running the code in the file `javafxeffects/InnerShadow1.fx`. As with the `DropShadow` examples, you can use the sliders to vary the effect parameters and see the results. The `choke` variable is equivalent to the `spread` variable of the `DropShadow` class.

Table 20-6 Variables of the `InnerShadow` Class

Variable	Type	Access	Default	Description
<code>blurType</code>	<code>BlurType</code>	RW	<code>THREE_PASS_BOX</code>	The type of blur to be used
<code>color</code>	<code>Color</code>	RW	<code>Color.BLACK</code>	The color to be used for the shadow
<code>offsetX</code>	<code>Number</code>	RW	0.0	The x-offset of the shadow
<code>offsetY</code>	<code>Number</code>	RW	0.0	The y-offset of the shadow
<code>radius</code>	<code>Number</code>	RW	10	The radius of the blur effect if a <code>GaussianBlur</code> is used
<code>width</code>	<code>Number</code>	RW	21	The width of the blur if a <code>BoxBlur</code> is used
<code>height</code>	<code>Number</code>	RW	21	The height of the blur if a <code>BoxBlur</code> is used

Table 20-6 Variables of the `InnerShadow` Class (Continued)

Variable	Type	Access	Default	Description
<code>choke</code>	Number	RW	0.0	The proportion of the radius (or box for <code>BoxBlur</code>) over which the shadow is fully opaque (see text)

Shadow

The `Shadow` effect produces a single-colored and blurred shadow from the node or input effect on which it operates. The extent of the blur depends on the value of the

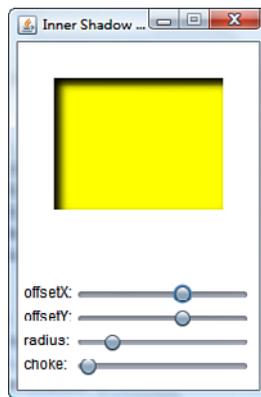


Figure 20-12 Configuring an `InnerShadow` effect

radius, which is one of the three variables that control this effect, all of which are listed in Table 20-5 on page 661.

You can see an example of this effect as applied to some text in Figure 20-13. You can experiment with different `radius` values by running this example, which you'll find in the file `javafxeffects/Shadow1.fx`:

```
Text {
    textOrigin: TextOrigin.TOP
    x: 30 y: 30
    content: "JavaFX Shadow Effect"
    font: Font { size: 24 }
    effect: Shadow {
        color: Color.BLUE
    }
}
```

```

        radius: bind radiusSlider.value
    }
}

```



Figure 20-13 The **Shadow** effect

Unlike the other two shadow effects, this one replaces its input instead of augmenting it, so the original text node is not drawn.

Bloom

The `Bloom` effect adds a glow to those areas of its input that are made up of pixels for which the luminosity value is above a given threshold. This effect has only two controlling variable, which are listed in Table 20-7.

Table 20-7 Variables of the `Bloom` Class

Variable	Type	Access	Default	Description
<code>input</code>	<code>Effect</code>	RW	null	The input to this effect.
<code>threshold</code>	<code>Number</code>	RW	0.3	The luminosity above which the glow effect will be applied, from 0.0 (all pixels will glow) to 1.0. (No pixels will glow.)

The luminosity of a pixel is a measure of how bright it seems to the human eye. You can see an example of this effect in Figure 20-14, which shows the `Bloom` effect applied to an `ImageView` node³:

```

ImageView {
    image: Image { url: "{__DIR__}image1.jpg" }
    effect: bloom = Bloom {

```

³ You'll find this code in the file `javafxeffects/Bloom1.fx`.

```
    threshold: bind (thresholdSlider.value as Number) / 10  
  }  
}
```

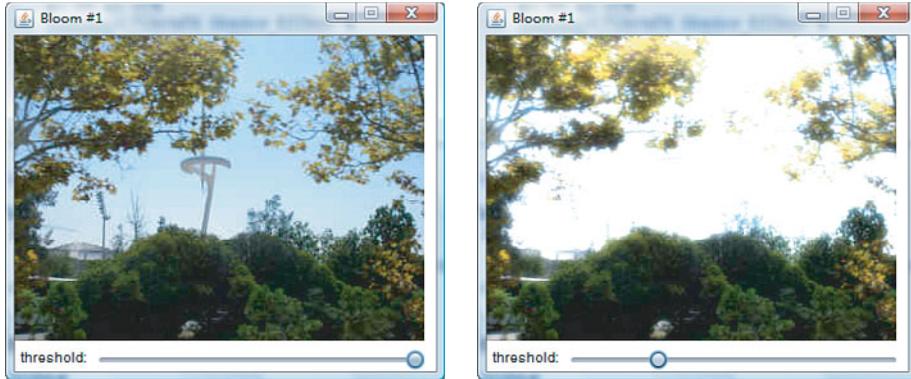


Figure 20-14 The **Bloom** effect

In the image on the left, the threshold value is 1.0. Because no pixel has a luminosity that is greater than 1.0, what you see here is the original image. On the right of the figure, the slider has been moved so that the threshold is now set to 0.3. The blue regions of the image, in particular the sky, are now noticeably brighter. Notice that this effect spills over onto adjacent pixels so that the leaves on the trees near the top of the image have also been brightened.

Glow

Glow is very similar to **Bloom**, except that the controlling parameter works in the reverse order. The glow effect makes bright pixels appear brighter. The more of the effect that you apply, as determined by the value of the `level` variable, the brighter those pixels appear. The two variables that control this effect are listed in Table 20-8.

Table 20-8 Variables of the **Glow** Effect

Variable	Type	Access	Default	Description
input	Effect	RW	null	The input to this effect.
level	Number	RW	0.3	Controls the intensity of the glow effect. 0.0 gives no glow; 1.0 gives maximum glow.

You'll find an example that allows you to vary the `level` parameter in the file `javafxeffects/Glow1.fx`. The following extract from that file shows how the glow effect is applied to a node:

```
ImageView {  
    image: Image { url: "{__DIR__}image1.jpg" }  
    effect: Glow {  
        level: bind (levelSlider.value as Number) / 10  
    }  
}
```

Figure 20-15 shows this effect applied to the same image as that used in our discussion of bloom in the previous section. In the image on the left of the figure, the `level` variable is 0, so no glow is applied. In the image on the right, the level is set to 0.6, and the result is almost exactly the same as the result of applying a small amount of bloom to the image, which you can see at the bottom of Figure 20-14. To apply more glow in this example, you move the slider farther to the right, whereas to apply more bloom in the example in the previous section, you moved it farther to the left.

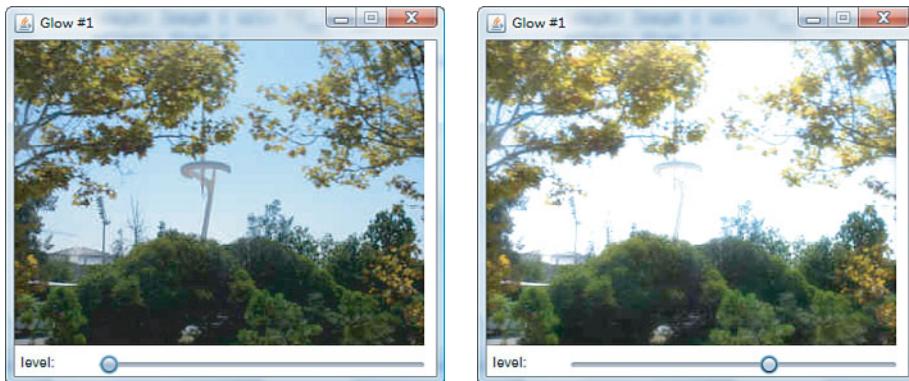


Figure 20-15 The **G**low effect

Identity

The `Identity` effect is a little different from the effects that you have seen so far—its sole purpose is to allow an `Image` object to be used as the input to another effect. It is always linked to a node, but that node does not appear in the scene; the result of applying one or more effects to the source image is seen instead. Table 20-9 lists the variables that control the behavior of this class.

The simplest way to explain how these variables work is by using an example. The following code, which you'll find in the file `javafxeffects/Identity1.fx`, applies a `GaussianBlur` effect to an image and places it in the `Scene`.

Table 20-9 Variables of the Identity Class

Variable	Type	Access	Default	Description
source	Image	RW	null	The source Image
x	Number	RW	0	The x coordinate of the Image relative to the source Node
y	Number	RW	0	The y coordinate of the Image relative to the source Node

```

1  Stage {
2      title: "Identity #1"
3      scene: Scene {
4          width: 380
5          height: 280
6          var image = Image { url: "{__DIR__}image1.jpg" }
7          content: [
8              Circle {
9                  centerX: 100 centerY: 100
10                 effect: GaussianBlur {
11                     input: Identity {
12                         source: image
13                         x: 10 y: 10
14                     }
15                 radius: 10
16             }
17         ]
18     }
19 }
20 }
```

The result of running this code is shown in Figure 20-16.



Figure 20-16 The Identity effect

The `Identity` effect on lines 11 to 14 converts the image to an `Effect` that is then used as the input to the `GaussianBlur`, resulting in a blurred version of the image. These two effects are both linked with a circle, but because the circle is not an input to either of the effects, it does not influence the output, and the blurred image appears instead of it. The only property of the circle that *is* inherited is its coordinate system, which, in this case, is the same as the coordinate system of the scene. The `x` and `y` variables of the `Identity` effect, which are set on line 13, determine where its output would be drawn relative to the circle's coordinate system. In this case, these values cause the image to be placed a little to the right of and below the coordinate origin.

The result of an `Identity` effect, like that of the `Flood` effect that is described in the next section, is often used as one of the inputs to a `Blend` effect, which is discussed later in this chapter.

Flood

Like `Identity`, the purpose of the `Flood` effect is to create an input to another effect, in this case a rectangular area filled with a `Paint` or a solid color. The variables that determine the fill color and the bounds of the filled area are listed in Table 20-10.

Table 20-10 Variables of the `Flood` Class

Variable	Type	Access	Default	Description
<code>paint</code>	<code>Paint</code>	RW	<code>Color.RED</code>	The <code>Paint</code> used to flood the area
<code>x</code>	<code>Number</code>	RW	0	The <code>x</code> coordinate of the filled area relative to the source <code>Node</code>
<code>y</code>	<code>Number</code>	RW	0	The <code>y</code> coordinate of the filled area relative to the source <code>Node</code>
<code>width</code>	<code>Number</code>	RW	0	The width of the area to be filled
<code>height</code>	<code>Number</code>	RW	0	The height of the area to be filled

The coordinates and lengths are specified in the coordinate system of the node that this effect is linked with. As with `Identity`, the node itself is replaced in the scene by the result of the effect. The code in the file `javafxeffects/Flood1.fx` uses the `Flood` effect to fill an area with a solid blue color and then applies a `MotionBlur`, giving the result shown in Figure 20-17.

ColorAdjust

The `ColorAdjust` effect produces an output that is the result of adjusting some or all the hue, saturation, brightness, and contrast values of its input. The input may be either

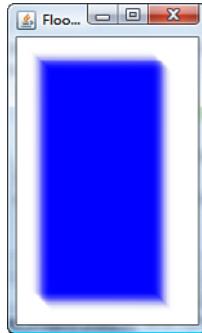


Figure 20-17 The **Flood** effect

another effect or a node of any kind, but most commonly an image in an `ImageView` object. The variables of this class are listed in Table 20-11.

You can experiment with this effect by running the example in the file `javafxeffects/ColorAdjust.fx`, which binds a slider to each of the hue, saturation, brightness, and contrast variables of a `ColorAdjust` object that is associated with an `ImageView` node. The values of the hue, saturation, and brightness sliders range from `-1.0` on the left to `1.0` on the right, while the contrast slider provides the value `0.25` in its

Table 20-11 Variables of the `ColorAdjust` Class

Variable	Type	Access	Default	Description
<code>input</code>	<code>Effect</code>	<code>RW</code>	<code>null</code>	The input to this effect.
<code>hue</code>	<code>Number</code>	<code>RW</code>	<code>0.0</code>	The amount by which the hue of each pixel should be adjusted, in the range <code>-1.0</code> to <code>1.0</code> . Value <code>0</code> leaves the hue unchanged.
<code>saturation</code>	<code>Number</code>	<code>RW</code>	<code>0.0</code>	The amount by which the saturation of each pixel should be adjusted, in the range <code>-1.0</code> to <code>1.0</code> . Value <code>0</code> leaves the saturation unchanged.
<code>brightness</code>	<code>Number</code>	<code>RW</code>	<code>0.0</code>	The amount by which the brightness of each pixel should be adjusted, in the range <code>-1.0</code> to <code>1.0</code> . Value <code>0</code> leaves the brightness unchanged.
<code>contrast</code>	<code>Number</code>	<code>RW</code>	<code>1.0</code>	The amount by which the contrast should be adjusted, in the range <code>0.25</code> to <code>4</code> . Value <code>1</code> leaves the contrast unchanged.

minimum position and 4.0 at its maximum position. On the left of Figure 20-18, you can see the result of applying almost the maximum brightness, and on the right you see the result of applying the maximum contrast.

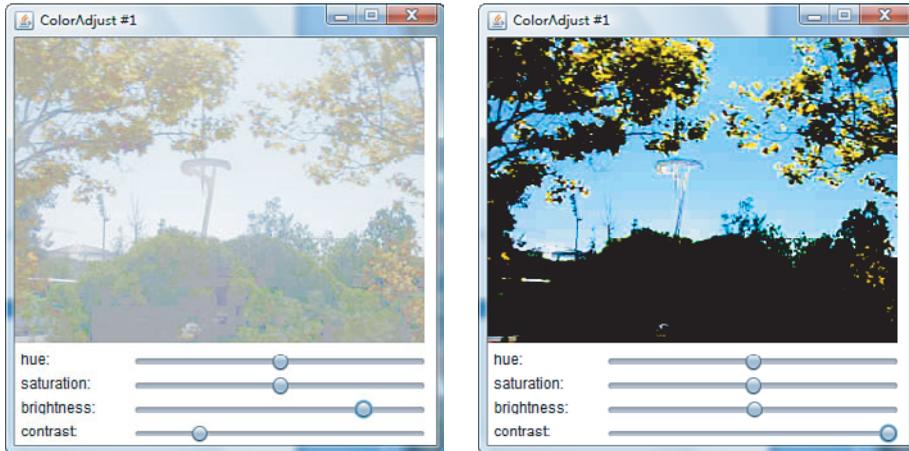


Figure 20-18 The `ColorAdjust` effect

InvertMask

The `InvertMask` effect takes another `Effect` as its input and produces a result in which all the transparent pixels from the input are opaque and all the opaque pixels are transparent. The output is typically used as one of the inputs to a `Blend` effect, which is discussed later in this chapter. The variables of the `InvertMask` class are listed in Table 20-12.

Table 20-12 The Variables of the `InvertMask` Class

Variable	Type	Access	Default	Description
<code>input</code>	<code>Effect</code>	RW	<code>null</code>	The input to this effect
<code>pad</code>	<code>Number</code>	RW	<code>0</code>	The padding to add to the sides of the resulting image

Reflection

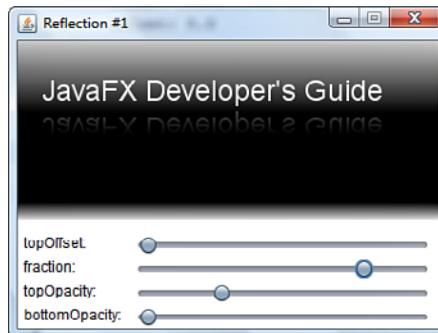
The `Reflection` effect provides an easy way to create a reflection of a node or group of nodes. The variables that you can use to specify the required characteristics of the reflection are listed in Table 20-13.

Table 20-13 The Variables of the `Reflection` Class

Variable	Type	Access	Default	Description
<code>input</code>	Effect	RW	<code>null</code>	The input to this effect
<code>topOffset</code>	Number	RW	0	The distance between the bottom of the input and the beginning of the reflection
<code>fraction</code>	Number	RW	0	The fraction of the input that is used to create the reflection
<code>topOpacity</code>	Number	RW	0.5	The opacity used for the topmost row of pixels in the reflection
<code>bottomOpacity</code>	Number	RW	0	The opacity of the bottom row of pixels in the reflection

The example code in the file `javafxeffects/Reflection1.fx` allows you to experiment with different values of these variables. A typical result, which is equivalent to the following code, is shown in Figure 20-19.

```
Text {
    content: "JavaFX Developer's Guide"
    x: 20 y: 50
    fill: Color.WHITE
    font: Font { size: 24 }
    effect: Reflection {
        topOffset: 0
        fraction: 0.8
        topOpacity: 0.3
        bottomOpacity: 0.0
    }
}
```

Figure 20-19 The `Reflection` effect

The `topOffset` variable lets you set the distance between the source object, here the text “JavaFX Developer’s Guide” (and its reflection). Increasing this distance makes it

seem that the source is further away from the reflecting surface. In Figure 20-19, the `topOffset` value is 0, which places the reflection as close as possible to the original. In this case, the reflected text might seem to be farther away than it should be with this value—that is because of the descender on the letter *p*, which is the closest point of contact with the reflection.

The `fraction` variable determines how much of the source appears in the reflection. Typically, unless the reflecting surface is very shiny, you will not want the whole of the source object to be reflected. In Figure 20-19, the `fraction` variable has the value 0.8, so about 80% of the source is reflected.

The `topOpacity` and `bottomOpacity` values give the opacity of the reflection at its top and bottom extents, respectively. In Figure 20-19, the `topOpacity` has been set to 0.3 and `bottomOpacity` to 0.0.

SepiaTone

The `sepiaTone` effect is used to give images (or any group of nodes) an “Olde Worlde” look, as if they have been photographed by an old black-and-white camera, or washed out by the effects of exposure to sunlight over a long period. This effect provides only the two variables listed in Table 20-14.

Table 20-14 Variables of the `SepiaTone` Class

Variable	Type	Access	Default	Description
<code>input</code>	<code>Effect</code>	RW	<code>null</code>	The input to this effect
<code>level</code>	<code>Number</code>	RW	<code>1.0</code>	The level of this effect, from 0.0 to 1.0

The `level` variable determines the extent to which the image is affected. The example code in the file `javafxeffects/SepiaTone1.fx` creates a `Scene` containing an image and a slider that lets you vary the value of the `level` variable and observe the result. The screenshot on the left of Figure 20-20 has `level` set to the value 0.4, while the one on the right has `level` set to 1.0.

PerspectiveTransform

The `PerspectiveTransform` is a useful effect that you can use to create the impression of a rotation in the direction of the z-axis—that is, into and out of the screen. It operates by deforming a node or group of nodes by moving its corners to specified locations and relocating the other pixels in such a way that straight lines drawn on the original nodes are mapped to straight lines in the result. Unlike the affine transforms that you saw in Chapter 17, “Coordinates, Transforms, and Layout,” this effect does *not* guarantee that lines that are parallel in the source will be parallel in the result and, in fact, the perspective effect requires that some parallel lines be made nonparallel.

The variables that control the perspective effect are listed in Table 20-15.

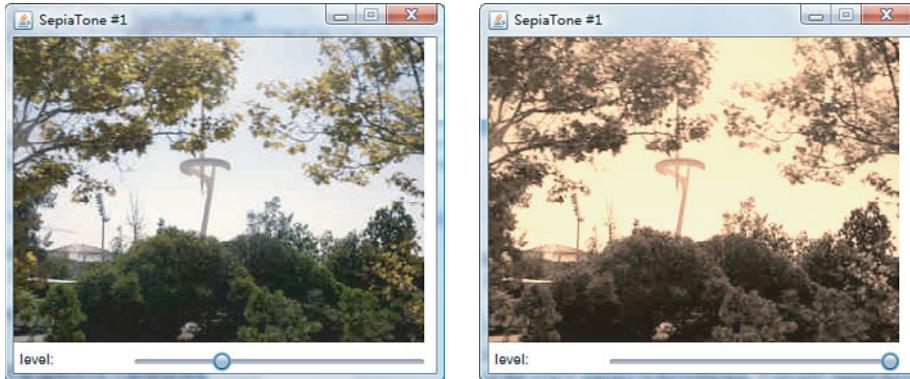


Figure 20-20 The `SepiaTone` effect

Table 20-15 Variables of the `PerspectiveTransform` Class

Variable	Type	Access	Default	Description
<code>input</code>	Effect	RW	<code>null</code>	The input to this effect.
<code>llx</code>	Number	RW	0	The <i>x</i> coordinate of the location to which the lower-left corner of the input is moved
<code>lly</code>	Number	RW	0	The <i>y</i> coordinate of the location to which the lower-left corner of the input is moved
<code>ulx</code>	Number	RW	0	The <i>x</i> coordinate of the location to which the upper-left corner of the input is moved
<code>uly</code>	Number	RW	0	The <i>y</i> coordinate of the location to which the upper-left corner of the input is moved
<code>lrx</code>	Number	RW	0	The <i>x</i> coordinate of the location to which the lower-right corner of the input is moved
<code>lry</code>	Number	RW	0	The <i>y</i> coordinate of the location to which the lower-right corner of the input is moved
<code>urx</code>	Number	RW	0	The <i>x</i> coordinate of the location to which the upper-right corner of the input is moved
<code>ury</code>	Number	RW	0	The <i>y</i> coordinate of the location to which the upper-right corner of the input is moved

To see what these variables represent, refer to Figure 20-1. Here, imagine that the image is mounted vertically and can rotate about its vertical axis, as shown by the white dashed line. In the figure, the black outline represent the view of the image after it has been rotated a few degrees so that the right edge has moved closer to the viewer and the

left edge farther away. This would cause the right edge to appear larger and the left edge correspondingly smaller, giving the impression of perspective.

You can use a `PerspectiveTransform` to create the rotated image from the original by moving the corners of the original to the new positions, as shown in Figure 20-21. The corner at the top left is the upper-left corner, and its position is given by the `u1x` and `u1y` variables. The corner at the top right is the upper-right corner, and its position is specified by the `urx` and `ury` variables, and so on.

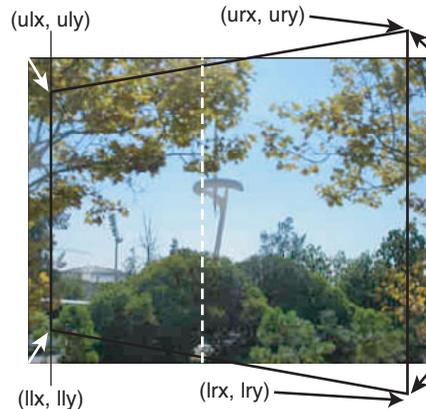


Figure 20-21 Illustrating the variables of the `PerspectiveTransform` class

It's easy to create a `PerspectiveTransform` that will rotate an image (or any other node or group) around a vertical axis that is a specified distance along its horizontal edge. It requires only a small amount of mathematics. We'll deal with the x and y coordinates separately, to make it easier to understand what is going on. The information needed to work out how to calculate the values of the x coordinates is shown in Figure 20-22.

Here, we are looking down at the image from above. The thick horizontal line, labeled APB , is the image before rotation, whereas the diagonal line, labeled $A'PB'$, is the image after it has been rotated through an angle (shown here as `angle`) about a pivot point, marked P , that is 11 pixels from its left side and 12 pixels from its right side. In this case, the pivot point is almost equidistant from the sides of the image, but the same calculation works even if this is not the case. The x -axis is shown at the bottom of the figure.

The x coordinate of the left side of the image after rotation is given by the distance AC , while the x coordinate of the right side is given by A . The distance AC is the same as $AP - CP$. Because AP has the value 11, elementary trigonometry gives the following:

$$AC = AP - CP = 11 - 11 * \cos(\text{angle})$$

Similarly,

$$AD = BP + PD = 11 + 12 * \cos(\text{angle})$$

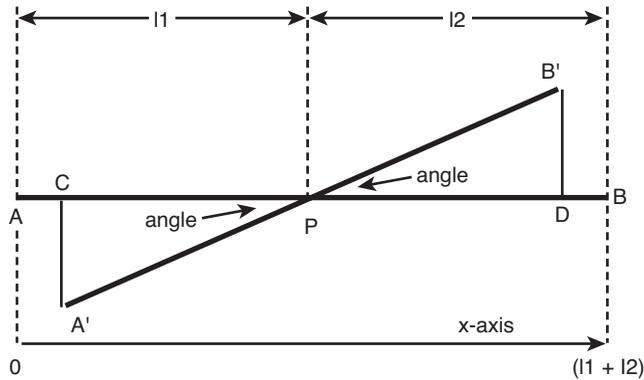


Figure 20-22 Rotating an image (top-down view)

AC is actually the value of both `ulx` and `llx`, while AD is the value that we need for `urx` and `ulx`. To make this simpler when using the `PerspectiveTransform`, we introduce two new parameters:

- `imgWidth`: The width of the image. This corresponds to the length AB and is equal to `l1 + l2`.
- `pivot`: The position of the pivot point along the line AB, as a ratio of `l1` to the total length AB. To place the pivot point in the center, set `pivot` to 0.5.

Given these parameters, we so far have the following `PerspectiveTransform`:

```
PerspectiveTransform {
    ulx: l1 - l1 * Math.cos(angle)
    uly: ?? // Not yet determined
    llx: l1 - l1 * Math.cos(angle)
    lly: ?? // Not yet determined
    urx: l1 + l2 * Math.cos(angle)
    ury: ?? // Not yet determined
    lrx: l1 + l2 * Math.cos(angle)
    lry: imgHeight ?? // Not yet determined
}
```

Now let's move on to the y coordinates. This part is slightly easier. Essentially, what we need to do is make the length of the side of the image that moves toward us larger and that of the side that moves away from us smaller. We can choose by how much we want each side to grow or shrink—the closer we are to the image, the more each side would grow or shrink. We'll make this a parameter of the transform and say that we want each side to grow or shrink by `htFactor` of its actual value at each end. That means, for example, that if the image is 100 pixels tall and we choose `htFactor` to be 0.2, the side of the

image that is nearer to us after the image has rotated through 90 degree will be larger by $0.2 * 100 = 20$ pixels at each end, or a total of 140 pixels tall. Similarly, the side that is farther away will shrink to 60 pixels in height.

Now refer to Figure 20-23. Here, we are looking at the image from the front again. The solid shape is the image after it has been rotated. The dashed vertical line is the axis of rotation, and the dashed extension that is outside the rectangle represents the maximum apparent height of the image when it has rotated through 90 degrees—that is, when it is edge-on to the viewer.

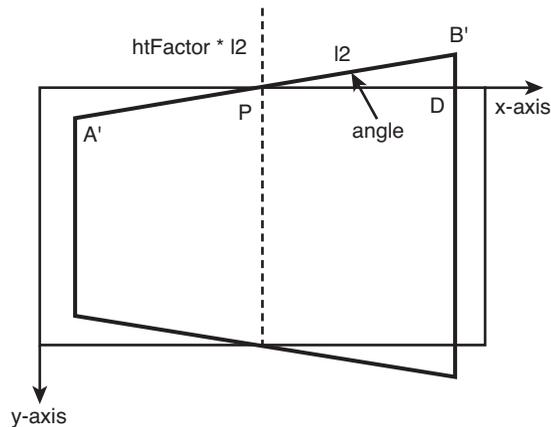


Figure 20-23 Rotating an image (front view)

In its current position, the y coordinate of the upper-right corner would be $-B'D$. This coordinate is negative because the y -axis runs along the top of the image, as shown. The length of $B'D$ is $l2 * \sin(\text{angle})$, but because we are limiting the maximum vertical extension of each side by $htFactor$, we use the value $htFactor * l1 * \sin(\text{angle})$ instead. Applying the same logic to each of the four corners gives us the following as the final transform, installed in an `ImageView` and with specific values assigned for the variables `pivot` and `htFactor`:

```
ImageView {
    translateX: bind (scene.width - imgWidth) / 2
    translateY: bind (scene.height - 30 - imgHeight) / 2
    image: image = Image { url: "{__DIR__}image1.jpg" }
    var angle = bind Math.toRadians(slider.value);
    var pivot = 0.5;
    var htFactor = 0.2;
    var l1 = bind pivot * imgWidth;
    var l2 = bind imgWidth - l1;
```

```

effect: bind PerspectiveTransform {
    ulx: 11 - 11 * Math.cos(angle)
    uly: htFactor * 11 * Math.sin(angle)
    llx: 11 - 11 * Math.cos(angle)
    lly: imgHeight - 11 * htFactor * Math.sin(angle)
    urx: 11 + 12 * Math.cos(angle)
    ury: -12 * htFactor * Math.sin(angle)
    lrx: 11 + 12 * Math.cos(angle)
    lry: imgHeight + 12 * htFactor * Math.sin(angle)
}
}

```

The file `javafxeffects/PerspectiveTransform1.fx` contains an example that incorporates this transform and provides a slider that allows you to vary the value of the `angle` variable from -90 degrees to $+90$ degrees. Figure 20-24 shows a couple of screenshots taken from this example with the image rotated by two different angular amounts. You can experiment with this example by changing the value of the `pivot` variable to get a rotation about a different point. Setting `pivot` to `0` causes a rotation around the left edge, while the value `1` gives rotation about the right edge.

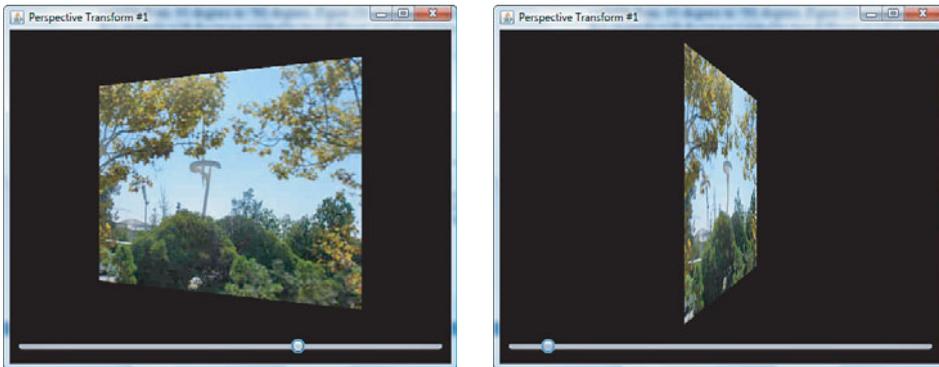


Figure 20-24 Examples of images rotated using a `PerspectiveTransform`

Note

You might be wondering why `PerspectiveTransform` is an effect and not one of the transforms discussed in Chapter 17. The reason is that it is not a true transform, in the sense that it does not affect the coordinate axes—it is just a visual effect. As a result of this, if you try to detect and act on mouse events from a node or group that has a `PerspectiveTransform` applied, you will not get reliable results because the coordinates in the event relate to the *untransformed* shape.

DisplacementMap

The `DisplacementMap` effect is, at first glance, the most complex of the effects that are provided by the JavaFX SDK, but it is also one of the most powerful. As its name suggests, this effect displaces pixels from their locations in the input image to different positions in the output image. Let's begin by listing the variables that you can use to parameterize the effect (see Table 20-16), and then we'll take a look at how they work.

Table 20-16 Variables of the `DisplacementMap` Class

Variable	Type	Access	Default	Description
<code>input</code>	<code>Effect</code>	RW	<code>null</code>	The input to this effect
<code>mapData</code>	<code>FloatMap</code>	RW	<code>Empty map</code>	The map that determines how input pixels are mapped to output pixels
<code>offsetX</code>	<code>Number</code>	RW	<code>0.0</code>	A fixed displacement along the x-axis applied to all pixel offsets
<code>offsetY</code>	<code>Number</code>	RW	<code>0.0</code>	A fixed displacement along the y-axis applied to all pixel offsets
<code>scaleX</code>	<code>Number</code>	RW	<code>1.0</code>	A scale factor applied to the map data along the x-axis
<code>scaleY</code>	<code>Number</code>	RW	<code>1.0</code>	A scale factor applied to the map data along the y-axis
<code>wrap</code>	<code>Boolean</code>	RW	<code>false</code>	Whether the displacement operation should wrap at the boundaries

How the `DisplacementMap` Effect Works

The reason for the apparent complexity of this effect is the equation that controls how the pixels are moved:

$$\text{dst}[x, y] = \text{src}[x + (\text{offsetX} + \text{scaleX} * \text{map}[x, y][0]) * \text{srcWidth}, \\ y + (\text{offsetY} + \text{scaleY} * \text{map}[x, y][1]) * \text{srcHeight}]$$

At first sight, this probably looks quite daunting, but in fact it turns out to be quite simple. Basically, it says each pixel in the output (here represented by the symbol `dst`) derives from a single pixel in the input (represented by `src`). The pixel value at coordinates (x, y) in the output is obtained from a source pixel whose coordinates are displaced from those of the destination pixel by an amount that depends on a value obtained from a map, together with some scale factors and an offset. The values `srcWidth` and `srcHeight` are respectively the width and height of the input source.

Let's start by assuming that the offset values are both 0 and the scale values are both 1. In this simple case, the equation shown above is reduced to this more digestible form:

```
dst[x, y] = src[x + map[x, y][0] * srcWidth,
               y + map[x, y][1] * srcHeight]
```

The map is a two-dimensional data structure that is indexed by the *x* and *y* coordinates of the destination point, relative to the top-left corner of the output image. Each element of this structure may contain a number of floats, which is why the class that holds these values is called a `FloatMap`. The `FloatMaps` that are used with a `DisplacementMap` must have two floats in each position,⁵ the first of which is used to control the displacement along the *x*-axis and the second the displacement along the *y*-axis. Suppose, for the sake of argument, that we have a `FloatMap` in which every element has the values $(-0.5, -0.5)$. In this case, the equation above can be written as follows:

```
dst[x, y] = src[x - 0.5 * srcWidth, y - 0.5 * srcHeight]
```

Now, you should be able to see that the pixel at any given position in the output is obtained from the source pixel that is a half of the width or height of the source away from it. If we assume that the source is 100 pixels square, we can make our final simplification:

```
dst[x, y] = src[x - 50, y - 50]
```

This says that the output pixel at any point comes from the source pixel that is 50 pixels above it and to its left. The reason for using `srcWidth` and `srcHeight` as multipliers is that the values in the map can then be encoded as fractions of the width and height of the input respectively and therefore would normally be in the range -1 to $+1$. A map value of -1 or $+1$ would move a point by the complete width or height of the input source.

A Simple Example

Let's look at how you would implement the example that you have just seen. You'll find the code in the file `javafxeffects/DisplacementMap1.fx`. Let's start by creating the map:

```
1   var image: Image = Image { url: "{__DIR__}image1.jpg" };
2   var imgWidth = image.width as Integer;
3   var imgHeight = image.height as Integer;
4   var map: FloatMap = FloatMap {
5       width: imgWidth
6       height: imgHeight
7   }
8
9   for (i in [0..

```

⁵ The `FloatMap` can have more than two floats in each position, but only the first two are used.

```

13     }
14 }

```

In this example, we are going to use an image as the input source, so we create a map that has the same dimensions as the image itself. The code on lines 4 to 6 declares the `FloatMap`, setting its dimensions from the width and height of the image. The nested loops on lines 9 to 14 initialize the `FloatMap`, assigning two samples for each element. Each sample has the value `-0.5`, which is the offset that we require. Note how these samples are installed:

```

map.setSample(i, j, 0, -0.50); // The x offset
map.setSample(i, j, 1, -0.50); // The y offset

```

`FloatMap` has several overloaded variants of the `setSample()` function that you can use. In the variant that we use here, the first two arguments are the `x` and `y` coordinates of the element, the third argument is the band number, and the fourth argument is the offset for that band. Band 0 is used for the `x`-offset and band 1 for the `y` offset.⁶

Now, here's the code that creates and uses the `DisplacementMap` effect:

```

var scene: Scene;
Stage {
    title: "DisplacementMap #1"
    scene: scene = Scene {
        width: 500
        height: 380
        fill: Color.BLACK
        content: [
            ImageView {
                translateX: bind (scene.width - imgWidth) / 2
                translateY: bind (scene.height - 30 - imgHeight) / 2
                image: image
                effect: DisplacementMap {
                    mapData: map
                }
            }
        ]
    }
}

```

As you can see, the effect is applied simply by creating a `DisplacementMap` based on the map data and installing it in an `ImageView` that contains the source image. We don't need to set the scale or offset values because we are using the defaults in this case. You can see the result in Figure 20-25.

⁶ The band numbers appear in the original equations. `map[x, y][0]` indicates the value in band 0 at the element in position (x, y) in the map.

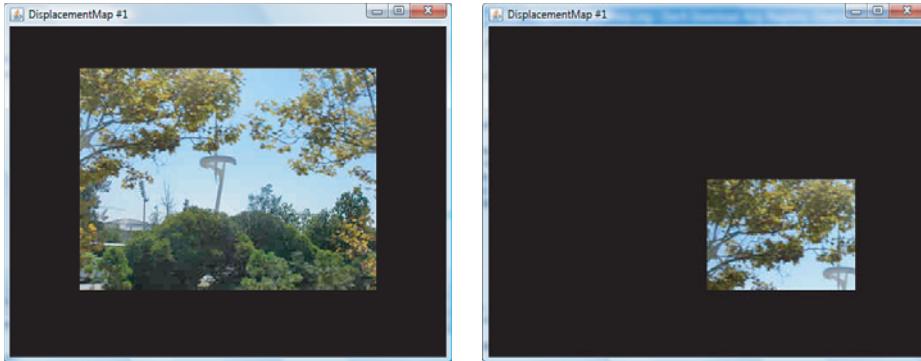


Figure 20-25 A simple `DisplacementMap` effect

The original image is shown on the left of the figure and the result of applying the `DisplacementMap` on the right. As you can see, the image has been moved halfway across and halfway down the area occupied by the source. It's easy to see why this has happened if you look back at the equation that describes this effect:

$$\text{dst}[x, y] = \text{src}[x - 0.5 * \text{srcWidth}, \\ y - 0.5 * \text{srcHeight}]$$

This says that the pixel at (x, y) comes from the source pixel that is half the source width to its left and half the source height above it. In other words, the image is moved down and to the right. To make this more obvious still, let's add some concrete numbers. We'll start by with the pixel at $(0, 0)$ in the destination image. According to the equation above, the color for this pixel comes from the pixel at $(0 - 0.5 * 340, 0 - 0.5 * 255) = (-170, -127)$. Because there is no such point, this pixel is not set, so this part of the destination is transparent. In fact, every pixel for which either of the source coordinates is negative will be transparent. The first pixel in the destination image that will not be transparent is the one at $(170, 127)$, which gets its color from the pixel at $(0, 0)$ in the source. By following this reasoning for any given pixel in the destination image, it is easy to see why the result of this effect is to move the source down and to the right, as shown in Figure 20-25.

The wrap Variable

You can achieve a slightly different effect to that shown above by setting the `wrap` variable of the `DisplacementMap` object to `true`. When you do this, the parts of the destination that would have been transparent because they correspond to points in the source image that are outside of its bounds (for example, those with negative coordinates) are populated by wrapping the coordinates modulo the size of the source. This means, for example, that the pixel at $(0,0)$, which should come from $(-170, -127)$ in the source, will actually come from $(-170 + 340, -127 + 255)$, or $(170, 128)$. You can see the overall effect of this by run-

ning the code in the file `javafxeffects/DisplacementMap2.fx`, which gives the result shown in Figure 20-26.

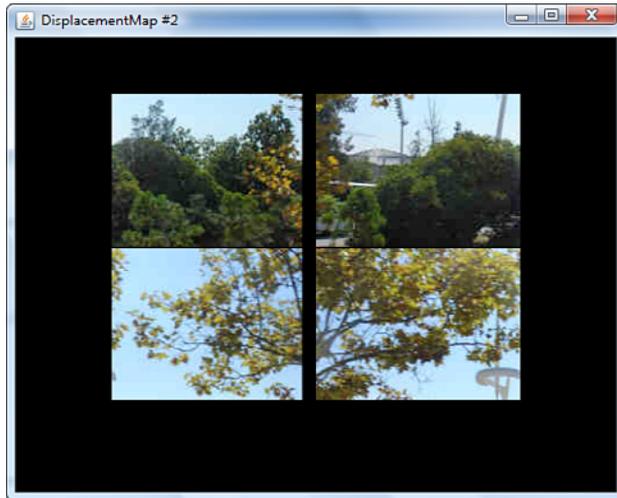


Figure 20-26 A `DisplacementMap` with wrap enabled

The offset Variables

Now that you have seen how the values in the map work in the simplest case, we'll make things a little more complex by adding back the `offsetX` and `offsetY` values. These values simply add a fixed offset to the distance between the destination pixel and the source pixel that supplies its color. Like the entries in the map, each offsets is scaled by the width or height of the source, as appropriate.

For example, let's suppose that we were to set the `offsetX` variable to `0.1` and leave `offsetY` as `0`. Then, using the same map as we did for the previous example, the equations that relate the source and destination pixel locations would now be as follows:

```
dst[x, y] = src[x + 0.1 * srcWidth - 0.5 * srcWidth,  
              y - 0.5 * srcHeight]
```

If, as before, the source is 340 pixels wide, this change would produce an additional offset of 34 pixels between the source and destination pixels.

You can see how the `offsetX` value works by running the code in the file `javafxeffects/DisplacementMap3.fx`. This example uses the same `FloatMap` as the previous one, but adds a slider that allows you to vary the `offsetX` value from `0` up to `1.0`, with the initial value being `0.0`. Initially, the result looks the same as before, because the `offsetX` value is still `0`—compare the image on the left of Figure 20-27 with that on the right of Figure 20-25 to see that this is the case.

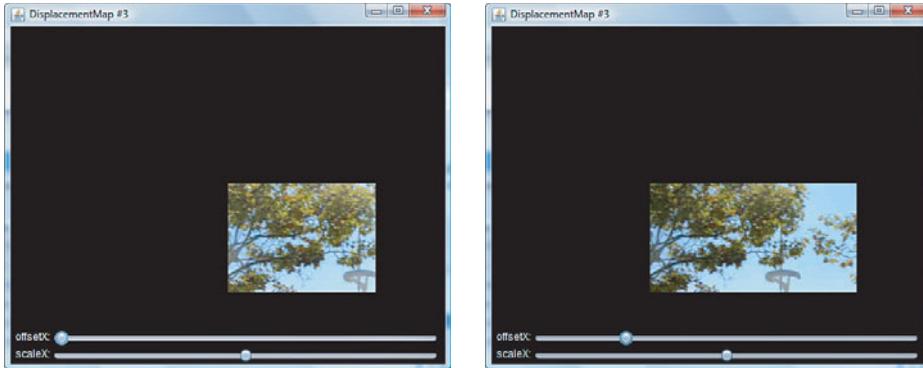


Figure 20-27 Varying the `offsetX` value of a `DisplacementMap` effect

Now if you move the offset slider to the right, you will see that the output image moves to the left. This is the offset at work. The farther you move the slider to the right, the more the result shifts to the left. The same effect would be seen along the `y`-axis if we had added a slider that allowed you to vary the `offsetY` value.

The `scale` Variables

The `scaleX` and `scaleY` variables are multipliers that are applied to the values from the `FloatMap`. If you use a `scaleX` value that is greater than 1, you make the offset between the source and destination pixels larger than that specified in the map. A `scaleX` value of 2 would double the offsets specified in the map. Similarly, if you use a value that is less than 1, the offset gets smaller. It is also possible to use a negative value, which would reverse the effect of the map.

The code in `javafxeffects/DisplacementMap3.fx` also includes a slider that lets you change the value of the `scaleX` variable over the range 0 to 2, with 1 as its initial value. If you move the slider, you will find that the output image also moves to reflect the magnified or reduced offset values. In this case, because every entry in the map has the same value, the effect is very similar to that obtained by changing the `offsetX` value, but this is not always the case, as you'll see later in this section.

Using the `DisplacementMap` to Create a Warp

The example that we have been using has the same value in every element of the map. This is a rather unusual case and it doesn't produce a very interesting effect. In this section, we'll take a look at how to create a warp effect by populating the map with values that depend on their position in the map. The completed effect is shown in Figure 20-28.

As you can probably tell, the effect is produced by simulating the effect of a wave moving in the direction of the `y`-axis, which causes successive pixel rows to be displaced



Figure 20-28 Using `DisplacementMap` to create a warped effect

to the left or right of their initial positions. As there is no movement of any kind in the y direction, you can immediately conclude that all the y values (those in the second band) in the map are 0. The wave effect is, in fact, a sine wave. Here's the code that populates the map⁷:

```
1    var image: Image = Image { url: "{__DIR__}image1.jpg" };
2    var imgWidth = image.width as Integer;
3    var imgHeight = image.height as Integer;
4    var map: FloatMap = FloatMap {
5        width: imgWidth
6        height: imgHeight
7    }
8
9    for (i in [0..<map.width]) {
10       for (j in [0..<map.height]) {
11           var value = (Math.sin(j/30.0 * Math.PI)/10;
12           map.setSample(i, j, 0, value);
13       }
14    }
```

⁷ You'll find this code in the file `javafxeffects/DisplacementMap4.fx`.

The part that does all the interesting work is on line 11. It is obvious that this is creating a sine wave by supplying the horizontal displacement (in the first band of the map) for each row of the input source based on the value of the `Math.sin()` function. The value of this function varies from 0 at 0 radians to 1 at $\pi/2$ radians, back to zero at π radians, to -1 at $3\pi/2$ radians, and then back to zero at 2π radians, and so on. In the inner loop, the value represents the pixel row. We divide it by 30 and multiply it by π so that we get a complete wave over the space of 30 pixels. If you make this number larger, you will find that the wave spaces out more. This code would place values ranging from $+1$ to -1 in every element of the map. Remembering that these offsets are multiplied by the width of the source, this would mean that the image would be distorted by up to its full width. To reduce the distortion, we divide every value by 10, so we end up with values in the range -0.1 to $+0.1$. That's all we need to do to create a warp effect.

If you run the code in the file `javafxeffects/DisplacementMap4.fx`, you can use the offset and scale sliders to change the parameters of the `DisplacementMap`. Notice that changing the scale increases or decreases the amplitude of the sine wave, which results in more or less distortion.

Blending

Blending is the process of combining two pixels that would occupy the same space to produce a third value that is actually placed at that space. Blending can be used to determine what should be seen in a region where two nodes overlap or where two effects are applied to a node. You can use blending either as an effect or as a mode that controls the drawing of overlapping nodes in a group or container.

The Blend Effect

The `Blend` effect combines two inputs and produces a result that depends on the selected blend mode. The variables of the `Blend` class are listed in Table 20-17.

Here's an example that demonstrates how to construct a `Blend` effect. This code is extracted from the file `javafxeffects/BlendEffect1.fx`, which you can run to try out all the available blend modes:

```
var image1 = Image { url: "{_DIR__}image1.jpg" };
var image2 = Image { url: "{_DIR__}image2.jpg" };

ImageView {
  x: 30
  y: 30
  image: image1
  effect: Blend {
    mode: BlendMode.ADD
    topInput: Identity {
      x: 150
      y: 150
      source: image2
    }
  }
}
```

Table 20-17 Variables of the Blend Class

Variable	Type	Access	Default	Description
mode	BlendMode	RW	SRC_OVER	The mode that determines how pixels from the two inputs are combined to produce the resulting pixel.
topInput	Effect	RW	null	The top input to this effect. If this is null, the node to which the effect is applied is used.
bottomInput	Effect	RW	null	The bottom input to this effect. If this is null, the node to which the effect is applied is used.
opacity	Number	RW	1.0	The opacity applied to the top input before blending.

Here, the bottom input is the `ImageView` itself (because the `bottomInput` variable is null, so the node itself becomes the input), while the top input is the output of an `Identify` effect applied to an `Image`. The second image is placed 150 pixels below and to the right of the `ImageView`, giving the result shown in Figure 20-29.

As you can see, the result of this effect is the union of the two images. There is a significant area of overlap between the two images, and in this region, their pixels are combined according to the unique set of rules that apply to the selected blend mode. There are 19 different modes, all defined as constants in the `BlendMode` class. You will find the details of each mode in the documentation for the `BlendMode` class. In Figure 20-29, `BlendMode.ADD` has been used. This adds all the color and alpha components from the two pixels to produce the result pixel. For example, if the RGBA values for two pixels were (0.6, 0.2, 0.3, 0.4) and (0.5, 0.6, 0.1, 0.5), the value of the resulting pixel would be (1.0, 0.8, 0.4, 0.9). Notice that the value of each channel is limited to 1.0, which is why the result of combining the red channels in this example is 1.0 rather than 1.1.

By selecting different values from the combo box at the top of the scene, you can see how each mode operates. Of particular interest are the `SRC_ATOP`, `SRC_IN`, `SRC_OUT` and `SRC_OVER` modes. In these modes, the “source” is the top input. You can see the results of applying `SRC_ATOP` mode on the left of Figure 20-30 and `SRC_IN` mode on the right.

The `SRC_ATOP` mode keeps all the bottom input plus that part of the top input that overlaps it. In the overlap area, only the top input is painted. By contrast, `SRC_IN` keeps only that part of the top input that overlaps with the bottom input, and everything else (including all the bottom input) is lost.

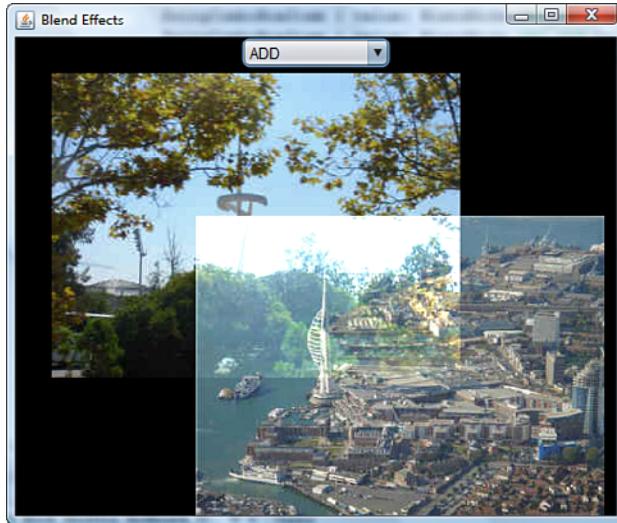


Figure 20-29 Using the `Blend` effect

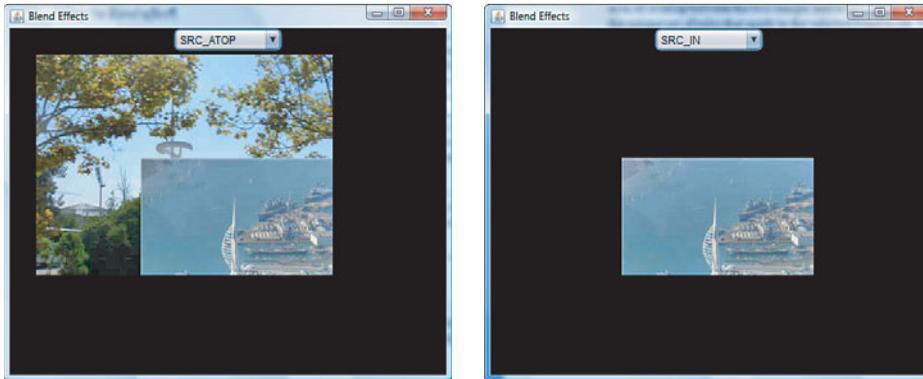


Figure 20-30 The `SRC_ATOP` and `SRC_IN` blend modes

The Group Blend Mode

A blend mode can be applied to a group (or a container, because `Container` is a subclass of `Group`) by setting its `blendMode` variable to one of the constants defined by the `BlendMode` class. The blend mode determines how the pixels in the areas in which there are overlapping nodes are constructed from those of the nodes themselves. By default, and in all the examples that you have seen so far, the `blendMode` variable has the value `BlendMode.SRC_OVER`, which causes the node in front to be drawn over those that are behind it.

The following code, from the file `javafxeffects/BlendGroup1.fx`, allows you to see how each of the possible blend modes operates when applied to the nodes of a group:

```

1   var image1 = Image { url: "{__DIR__}image1.jpg" };
2   var image2 = Image { url: "{__DIR__}image2.jpg" };
3   var scene: Scene;
4   Stage {
5       title: "Blend Group"
6       scene: scene = Scene {
7           var modeCombo: SwingComboBox;
8           var mode = bind modeCombo.selectedItem.value
9                   as BlendMode;
10          width: 500
11          height: 400
12          fill: Color.BLACK
13          content: [
14              modeCombo = SwingComboBox {
15                  translateX: bind (scene.width
16                      - modeCombo.layoutBounds.width) / 2
17                  editable: false
18                  items: [
19                      SwingComboBoxItem {
20                          value: BlendMode.ADD
21                          text: "ADD"
22                      }
23                      // Further items not shown here
24                  ]
25                  selectedIndex: 0
26              }
27          Circle {
28              centerX: 100
29              centerY: 100
30              radius: 80
31              fill: Color.YELLOW
32          }
33          Group {
34              blendMode: bind if (mode != null) mode
35                          else BlendMode.ADD
36              content: [
37                  ImageView {
38                      x: 30
39                      y: 30
40                      image: image1
41                  }
42                  ImageView {
43                      x: 150

```

```

45                                     y: 150
46                                     image: image2
47                                 }
48                             ]
49                         }
50                     ]
51                 }
52             }

```

This code creates a scene containing a combo box that allows a `BlendMode` to be selected, a circle, and a group containing two overlapping `ImageView`s. On the left of Figure 20-31, you can see the result of applying the default mode, which is `SRC_OVER`. On the right of the figure, the `MULTIPLY` mode is used.

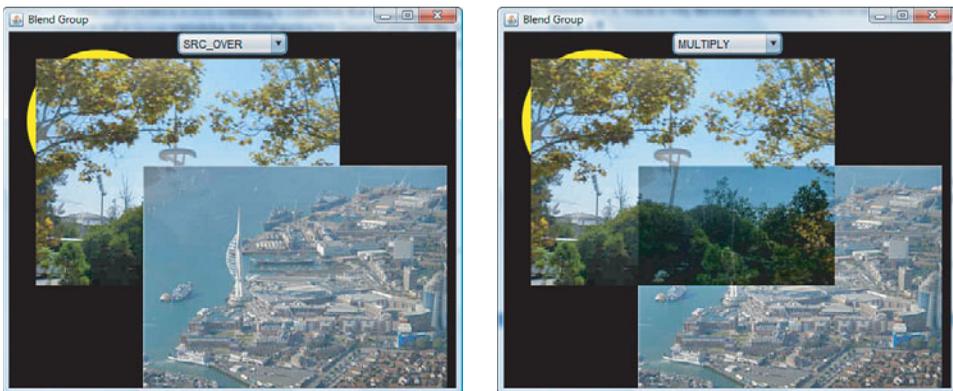


Figure 20-31 Using blend modes in a group

As you can see, these two modes have different effects on the pixels in the area of overlap between the two images. Note, however, that the blend mode has no effect on the region in which the upper-left `ImageView` overlaps the circle, because the circle is not in the group and therefore is not subject to the blend mode.

Lighting

The final effect that we are going to look at is called `Lighting`. As its name suggests, it allows you to specify how a node or a group should be lit. Three different types of lighting can be used, all of which are represented by classes in the

`javafx.scene.effect.light` package. The `Lighting` class itself, like all the other effects, is in the `javafx.scene.effect` package.

To define the lighting for a node or group, you create an instance of the `Lighting` class and install it in the `effect` variable of that node or group. Using lighting will make your scenes look more three-dimensional than they otherwise would, as you'll see in the examples in this section. The variables of the `Lighting` class are listed in Table 20-18.

Table 20-18 Variables of the `Lighting` Class

Variable	Type	Access	Default	Description
<code>light</code>	<code>Light</code>	RW	<code>DistantLight</code>	The type of light to be used.
<code>bumpInput</code>	<code>Effect</code>	RW	<code>null</code>	The bump map to be applied.
<code>contentInput</code>	<code>Effect</code>	RW	<code>null</code>	The input to this effect, which is the target node itself if this value is <code>null</code> .
<code>diffuseConstant</code>	<code>Number</code>	RW	<code>1.0</code>	Determines how much diffuse light is reflected from the surface, in the range 0 to 2, inclusive.
<code>specularConstant</code>	<code>Number</code>	RW	<code>0.3</code>	Determines how much specular light is reflected from the surface, in the range 0 to 2, inclusive.
<code>specularExponent</code>	<code>Number</code>	RW	<code>20</code>	Determines how shiny the surface appears to be in the range 0 to 40, inclusive.
<code>surfaceScale</code>	<code>Number</code>	RW	<code>1.5</code>	Determines the height assigned to pixels in the source, based on their opacity. Valid range is 0 to 10, inclusive.

The `light` variable determines the type of lighting required, which must be one of `DistantLight` (which is the default), `PointLight`, and `SpotLight`. The other variables set the characteristics of the surface that will be lit and will be explained in the rest of this section by reference to examples.

The surfaceScale Variable

To create a 3D effect when lighting a 2D surface, the opacity value of each pixel is used as a guide to how “high” that pixel should appear to be when lit. Transparent pixels appear to be the lowest, while fully opaque pixels appear to be raised up from the surface. This effect can be increased or decreased by using the `surfaceScale` variable. Values that are greater than 1 cause the effect to be increased, while values between 0 and 1 cause it to be decreased. The same effect can also be seen at the edges of shapes.

The following code, which comes from the file `javafxeffects/SurfaceScale.fx`, shines a distant light on a rectangle. The details of the lighting are not important right now, but you’ll see that the `surfaceScale` variable is bound to a slider. If you run this example, you can see the effect of varying the `surfaceScale` variable over its full range:

```
Rectangle {
    x: 20
    y: 20
    width: 100
    height: 100
    fill: Color.YELLOW
    effect: Lighting {
        light: DistantLight {
            azimuth: 0
            elevation: 30
        }
    }
    surfaceScale: bind (scaleSlider.value as Number) / 10
}
}
```

You can see how the `surfaceScale` value is used by comparing the two screenshots shown in Figure 20-32.

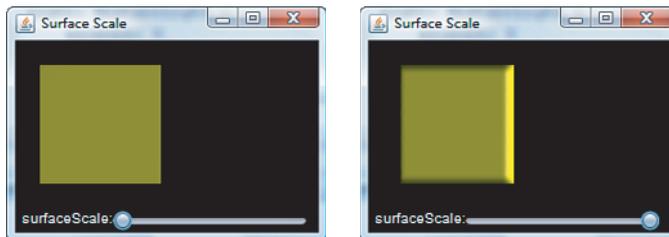


Figure 20-32 The `surfaceScale` variable of the **Lighting** effect

On the left of the figure, the `surfaceScale` value is 0, so there is no 3D effect at all. On the right, `surfaceScale` has its maximum possible value, and now you can see that the center of the rectangle appears to be raised up above its edges. The higher the `surfaceScale` value, the higher the center will appear to be.

The Bump Map

You can add additional surface relief by creating a *bump map* and installing it in the `bumpInput` variable of the `Lighting` effect. If this variable is `null`, the node on which the effect is applied is itself used to generate a bump map, which is what causes the 3D effect at the edges that you saw in Figure 20-32.

The bump map is just an `Effect`, which supplies pixels from which the relief of the lit surface is calculated. As before, the apparent height of a pixel on the lit shape depends on the opacity of the corresponding pixel of the bump map. This effect is affected by the value of the `surfaceScale` variable, as described in the preceding section.

A common way to specify a bump map is to create an image, set the opacity to reflect the contours that you want to appear in the finished result, and then turn it into an effect by using the `Identity` class that we discussed earlier in this chapter. The code in the file `javafxeffects/BumpMap.fx` shows how to apply a bump map in the form of an image file:

```

1    var logo = Image { url: "{__DIR__}javafxlogo.gif" };
2    Stage {
3        title: "Bump Map"
4        scene: Scene {
5            width: 240
6            height: 140
7            fill: Color.BLACK
8            content: [
9                Rectangle {
10                   x: 20
11                   y: 20
12                   width: 200
13                   height: 100
14                   fill: Color.YELLOW
15                   effect: Lighting {
16                       light: DistantLight {
17                           azimuth: 90
18                           elevation: 25
19                       }
20                       bumpInput: Identity {
21                           source: logo
22                           x: 30
23                           y: 40
24                       }
25                   }
26               }
27           ]
28       }
29   }

```

The code on line 1 loads the bump image from a file called `logo.gif` that is in the same directory as the script file. This image is then converted to an `Effect` by the code on lines 20 to 24, and placed appropriately relative to the rectangle node to which the lighting effect is applied by using the `x` and `y` variables of the `Identity` class. The lighting effect itself is a `DistantLight`, to which the image is supplied as the bump map on line 20.

The image that is used as the bump map is shown on the left of Figure 20-33. The image consists of the word `JavaFX` in black text on a white background. The white background is actually completely transparent, while the black text is completely opaque. You can see the effect of the bump map on the right of Figure 20-33, where the lighting effect causes the word `JavaFX` to appear to be raised above the surface of the `Rectangle`.



Figure 20-33 Using a bump map with a **Lighting** effect

You can apply the same effect to any node, including an `ImageView`, where you can use it to create the appearance of a watermark within the image.

DistantLight

The `DistantLight` class is used when you want to apply a more-or-less uniform light to a node or group. Depending on where the light source is, you may see some shadows, but you will not see reflections of the type that are a characteristic of `PointLight` and `SpotLight`, which are discussed in the sections that follow. `DistantLight`, `PointLight`, and `SpotLight` are all derived from the base class `javafx.scene.effect.light.Light`, which has a single variable called `color` (of type `Color`) that specifies the color of the light to be used, which is white by default. The other variables of the `DistantLight` class specify the position of the light source and are listed in Table 20-19.

The `azimuth` is the position of the light source on the plane of the scene. An `azimuth` angle of 0 degrees places the light source at the 3 o'clock position, one of 90 degrees moves it to 6 o'clock, and so on. Negative angles can also be used and are measured counterclockwise. For example, setting the `azimuth` variable to either -90 or 270 places the light source at 12 o'clock.

Table 20-19 Variables of the `DistantLight` Class

Variable	Type	Access	Default	Description
<code>azimuth</code>	Number	RW	45	The azimuth of the light source, in degrees
<code>elevation</code>	Number	RW	45	The elevation of the light source, in degrees

The `elevation` gives the angle of the light source above or below the plane of the scene. When the `elevation` is 0 or 180, the light source is on the plane of the scene, when it is 90, it is overhead the scene and shining directly down on it, and when it is 270 (or -90), it is directly below the scene.

The following code, which you will find in the file `javafxeffects/DistantLight1.fx`, allows you to move a `DistantLight` source around a large yellow circle to see the effect that is created:

```
Circle {
    centerX: 200
    centerY: 180
    radius: 150
    fill: Color.YELLOW
    effect: Lighting {
        light: DistantLight {
            azimuth: bind azimuthSlider.value
            elevation: bind elevationSlider.value
        }
        surfaceScale: 5
    }
}
```

Figure 20-34 shows two different configurations of the `DistantLight` source.

On the left of the figure, the `azimuth` and `elevation` variables both have the value 45, which places the light source at approximately the 4.30 position and elevated 45 degrees above its surface. You can see that this is the case because the lower-right edge of the circle is much brighter than the rest of it. On the right, the light source has been moved to the 9 o'clock position by setting the `azimuth` variable to 180 and moved very close to the plane of the scene as a result of the `elevation`, which is very nearly 0 degrees. Because of the low elevation, most of the circle is quite dark, with the exception of the edge at around the 9 o'clock position, which is closest to the light source.

It is worth examining here the effect of the `diffuseConstant` of the `Lighting` class. This constant acts as a multiplier to the RGB values of all the pixels on the lit surface. Therefore, you can use this variable to make the surface lighter or darker. The example in

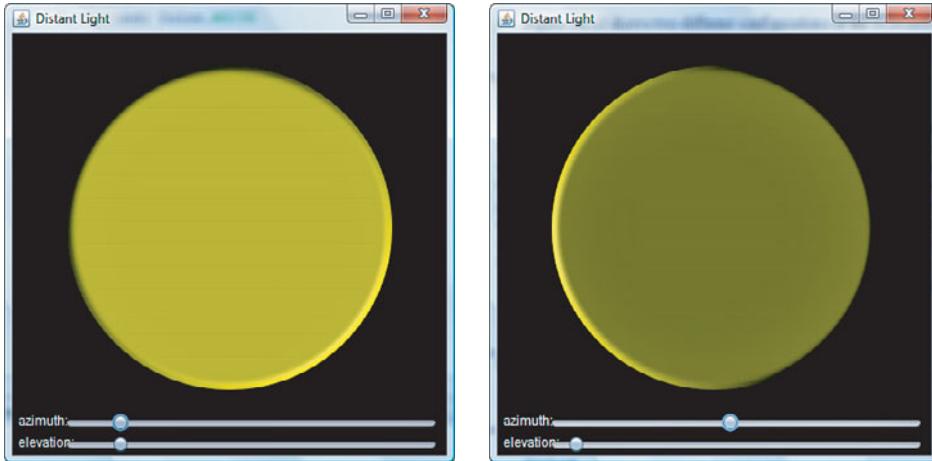


Figure 20-34 Using a `DistantLight` source

the file `javafxeffects/DistantLight2.fx` illustrates this by setting the `diffuseConstant` value of the `Lighting` effect to 1.5, which has the result of making the circle brighter, as you can see by comparing the result shown in Figure 20-35 with Figure 20-34, where this variable had the value 1.

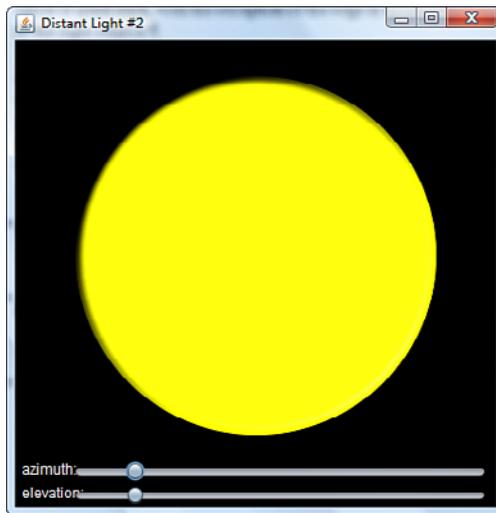


Figure 20-35 The effect of the `diffuseConstant` on a lit surface

PointLight

`PointLight` represents a single point of light that is positioned somewhere relative to the surface to be lit. The variables of the `PointLight` class, as shown in Table 20-20, allow you to specify exactly where the light source should be placed.

Table 20-20 Variables of the `PointLight` Class

Variable	Type	Access	Default	Description
<code>x</code>	Number	RW	0	The <code>x</code> coordinate of the light source
<code>y</code>	Number	RW	0	The <code>y</code> coordinate of the light source
<code>z</code>	Number	RW	0	The <code>z</code> coordinate of the light source

In the following code, a `PointLight` source whose position is bound to the values of three sliders is created and applied to a large yellow circle. If you run this example, which can be found in the file `javafxeffects/PointLight1.fx`, you can experiment with the effect of changing the location of the light source:

```
Circle {
    centerX: 200
    centerY: 180
    radius: 150
    fill: Color.YELLOW
    effect: Lighting {
        light: PointLight {
            x: bind xSlider.value
            y: bind ySlider.value
            z: bind zSlider.value
        }
        surfaceScale: 5
        specularConstant: bind (specCSlider.value as Number) / 10
        specularExponent: bind specESlider.value
    }
}
```

You can see two different `PointLight` configurations in Figure 20-36. On the left, the light is at (`x = 45, y = 45, z = 45`), which is to the top left of the circle itself. You can see that a `PointLight` source results in a more concentrated area of illumination than a `DistantLight`. On the right of the figure, the light source has been moved so that its reflection has moved more toward the center of the circle.

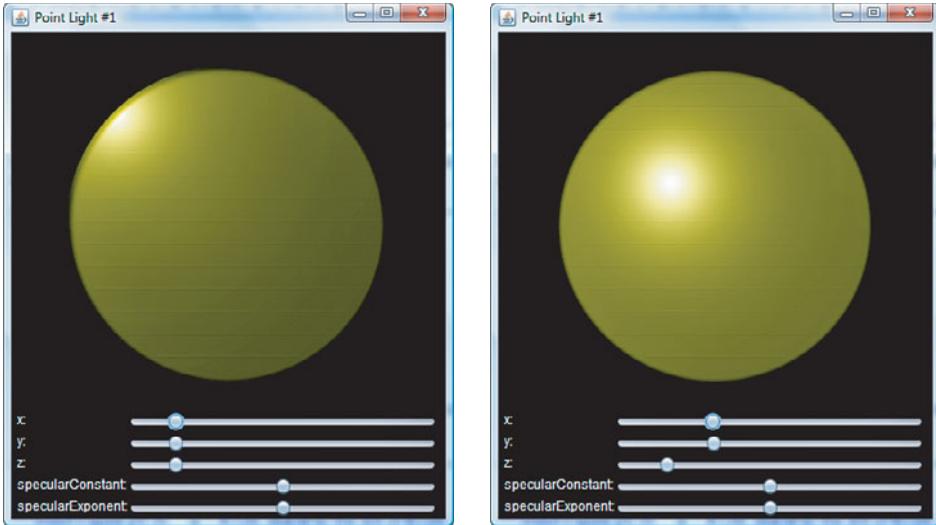


Figure 20-36 Using a `PointLight` source

The size and intensity of the reflection depends on values of the `specularConstant` and `specularExponent` variables of the `PointLight` class. Like `diffuseConstant`, `specularConstant` is a multiplier that is applied to the RGB values of the lit source, so values greater than 1 make the reflection bright, while values less than 1 make it dimmer. The `specularExponent` controls the spread of the light and therefore the radius of the reflected area. Increasing values of `specularExponent` reduce this radius and therefore make the reflection brighter. You can see examples that use different settings for these variables in Figure 20-37.

SpotLight

`PointLight` represents a single-point source of light that shines uniformly in every direction, much like the sun. `SpotLight` is a subclass of `PointLight` that acts like a point light source that radiates light over a more confined area. The rays of light are confined to the inside of a cone with its tip at the source. The axis of the cone points to a specified location on the surface of the object being lit. The combination of the position of the light source, the point at which it is aimed, and the width of the cone at the point at which the light reaches the lit object determines the lighting effect that you see. You can specify these values using the variables listed in Table 20-21.

The following code, which is from the file `javafxeffects/SpotLight1.fx`, applies a `SpotLight` effect to the same circle that we illuminated with a `PointLight` source in the previous example. The light source is placed 60 pixels above the center of the circle, while

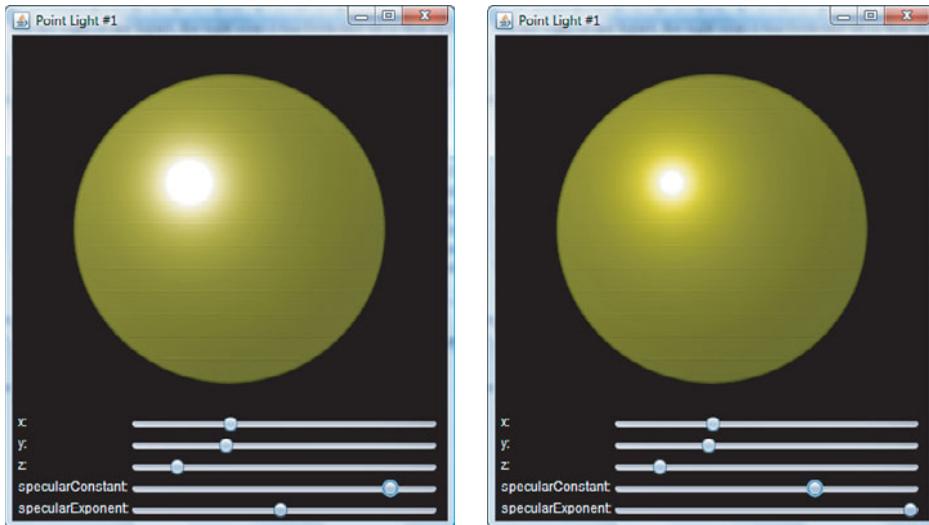


Figure 20-37 The effects of the `specularConstant` and `specularExponent` variables

Table 20-21 The Variables of the `SpotLight` Class

Variable	Type	Access	Default	Description
<code>pointsAtX</code>	Number	RW	0.0	The x coordinate of the point at which the light is aimed
<code>pointsAtY</code>	Number	RW	0.0	The y coordinate of the point at which the light is aimed
<code>pointsAtZ</code>	Number	RW	0.0	The z coordinate of the point at which the light is aimed
<code>specularExponent</code>	Number	RW	1.0	Controls the width of the light cone, in the range 0 to 4, inclusive

the point at which it is aimed can be controlled by three sliders. The effect produced by the `SpotLight` with these initial variable settings is shown on the left of Figure 20-38.

```
Circle {
  centerX: 200
  centerY: 180
  radius: 150
  fill: Color.YELLOW
  effect: Lighting {
    light: SpotLight {
      pointsAtX: bind xSlider.value
      pointsAtY: bind ySlider.value
      pointsAtZ: bind zSlider.value
    }
  }
}
```

```

        x: 200
        y: 180
        z: 60
        specularExponent: bind (specESlider.value as Number)/10
    }
    surfaceScale: 5
}
}

```

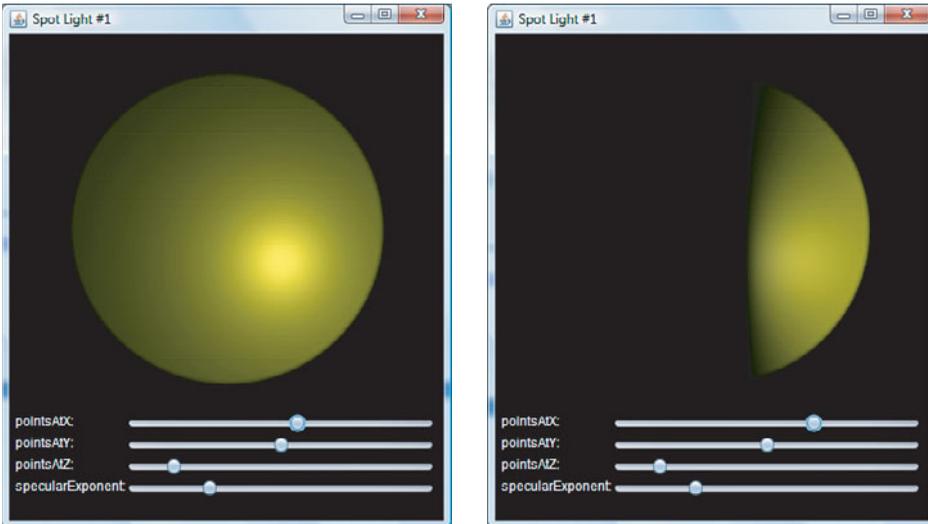


Figure 20-38 The **SpotLight** effect

Moving the aiming point changes the resultant lighting effect. When the light source is quite close to the target object, as it is in this case, even a small change in the aiming point can make a noticeable difference to the result. On the right of Figure 20-38, the aiming point has been moved only a small amount to the right and, as you can see, almost half of the circle is now in darkness.

The bottom slider in Figure 20-38 allows you to see the effect of changing the `specularExponent` value,⁸ which is initially set to its default value of 1. Increasing this value makes the light cone narrower that produces a more focused beam and therefore a smaller and brighter effect on the target, as you can see in Figure 20-39, where this variable has its maximum value of 4.0. With this setting, almost all the light is confined to a small area around the aiming point.

⁸ Do not confuse this variable with the `specularExponent` variable of the `Lighting` class.

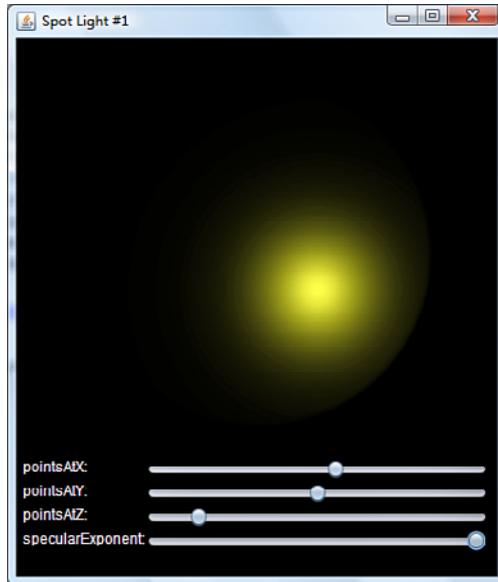


Figure 20-39 The effect of the `specularExponent` variable of the `SpotLight` class

Index

Symbols and Numerics

+= (add and assign) operator, 122
+ (addition) operator, 122
= (assignment) operator, 122
/= (divide and assign) operator, 122
/ (division) operator, 122
== (equality) operator, 8, 122, 133, 157
\ⁿ escape sequence, 103
\^l escape sequence, 103
\\ escape sequence, 103
\b escape sequence, 103
\f escape sequence, 103
> (greater than) operator, 122
>= (greater than or equal) to operator, 122
!= (inequality) operator, 122, 133-134
< (less than operator), 122
<= (less than or equal to operator), 122
*** (multiplication) operator, 122**
***= (multiply and assign) operator, 122**
\n escape sequence, 103
{ object literal } operator, 122
. (period) operator, 131
++ (postfix) operator, 122
— (postfix) operator, 122
++ (prefix) operator, 122
— (prefix) operator, 122
\r escape sequence, 103
-= (subtract and assign) operator, 122
- (subtraction) operator, 122
\t escape sequence, 103
- (unary minus) operator, 122
\012 escape sequence, 103
3D features, 372-374

A

Absolute resource names, 306-307

Abstract base class, 249-251

abstract keyword, 26

Access control, 306

Accessing

- arrays, 176
- databases, 1015, 1023
- external data sources, 949
- instance variables, 96-97
- wrapped Swing component, 832

action variable

- Button class, 749
- Hyperlink class, 752
- MediaTimer class, 638
- SwingButton component, 844
- SwingTextField class, 840
- TextInputControl class, 762
- Transition class, 614

Adobe Illustrator, 14, 703-704

Adobe Photoshop, 14, 704

after keyword, 26

Alerts

- confirm alert, 370-371
- defined, 342
- information alert, 369-370
- question alert, 371

Alignment of text, 468-469

altDown variable

KeyEvent class, 431

- MouseEvent class, 414

and keyword, 26

and operator, 122

angle variable (MotionBlur class), 659

Animation

- automatically reversing, 607-608
- changing speed and direction, 609-610
- defined, 591
- Duration class, 595-596
- javafxanimation package, 591
- length, 613
- media players, 633

pausing, 608

repeating, 605-607

restarting, 609

starting timeline from the end, 610-611

stopping, 608-609

support, 10-12

Timeline class, 12, 594

timelines, 591-594, 611-613

transitions, 591, 613-614

Animation paths, 717-720

Anonymous functions, 145-147

Ant build script, 33, 1049, 1052-1055

Ant task, 13

Ant version 1.7.0 or later, 1049

APIs (Application Programming Interfaces).

See also Platform API

- audio, 627
- common profile, 4-5
- desktop profile, 4-5
- documentation, 5
- Java Scripting API, 9
- JavaFX Desktop, 9
- JavaFX Mobile, 9
- JavaFX platform, 3-4
- JavaFX runtime, 9
- JavaFX Script language, 6-7
- networking APIs, 12-13
- profiles, 4-5
- Reflection, 309
- video, 627

Appearance of controls, 738

Applets

- converting into desktop applications, 70
- converting to installed applications, 15
- deploying, 14, 1033-1037
- Java plug-in, 15
- Java VM settings, 15
- loading, 15
- signing, 1043-1045
- writing, 4

AppletStageExtension class, 307

Application launcher, 13

Application Programming Interfaces (APIs).**See also Platform API**

audio, 627
 common profile, 4-5
 desktop profile, 4-5
 documentation, 5
 Java Scripting API, 9
 JavaFX Desktop, 9
 JavaFX Mobile, 9
 JavaFX platform, 3-4
 JavaFX runtime, 9
 JavaFX Script language, 6-7
 networking APIs, 12-13
 profiles, 4-5
 Reflection, 309
 video, 627

Applications

compiling, with command-line tools,
 1049-1051
 compiling, with NetBeans, 39
 converting applets to an installed
 application, 15
 deploying, applets, 14
 deploying, desktop applications, 14
 deploying, in multiple environments, 4
 deploying, mobile applications, 14
 deploying, TV applications, 14
 deployment options, 14
 executing, 1049-1052
 Graphical User Interface (GUI), 5
 packaging options, 14
 running, with command-line tools,
 1049, 1051-1052
 running, with Eclipse, 42
 running, with NetBeans, 39
 shutdown, 292-294
 signing, 1043-1045

Arc class, 442-444**Architecture of JavaFX platform, 3****ArcTo class, 453-455****Area charts, 930-931****args argument, 28****Arguments**

args, 28
 getting, 287-288
 named arguments, 288-290

Arithmetic operations, 123-125**Arrays**

accessing, 176
 declaring, 174-175
 initializing, 175-176
 iterating over an array, 193
 modifying, 176
 scope, 174
 similarity to sequences, 24
 size of, 177
 support for, 153

Artwork

exporting, 14
 importing, 14

as keyword, 26**as operator, 122, 131-132****assert keyword, 26****Assigning values, 23****associate() function, 297-298****at keyword, 26****Atom feeds, 949, 1004, 1008****Attaching triggers, 221, 229****attribute keyword, 26****Attributes**

classpathref, 1054
 compilerclasspath, 1054
 compilerclasspathref, 1054

Audio

Application Programming Interfaces
 (APIs), 627
 controlling media playback, 630-631
 FLV, 630
 FXM, 630
 media players, 627
 MP3 files, 629
 pausing, 632
 playing, 10, 627-628
 repeating playback, 632-633
 restricting playback, 632-633

- speeding up playback, 632
- stopping playback, 632
- support, 10
- volume control, 633

- autoKern variable (Font class), 478**
- Automating build process, 1049**
- autoPlay variable (MediaPlayer class), 631-632**
- autoReverse variable (Timeline class), 594**
- Axes (charts), 913**

B

- background variable (SwingTextField class), 839**
- backgroundLoading variable (Image class), 488**
- balance variable (MediaPlayer class), 634**
- Bar charts, 919-926**
- Batch files for automating build process, 1049**
- before keyword, 26**
- Behavior class, 894-895**
- Bidirectional binding, 24-25, 203-206**
- bind keyword, 8, 26**
- Binding**
 - bidirectional, 24-25, 203-206
 - conditional expressions, 202-203
 - creating, 194
 - def statement, 201
 - eager, 206-207
 - explained, 8, 194
 - expressions, 24-26, 199-200
 - functions, 25-26, 207-211, 217
 - instance variables, 201-202
 - javafxbinding package, 194
 - lazy, 206-207
 - object literals, 196-199
 - script variables, 194-196
 - sequences, 217
 - triggers, 223-224
 - unidirectional binding, 24-25, 206
- Blend class, 686-690**
- Blending**
 - defined, 686
 - groups, 651, 688-690
- Bloom class, 665-666**
- blurType variable**
 - DropShadow class, 660
 - InnerShadow class, 663
 - Shadow class, 661
- Boolean data type, 92, 102-103**
- Boolean operations, 129-130**
- Border container, 869-884**
- borderless variable (SwingTextField class), 839**
- bottlenecks, 46**
- bottomInput variable (Blend class), 687**
- bottomOpacity variable (Reflection class), 672**
- Bound functions, 134, 208-211, 217, 258-259**
- bound keyword, 26**
- Bounds class, 528-529**
- Bounds of nodes, 527-538**
- BoxBlur class, 657-659**
- break keyword, 26**
- break statement, 182-183**
- Breakpoints**
 - deleting, 76-77
 - disabling, 76-77
 - removing, 76-77
 - setting, 72-73
- brightness variable (ColorAdjust class), 670**
- Bubble charts, 934-935**
- bufferProgressTime variable (MediaPlayer class), 635**
- Build process, automating, 1049**
- Building SnowStorm application, 46**
- Built-in functions, 285**
- Bump map, 693-694**
- bumpInput variable (Lighting class), 691**
- Button control**
 - appearance, 739
 - Button class, 749
 - CSS properties, 1067-1068
 - examples, 749-751
 - features and functionality, 749
 - states, 751-752
- button variable (MouseEvent class), 410**
- Byte data type, 91**

C

Call Stack view, 73
Cameras, 372-374
Cascading Style Sheets (CSS). See CSS (Cascading Style Sheets)
Caspian theme, 1067
catch keyword, 26
causeOfFailure variable (Task class), 1009
Centering nodes, 577-579
centerX variable, 393
centerY variable, 393
Certificates, 1044-1045
Changing variable values, 74
char variable (KeyEvent class), 429
Character data type, 92, 101-102
characterEncoding variable (PullParser class), 976
Charts

- area charts, 930-931
- axes, 913
- bar charts, 919-926
- bubble charts, 934-935
- components, 912
- creating, 911
- customizing, 911, 937-946
- data, 914
- defined, 911
- hierarchy of classes, 912
- legend, 912
- line charts, 926-928
- pie charts, 914-919
- scatter charts, 932-934
- series, 913
- title, 912
- user interaction, 914, 936-937

CheckBox control

- appearance, 739
- CheckBox class, 759
- tri-state checkboxes, 759-761

ChoiceBox control, 739, 786-787
choke variable (InnerShadow class), 664
chooseBestCursor() method, 378
Circle class, 433, 438-439

Class files

- deleting, 1055
- JavaFX Script language, 8-9
- package/file.class, 1050
- source files, 1050

Class instances, creating, 328, 336-337**class keyword, 26****Class mapping, 297-298****Classes**

- abstract base class, 249-251
- AppletStageExtension, 307
- Arc, 442-444
- ArcTo, 453, 455
- Blend, 686-690
- Bloom, 665-666
- Behavior, 894-895
- Bounds, 528-529
- BoxBlur, 657-659
- Button, 749
- Circle, 433, 438-439
- ClipView, 574-577, 1064
- ColorAdjust, 669-671
- composite pattern, 267
- Container, 501
- Control, 735, 891
- CoordinateGrid, 866-869
- CubicCurve, 448-449
- CubicCurveTo, 453
- declaring, 240
- defining, 26
- DelegateShape, 434
- direct class references, 23
- DisplacementMap, 679-686
- DistantLight, 694-696
- documentation, 1058
- DropShadow, 651, 660
- Duration, 595-596
- Ellipses, 439-440
- example class, 241-242
- extending a base class, 251-253
- FadeTransition, 618-619
- FeedTask, 997

- Flood, 669
- Flow, 547-548
- Font, 477-479
- FXContext, 309
- GaussianBlur, 656-657
- Glow, 666-667
- Group, 482
- HBox, 560-562
- HttpRequest, 949-952
- HttpStatus, 959
- Identity, 667, 669
- Image, 377, 433, 487-488
- ImageCursor, 377
- ImageView, 433, 497-501, 1062
- importing, automatic imports, 22-23
- importing, by name, 20
- importing, static imports, 20-22
- importing, wildcard imports, 20
- initializing, 261-265
- InnerShadow, 664
- instance functions, 246, 248
- instance variables, 243-246
- InvertMask, 671
- Java classes, 97, 239-240
- JavaFX classes versus Java classes, 239-240
- JavaFX Script language, 6
- javafxclasses package, 240
- javafx.scene.Cursor, 375
- javafx.scene.paint.LinearGradient, 384
- javafx.scene.paint.RadialGradient, 393
- javafx.scene.text.Font, 7
- javafx.scene.text.Text, 7
- javafx.util.Math, 7
- java.lang.Math, 7
- java.lang.System, 6
- java.util.Random, 6-7
- KeyFrame, 596-600
- LayoutInfo, 542
- Lighting, 651, 690-694
- Line, 434-436
- ListView, 773-776
- Media, 628-630
- MediaPlayer, 628, 630
- MediaPlayerBehavior, 908-909
- MediaTimer, 638-639
- MediaView, 10, 628, 639-640
- mixins, 266-267
- MotionBlur, 659-660
- MouseEvent, 403-404
- nesting, 266
- Observable, 267
- Panel, 884-887
- ParallelTransition, 622-625
- PasswordBox, 771-773
- Path, 451
- PathElement, 452-453
- PathTransition, 619-622
- PauseTransition, 622
- PerspectiveTransform, 673-678
- PointLight, 697-698
- Polygon, 440-442
- Polyline, 440-442
- PullParser, 949, 975
- QuadCurve, 444-448
- QuadCurveTo, 453
- Rectangle, 433, 436-438, 1064
- Reflection, 671-673
- reserved words, 26
- Resource, 302-303
- RotateTransition, 616-617
- ScaleTransition, 617-618
- Scene, 358-360
- Screen, 585-590
- script files, 265-266
- ScrollBar, 788-789
- Scroller, 794
- ScrollView, 794-797
- SepiaTone, 673
- SequentialTransition, 622-625
- Shadow, 664-665
- Shape, 433
- ShapeIntersect, 449-451
- ShapeSubtract, 449-451
- Skin, 891-894

- Slider, 797-804
- Spotlight, 698-700
- Stack, 556-557
- Stage, 342, 358
- states, 239
- Storage, 301-302
- subclassing, 249
- SVGPath, 456
- SwingComponent, 830
- Text, 433, 466-472
- TextOrigin, 467-468
- Tile, 564
- Timeline, 12, 594, 605
- TimeSlider, 904-908
- Transition, 614
- TranslateTransition, 614, 616
- user interface classes, 9-10
- VBox, 563
- visibility, 242-243
- classpath path compiler option, 1051**
- classpathref attribute, 1054**
- clean target, 1055**
- clear() function, 303**
- clickCount variable (MouseEvent class), 410**
- Clipping nodes, 523-526**
- ClipView class**
 - fx-pannable property, 1064
 - variables, 574-577
- Code, example source, 1049**
- code variable (KeyEvent class), 429**
- color variable**
 - DropShadow class, 660
 - InnerShadow class, 663
 - Shadow class, 661
- ColorAdjust class, 669, 671**
- Colors**
 - CSS (Cascading Style Sheets), 822-828
 - cycle methods, 388-390
 - linear gradients, 383-392
 - nodes, 367, 378
 - radial gradients, 392-401
 - solid colors, 379-381
 - Stop objects, 385-388
 - Swing JColorChooser component, 382-383
- column variable (PullParser class), 976**
- columns variable (SwingTextField class), 839**
- Combining transforms, 517**
- Command-line compiler, 13**
- Command-line tools**
 - development, 33, 1049
 - javafxdoc, 13, 1055-1058
 - javafxpackager, 13-14, 46, 1025-1026
- Commands**
 - exec, 1055
 - java, 1051
 - javac, 1054
 - javafx, 1050-1052
 - javafx, 1050, 1054
 - javafxw, 1052
- Comments**
 - documentation comments, 18
 - extracting from documentation, 1055-1056
 - multiline comments, 18
 - single-line comments, 18
 - source files, 18
- Common profile (APIs), 4, 5**
- Comparing**
 - objects, 286-287
 - sequences, 167-168
- Comparing objects, 133-134**
- Comparison operators, 127-128**
- Compiler options**
 - classpath path, 1051
 - cp path, 1051
 - d classdirectory, 1051
 - profile name, 1051
 - sourcepath path, 1051
 - verbose, 1051
 - XDdumpjava, 1055
- compilerclasspath attribute, 1054**
- compilerclasspathref attribute, 1054**

Compiling applications

- with command-line tools, 1049-1051
- with Eclipse, 42
- with NetBeans, 39
- composable variable (MediaView class), 648**
- Composite pattern, 267**
- Compressing JAR files, 1042-1043**
- Concatenation of strings, 104-105**
- Conditional expressions, binding, 202-203**
- Conditional features, 307**
- Confirm alert, 370-371**
- connecting variable (GET operation), 953**
- Constraints for triggers, 224-225**
- Consumer JRE, 15**
- Container class, 501**
- Containers**
 - Border container, 869-884
 - CSS properties, 1064-1066
 - customizing, 869-887
 - fill color, 572-574
 - Flow container, 547-555, 1065
 - HBox container, 560-562, 1065
 - layout of notes, 540-542
 - Panel container, 572-574
 - Stack container, 555-559, 1065-1066
 - Tile container, 563-572, 1066
 - VBox container, 562-563, 1066
- containsFocus variable (Stage class), 342**
- content variable (Scene class), 359**
- contentInput variable (Lighting class), 691**
- continue keyword, 26**
- continue statement, 184**
- contrast variable (ColorAdjust class), 670**
- controlDown variable**
 - KeyEvent class, 431
 - MouseEvent class, 414
- Controlling media playback, 630-631**
- Controls. See also Swing controls**
 - appearance, 738
 - available controls, 738
 - Button control, 739, 749-752
 - CheckBox control, 739, 759-761

- ChoiceBox control, 739, 786-787
- Controls class, 735
- CSS properties, 1067-1070
- customizing, 864, 887-909
- features and functionality, 735
- Hyperlink control, 739, 752-755
- javafxcontrols package, 735
- Label control, 738-749
- ListView control, 739, 773-785, 1068
- PasswordBox control, 739, 771-773, 1070
- ProgressBar control, 739, 804-807
- ProgressIndicator control, 739, 804-807
- RadioButton control, 739, 756-759
- ScrollBar control, 739, 787-794, 1068-1069
- ScrollView control, 739, 794-797, 1069
- Separator control, 739, 807-808, 1069
- Slider controls, 739, 797-804, 1069-1070
- TextBox control, 739, 761-771, 1070
- ToggleButton control, 739, 756-759
- Tooltip control, 739, 808-809

Control class, 735, 891**Converting**

- applets, into desktop applications, 70
- applets, into installed applications, 15
- media files into FXM file format, 630
- numeric values, 99-101
- SVG-to-JavaFX Graphics Converter, 734

CoordinateGrid class, 866-869**Coordinates**

- nodes, 501
- screens, 585-590

Copying sequences, 158**-cp path compiler option, 1051****CPU profiling, 78-80****createScene() function, 1056****Creating**

- binding, 194
- charts, 911
- class instances, 328-337
- mappings, 297-298

- multiple copies of graphics, 731-733
 - objects, 95, 131
 - packages, in Eclipse, 41
 - packages, in NetBeans, 38
 - projects, in Eclipse, 40
 - projects, in NetBeans, 34
 - script files, 41
 - sequences, 153-155
 - SnowStorm application, 46-47
 - triggers, 221
- CROSSHAIR cursor, 376**
- Cross-platform controls. See Controls**
- CSS (Cascading Style Sheets)**
- colors, 822-828
 - effects, 828
 - features and functionality, 809
 - fonts, 824-826
 - hierarchical selectors, 818-820
 - inheritance, 820-822
 - installing a style sheet, 812
 - JavaFX support, 809, 1061
 - multiple selectors, 818
 - multiple style sheets, 824
 - paints, 826-828
 - selection by class, 813-814
 - selection by class and ID, 815-816
 - selection by ID, 814-815
 - state-dependent style, 816-818
 - structure, 813
 - Swing, 809
- CSS properties**
- fx-arc-height, 1064
 - fx-arc-width, 1064
 - fx-background, 1067
 - fx-blend-mode, 1062
 - fx-block-increment, 1069-1070
 - fx-click-to-position, 1069-1070
 - fx-columns, 1066, 1070
 - fx-cursor, 1061
 - fx-echo-char, 1070
 - fx-editable, 1070
 - fx-effect, 828, 1061
 - fx-fill, 826, 1063
 - fx-fit-to-height, 1069
 - fx-fit-to-width, 1069
 - fx-focus-traversable, 1061
 - fx-font, 826, 1062, 1067, 1070
 - fx-font-family, 824
 - fx-font-size, 824
 - fx-font-style, 825
 - fx-font-weight, 825-826
 - fx-graphic, 1067-1068
 - fx-graphic-hpos, 1067
 - fx-graphic-text-gap, 1067
 - fx-graphic-vpos, 1067
 - fx-hbar-policy, 1069
 - fx-hgap, 1065-1066
 - fx-hpos, 1065-1069
 - fx-image, 1062
 - fx-lines property, 1070
 - fx-major-tick-unit, 1070
 - fx-minor-tick-unit, 1070
 - fx-node-hpos, 1065-1066
 - fx-node-vpos, 1065-1066
 - fx-opacity, 1061
 - fx-padding, 1066
 - fx-pannable, 1064, 1068-1069
 - fx-rotate, 1061
 - fx-rows, 1066
 - fx-scale-x, 1061
 - fx-scale-y, 1061
 - fx-scale-z, 1061
 - fx-select-on-focus, 1070
 - fx-show-tick-labels, 1070
 - fx-show-tick-marks, 1070
 - fx-smooth, 1063
 - fx-snap-to-pixel, 1064
 - fx-snap-to-ticks, 1070
 - fx-spacing, 1065-1066
 - fx-strikethrough, 1062
 - fx-stroke, 1063
 - fx-stroke-dash-array, 1063
 - fx-stroke-dash-offset, 1063
 - fx-stroke-line-cap, 1063

- fx-stroke-line-join, 1063
- fx-stroke-miter-limit, 1063
- fx-stroke-width, 1063
- fx-text, 1068
- fx-text-alignment, 1062, 1068
- fx-text-fill, 1067
- fx-text-origin, 1062
- fx-text-overflow, 1068
- fx-text-wrap, 1068
- fx-tile-height, 1066
- fx-tile-width, 1066
- fx-translate-x, 1061
- fx-translate-y, 1062
- fx-translate-z, 1062
- fx-underline, 1062
- fx-unit-increment, 1069
- fx-vbar-policy, 1069
- fx-vertical, 1065-1070
- fx-vgap, 1065-1066
- fx-vpos, 1065-1069

CubicCurve class, 448-449

CubicCurveTo class, 453

currentCount variable (MediaPlayer class), 631

currentRate variable (Timeline class), 594

currentTime variable (MediaPlayer class), 635

cursor variable

- cursor shapes, 375
- Scene class, 358

Cursors

- CROSSHAIR, 376
- cursor variable, 375
- customizing, 377-378
- DEFAULT, 376
- E_RESIZE, 376
- HAND, 376
- H_RESIZE, 376
- javafx.scene.Cursor class, 375
- Microsoft Windows Platform, 376
- MOVE, 376
- NE_RESIZE, 376
- nodes, 375
- N_RESIZE, 376
- NW_RESIZE, 376

- SE_RESIZE, 376
- setting, 375-378
- shape of, 375
- S_RESIZE, 376
- SW_RESIZE, 376
- TEXT, 376
- V_RESIZE, 376
- WAIT, 376
- W_RESIZE, 376

Customizing

- charts, 911, 937-946
- containers, 869-887
- controls, 864, 887-909
- cursors, 377-378
- nodes, 482-485, 864

Cycle methods

- linear gradients, 388-390
- radial gradients, 398-399

cycleDuration variable (Timeline class), 594

cycleMethod variable

- linear gradients, 384
- radial gradients, 393

D

-d classdirectory compiler option, 1051

Data

- reading, 299, 301
- removing, 303-305
- writing, 299, 301

Data sources, accessing, 949

Data types

- Boolean, 92, 102-103
- Byte, 91
- Character, 92, 101-102
- defined, 89
- Double, 91
- Duration, 92-97
- Float, 91
- Integer, 91, 97-99
- Long, 91
- Number, 91, 99
- numeric, 97-98
- Short, 91

- String, 92, 103-108
- variables, 6
- Databases, accessing, 1015-1023**
- data-manipulation statements, 24**
- Debug toolbar, 75**
- Debugging**
 - breakpoints, disabling, 76-77
 - breakpoints, removing, 76-77
 - breakpoints, setting, 46-73
 - Call Stack view, 73
 - Local Variables view, 73-74
 - SnowStorm application, 46-76
 - stepping through code, 75-76
- Declaring**
 - arrays, 174-175
 - classes, 240
 - functions, 134-137
 - triggers, 221-222
 - variables, 6, 23, 89-95
- def keyword, 6, 26**
- def statement, 92, 201**
- DEFAULT cursor, 376**
- deferAction() function, 294**
- Deferring operations, 294-295**
- Defining**
 - classes, 26
 - functions, 25
- DelegateShape class, 434**
- delete keyword, 26**
- DELETE requests, 973-974**
- Deleting**
 - breakpoints, 76-77
 - class files, 1055
- Deletion operations with sequence variables, 230-232**
- Deploying applications**
 - applets, 14, 1033-1037
 - default restrictions, 1025
 - desktop applications, 14, 1029-1033
 - mobile applications, 14, 1047-1048
 - in multiple environments, 4
 - options, 14
 - support for deployment mechanisms, 1026
 - TV applications, 14
- Desktop applications**
 - converting applets into desktop applications, 70
 - deploying, 14, 1029-1033
 - packaging, 1026-1029
 - SnowStorm application, 70
- Desktop environments, 4**
- Desktop profile (APIs), 4-5**
- Development**
 - Ant build script, 33, 1049, 1052-1055
 - command-line tools, 1049
 - Eclipse, 3, 33, 39-42
 - Integrated Development Environment (IDE), 33
 - NetBeans, 3, 33, 34-39
 - options, 33
- diffuseInput variable (Lighting class), 691**
- __DIR__ variable, 31**
- Direct class references, 23**
- disable variable (Node class), 426**
- disabled state, 816**
- disabled variable (Node class), 426**
- Disabling breakpoints, 76-77**
- disassociate() function, 298**
- DisplacementMap class, 679, 686**
- Displaying**
 - images, 486, 497, 501
 - text, 467
- DistantLight class, 694, 696**
- Division**
 - of a nonzero value by zero, 127
 - of zero by zero, 127
- Documentation**
 - APIs (Application Programming Interfaces), 5
 - classes, 1058
 - comments, 1055-1056
 - Eclipse, 44
 - extracting, 13, 1049, 1055-1058
 - generating, 1055-1058

- NetBeans, 43–44
- source code, 43
- source files, 1057
- viewing, 1055–1056

Documentation comments, 18**documentType variable (PullParser class), 975****done variable (GET operation), 955****done variable (Task class), 1009****doneConnect variable (GET operation), 953****doneHeaders variable (GET operation), 954****doneRead variable (GET operation), 955****doRequest() function, 956****Double data type, 91****Downloading**

- example source code, 33, 1049
- JavaFX, 4
- JavaFX Production Suite, 704
- Production Suite, 14

dragAnchorX variable (MouseEvent class), 416**dragAnchorY variable (MouseEvent class), 416****dragX variable (MouseEvent class), 416****dragY variable (MouseEvent class), 416****Drawing**

- arcs, 443–444
- ellipses, 439–440
- lines, 435
- polygons, 440–441
- rectangles, 437

DropShadow class, 651, 660**Duplicating graphics, 731–733****Duration class, 595–596****Duration data type, 92–97****duration variable (Media class), 628**

E

Eager binding, 206–207**Eclipse**

- development, 3, 33, 39–42
- documentation, 44
- Installing, 33
- Java Web Start, 67

- mobile emulator, 71

- support, 3, 13

- TV emulator, 72

- viewing documentation, 1055

editable variable

- SwingComboBox component, 854

- SwingTextField class, 839

Effects

- blend, 686, 690

- bloom, 665–666

- box blur, 657–659

- bump map, 693–694

- chains, 651

- color adjust, 669–671

- CSS (Cascading Style Sheets), 828

- defined, 651

- displacement map, 679–686

- drop shadow, 651, 660

- experimenting with, 656

- flood, 669

- Gaussian blur, 656–657

- glow, 666–667

- groups, 655

- identity, 667–669

- inner shadow, 663

- invert mask, 671

- javafx.effects package, 651

- javafx.scene.effects package, 651

- javafx.scene.effects.lighting package, 651

- lighting, 651, 690–700

- motion blur, 659–660

- nodes, 368–369, 654

- perspective transform, 673–678

- reflection, 671–673

- sepia tone, 673

- shadow, 664–665

- transformations, 654

- video, 646

- warp, 684–686

Effects Playground application, 656**Ellipses class, 439–440****else keyword, 26**

Embedded expressions, 105-106

Embedding

- fonts, 720-724
- images, 724-725

embolden variable (Font class), 478

encoding variable (PullParser class), 976

endX variable, 384

endY variable, 384

Equality of sequences, 157

equals() function, 8

E_RESIZE cursor, 376

error variable (GET operation), 954

error variable (Image class), 488

Escape sequences, 103-104

Escape sequences for string literals, 103

Evaluation order of operators, 121-123

event variable (PullParser class), 976

Events

- keyboard events, 425-432
- mouse events, 402
- nodes, 367, 401-402

Example source code

- downloading, 33, 1049
- installing, 33, 1049

example.class property, 1055

Exception handling, 193-194

exception variable (GET operation), 953

exclusive keyword, 26

exec command, 1055

Executing

- applications, 1049-1052
- triggers, 221-222

exit() function, 293

Experimenting with effects, 656

Exporting graphics, 14, 705-708

Expressions

- binding, 24-26, 199-200
- defined, 121
- embedded, 105-106
- formatting, 106-108
- for statement, 186-187

extends keyword, 26

Extensible Markup Language (XML)

- HttpRequest class, 949
- PullParser class, 949

extensions variable (Stage class), 357

External data sources, accessing, 949

Extracting documentation, 13, 1049, 1055-1058

F

fader variable (MediaPlayer class), 634

FadeTransition class, 618-619

failed variable (Task class), 1009

false keyword, 26

family variable (Font class), 478

FeedTask class, 997

__FILE__ variable, 31

Files

- class files, 1050
- package/file.class, 1050
- source files, 15-18, 20, 1050

fill variable (Scene class), 358

Fills

- shapes, 434, 463-466
- text, 470-471

Filtered searches, 326-327

finally keyword, 26

findClass() method, 310

first keyword, 26

fitHeight variable

- ImageView class, 498
- MediaView class, 640

fitWidth variable

- ImageView class, 498
- MediaView class, 640

Float data type, 91

Flood class, 669

Flow class, 548

Flow container

- CSS properties, 1065
- Flow class, 547-548
- fx-hgap property, 1065
- fx-hpos property, 1065

- fx-node-hpos property, 1065
- fx-node-vpos property, 1065
- fx-snap-to-pixel property, 1064
- fx-vertical property, 1065
- fx-vgap property, 1065
- fx-vpos property, 1065
- layout, 547

Flow-of-control statements

- break statement, 182-183
- continue statement, 184
- defined, 26
- if statement, 179-181
- if/else statement, 26
- Java versus JavaFX, 179
- return statement, 26
- for statement, 26, 184, 193
- throw statement, 26
- try/catch/finally statements, 26, 194
- while statement, 26, 181-182

FLV media files, 630**focused state, 816****focused variable (Node class), 426****focusTraversable variable (Node class), 426****focusX variable, 393****focusY variable, 393****Font class, 477-479****font variable**

- SwingComponent class, 830
- Text class, 472

Fonts

- characteristics, 473
- CSS (Cascading Style Sheets), 824-826
- embedding, 720-724
- font family name, 473-474
- font name, 473-474
- font variable, 472
- font weight, 474
- listing fonts and font families, 479
- logical fonts, 477
- object literals, 480-481
- physical fonts, 476-477
- position, 475-476
- posture, 474

searching, 481-482

selecting, 479-482

size, 476

for keyword, 26**for statement, 26, 184-193****foreground variable (SwingComponent class), 830****Formatting expressions, 106-108****fraction variable (Reflection class), 672****framerate variable (Timeline class), 594****from keyword, 26****fullScreen variable (Stage class), 348****function keyword, 26****Function overloading, 253-254****Function overriding, 254-257****function() operator, 122****Functions**

- anonymous functions, 145-147
- associate(), 297-298
- binding, 25-26, 207-211, 217
- bound functions, 134, 208-211, 217, 258-259
- built-in, 285
- clear(), 303
- createScene(), 1056
- declaring, 134-137
- deferAction(), 294
- defined, 121, 134
- defining, 25
- disassociate(), 298
- doRequest(), 956
- equals(), 8
- exit(), 293
- within functions, 138
- getArguments(), 287-288
- getFontNames(), 479
- getMembers(), 327-328
- getProperty(), 290
- getting, 324-326
- invoking, 95-96, 131, 138
- isReadOnly(), 206
- isSameObject(), 286-287
- logic, 134

- object literals, 150-151
- onConnecting, 953
- onDoneConnect, 953
- onException, 953
- onReadingHeaders, 954
- onResponseCode, 954
- onStarted, 953
- passing a function to another function, 142-144
- print(), 286
- println(), 9, 285-286
- random(), 7
- reserved words, 26
- run(), 28-29, 287
- script functions, 137-138
- selecting by classname, 258
- start(), 951, 955
- stop(), 951
- substring(), 108
- toString(), 155
- trim(), 8
- unbound functions, 207-208
- updateViewport(), 501
- uses, 25
- variables, 137-142
- visibility, 147, 255-256
- .fx naming suffix, 15**
- fx-arc-height property, 1064**
- fx-arc-width property, 1064**
- fx-background property, 1067**
- fx-blend-mode property, 1062**
- fx-block-increment property, 1069-1070**
- FXClassType class, 315-316, 324**
- fx-click-to-position property, 1069-1070**
- fx-columns property, 1066, 1070**
- FXContext class, 309**
- fx-cursor property, 1061**
- fx-echo-char property, 1070**
- fx-effect property, 828, 1061**
- fx-fill property, 826, 1063**
- fx-fit-to-height property, 1069**
- fx-fit-to-width property, 1069**
- fx-focus-traversable property, 1061**
- fx-font property, 826, 1062, 1067**
- fx-font-family property, 824**
- fx-font-size property, 824**
- fx-font-style property, 825**
- fx-font-weight property, 825-826**
- FXFunctionType class, 316-317**
- FXFunctionValue class, 320**
- fx-graphic property, 1067-1068**
- fx-graphic-hpos property, 1067**
- fx-graphic-text-gap property, 1067**
- fx-graphic-vpos property, 1067**
- fx-hbar-policy property, 1069**
- fx-hgap property**
 - Flow container, 1065
 - Tile container, 1066
- fx-hpos property**
 - Flow container, 1065
 - HBox container, 1065
 - Labeled nodes, 1067
 - Separator control, 1069
 - Tile container, 1066
 - VBox container, 1066
- fx-image property, 1062**
- FXJavaArrayType class, 316**
- fx-lines property, 1070**
- FXM media files, 630**
- fx-major-tick-unit property, 1070**
- FXMember class, 320, 322**
- FXMemberFilter class, 326**
- fx-minor-tick-unit property, 1070**
- fx-node-hpos property**
 - Flow container, 1065
 - Stack container, 1065
 - Tile container, 1066
 - VBox container, 1066
- fx-node-vpos property**
 - Flow container, 1065
 - HBox container, 1065
 - Stack container, 1065
 - Tile container, 1066
- FXObjectValue class, 319**
- fx-opacity property, 1061**

- fx-padding property, 1066
- fx-pannable property
 - ClipView class, 1064
 - ScrollView control, 1069
- FXPrimitiveType class, 315
- FXPrimitiveValue class, 318-319
- fx-rotate property, 1061
- fx-rows property, 1066
- fx-scale-x property, 1061
- fx-scale-y property, 1061
- fx-scale-z property, 1061
- FXSequenceType, 316
- FXSequenceValue class, 320
- fx-show-tick-labels property, 1070
- fx-show-tick-marks property, 1070
- fx-smooth property, 1063
- fx-snap-to-pixel property, 1064
- fx-snap-to-ticks property, 1070
- fx-spacing property, 1065-1066
- fx-strikethrough property, 1062
- fx-stroke property, 1063
- fx-stroke-dash-array property, 1063
- fx-stroke-dash-offset property, 1063
- fx-stroke-line-cap property, 1063
- fx-stroke-line-join property, 1063
- fx-stroke-miter-limit property, 1063
- fx-stroke-width property, 1063
- fx-text property, 1068
- fx-text-alignment property, 1062, 1068
- fx-text-fill property, 1067
- fx-text-origin property, 1062
- fx-text-overflow property, 1068
- fx-text-wrap property, 1068
- fx-tile-height property, 1066
- fx-tile-width property, 1066
- fx-translate-x property, 1061
- fx-translate-y property, 1062
- fx-translate-z property, 1062
- FXType class, 314-315
- fx-underline property, 1062
- fx-unit-increment property, 1069
- FXValue class, 318
- FXVarMember class, 322

- fx-vbar-policy property, 1069
- fx-vertical property
 - Flow container, 1065
 - ListView control, 1068
 - ScrollBar control, 1069
 - Slider control, 1070
 - Tile container, 1066
- fx-vgap property, 1065-1066
- fx-vpos property
 - Flow container, 1065
 - HBox container, 1065
 - Labeled nodes, 1068
 - Separator control, 1069
 - Tile container, 1066
 - VBox container, 1066
- FXZ file, 705

G

- garbage collection, 81-82
- GaussianBlur class, 656-657
- Generating documentation, 1055-1058
- GET requests
 - callback functions, 953
 - connecting variable, 953
 - doneConnect variable, 953
 - exception variable, 953
 - lifecycle, 952-955
 - onConnecting function, 953
 - onDoneConnect function, 953
 - onException function, 953
 - onReadingHeaders function, 954
 - onResponseCode function, 954
 - onStarted function, 953
 - readingHeaders variable, 954
 - responseCode variable, 954
 - started variable, 953
 - variables, 953
- getArguments() function, 287-288
- getBestSize() method, 377
- getFontNames() function, 479
- getMembers() function, 327-328
- getName() method, 310
- getProperty() function, 290

getSuperClasses() method, 310-313

Getting

- arguments, 287-288
- class information, 313
- functions, 324-326
- previous value of variables, 223
- system properties, 290

getValue() method, 316

GlassFish application server, 1015

Glow class, 666-667

Graphical User Interface (GUI) applications and JavaFX Script language, 5

Graphics

- animation paths, 717-720
- creating multiple copies, 731-733
- duplicating, 731-733
- embedding fonts, 720-723
- embedding images, 724-725
- exporting, 14, 705-708
- FXZ file, 705
- importing, 14, 703, 733-735
- loading, 710-717
- previewing, 708-710
- Scalable Vector Graphics (SVG), 704-705, 733-735
- UI Stub class, 725-731

Graphics effects. See Effects

greater than (>) operator, 122

greater than or equal (>=) to operator, 122

Group class, 482

Grouping import statements, 20

Groups

- blending, 651, 688-690
- effects, 655
- nesting, 364
- nodes, 9, 482-485
- shapes, 482

GUI (Graphical User Interface) applications and JavaFX Script language, 5

H

HAND cursor, 376

HBox class, 560-562

HBox container

- CSS properties, 1065
- fx-hpos property, 1065
- fx-node-vpos property, 1065
- fx-snap-to-pixel property, 1064
- fx-spacing property, 1065
- fx-vpos property, 1065
- HBox class, 560-562

headers variable (FeedTask class), 996

height variable

- BoxBlur class, 657
- DropShadow class, 660
- Flood class, 669
- Image class, 488
- InnerShadow class, 663
- Media class, 628
- Scene class, 358
- Shadow class, 661
- Stage class, 348

Hello, World example

- Java, 5-6
- JavaFX Script language, 5-6

high identifier, 230

horizontalAlignment variable

- SwingButton component, 844
- SwingLabel class, 833, 836-837
- SwingTextField class, 840

horizontalTextPosition variable

- SwingButton component, 844
- SwingLabel class, 833, 838

hotspotX variable (CursorImage object), 377

hotspotY variable (CursorImage object), 377

hover state, 816

hover variable (Node class), 424-425

H_RESIZE cursor, 376

HTTP request

- DELETE requests, 973-974
- doRequest() function, 956
- GET requests, 952-964
- InputStream, 952
- lifecycle, 952-955
- OutputStream, 952

POST requests, 971-973

PUT requests, 965-971

HttpRequest class

doRequest() function, 956

example use, 950-951

functionality, 949-950

HTTP operations, 952

HTTP protocol, 950

main thread, 955

retrieving data from an FTP server,
964-965

start() function, 951, 955

stop() function, 951

streams, 951

HttpStatus class, 959

hue variable (**ColorAdjust class**), 670

Hyperlink control, 739, 752-755

I

icon variable

SwingButton component, 844

SwingLabel class, 833, 834-835

iconified variable (**Stage class**), 348

icons variable (**Stage class**), 342-344

ID3 format metadata, 629

IDE (**Integrated Development Environment**),
3, 33

identifiers

high, 230

low, 230

newVals, 230

oldVals, 229

Identity class, 667-669

if keyword, 26

if statement, 179-181

if/else statement, 26

ignoreWhiteSpace variable (**PullParser class**),
976

IllegalArgumentException, 225

Illustrator (**Adobe**), 14, 703-704

Image class, 377, 433, 487-488

image variable (**ImageView class**), 498

ImageCursor class, 377

Images

displaying, 486, 497, 501

embedding, 724-725

fetching image data, 490-493

loading, 486-487

resizing images, 493-497

specifying location of, 488-489

ImageView class, 433, 497-501, 1062

Immutable sequences, 154

import keyword, 26

import statements

grouping, 20

source files, 15, 20

Importing

classes, automatic imports, 22-23

classes, by name, 20

classes, static imports, 20-22

classes, wildcard imports, 20

graphics, 14, 703, 733-735

in keyword, 26

indexOf keyword, 26

indexOf operator, 122

Infinity, 128-129

Information alert, 342-370

Inheritance

CSS (**Cascading Style Sheets**), 820-822

mixins, 273-280

init block, 263-265

init keyword, 26

Initializing

arrays, 175-176

classes, 261-265

mixins, 281-283

objects, 148-149

Inkscape, 14, 705, 734

InnerShadow class, 663

input variable

Bloom class, 665

BoxBlur class, 657

ColorAdjust class, 670

DisplacementMap class, 679

GaussianBlur class, 656

- Glow class, 666
 - InvertMask class, 671
 - MotionBlur class, 659
 - PerspectiveTransform class, 674
 - PullParser class, 976
 - Reflection class, 672
 - SepiaTone class, 673
 - Shadow class, 661
 - input variable (GET operation), 955**
 - InputStream, 952**
 - insert keyword, 26**
 - Insertion operations with sequence variables, 230, 233-234**
 - Inserting elements into sequences, 163-165**
 - Inspecting variables, 73**
 - Installing**
 - Ant version 1.7.0 or later, 1049
 - Eclipse, 33
 - example source code, 33, 1049
 - Java, 14
 - NetBeans, 33
 - Software Development Kit (SDK), 1049
 - style sheets, 812
 - Instance functions, 246, 248**
 - Instance methods, 24**
 - Instance variables**
 - accessing, 96-97
 - binding, 201-202
 - classes, 243-246
 - object creation, 95
 - triggers, 236-237
 - instanceof keyword, 26**
 - instanceof operator, 122, 132-133**
 - Integer data type, 91, 97-99**
 - Integer overflow, 125-126**
 - Integrated Development Environment (IDE), 3, 33**
 - Internationalization, 109-117, 295, 298**
 - Internet access, 12-13**
 - interpolate variable (Timeline class), 594**
 - Interpolation, 600-603**
 - interpolator variable (Transition class), 614**
 - interval variable (FeedTask class), 996**
 - into keyword, 26**
 - invalidate keyword, 26**
 - inverse keyword, 26**
 - InvertMask class, 671**
 - invokeLater() method, 294**
 - Invoking**
 - functions, 95-96, 131, 138
 - methods, 24, 138-139
 - isAssignableFrom() method, 313-314**
 - isJfxType() method, 313**
 - isMixin() method, 313**
 - isReadOnly() function, 206**
 - isSameObject() function, 286-287**
 - isSupported() method, 307**
 - items variable (SwingComboBox component), 854**
 - Iterating**
 - over a list, 190-191
 - over a map, 191-192
 - over a sequence, 184-186
 - over an array, 193
 - over an iterable, 190
 - over multiple sequences, 188-190
 - iterations variable (BoxBlur class), 657**
-
- ## J
- JAR files, compressing, 1042-1043**
 - Java**
 - Hello, World example, 5-6
 - installing, 14
 - platform dependencies, 14-15
 - Java classes, 97, 239-240**
 - java command, 1051**
 - Java ME, 4**
 - Java methods, invoking, 138-139**
 - Java plug-in, 15**
 - Java Runtime Environment (JRE), 14-15**
 - Java Scripting API, 9**
 - Java SE, 4**
 - Java TV, 4**
 - Java VM settings for applets, 15**

Java Web Start

- Eclipse, 67
- features, 46–66
- NetBeans, 66–67

javac command, 1054**JavaFX**

- CSS support, 809, 1061
- downloading, 4
- overview, 3

javafx command, 1050-1052**JavaFX Desktop and APIs, 9****javafx launcher, 1055****JavaFX Mobile and APIs, 9****JavaFX platform**

- APIs (Application Programming Interfaces), 3–4
- architecture, 3
- security, 3

JavaFX Production Suite, 703-705**JavaFX runtime**

- APIs (Application Programming Interfaces), 9
- Desktop environments, 4
- Java TV, 4
- mobile devices, 4
- profile-dependence, 9
- profile-independence, 9
- versions, 4

JavaFX Script language

- APIs (Application Programming Interfaces), 6–7
- class files, 8
- classes, 6
- defined, 3
- Graphical User Interface (GUI) applications, 5
- Hello, World example, 5–6
- Java Scripting API, 9
- overview, 5
- scripts, 8–9
- syntax, 3
- Type inference, 6

JavaFX TV emulator, 72, 371**javafxanimation package, 591****javafx.applet.codebase property, 291****javafx.application.codebase property, 291****javafx.async package, 949****javafxbinding package, 194****javafxcommand, 1050, 1054****javafxclasses package, 240****javafxcontrols package, 735****javafxdata package, 949****javafx.data.pull package, 950****javafxdoc command-line tool**

- classes, 1058
- command-line arguments, 1056
- d option, 1057
- illustration of, 1056–1057
- output files, 1057–1058
- source files, 1057
- sourcepath option, 1057
- summary information, 1058
- uses, 13, 1055–1056

javafxeffects package, 651**javafx.encoding property, 291****javafx.file.separator property, 291****javafximport package, 703****javafx.io.http package, 950****javafx.java.ext.dirs property, 291****javafx.java.io.tmpdir property, 291****javafx.java.vendor property, 291****javafx.java.vendor.url property, 291****javafx.java.version property, 291****javafx.language property, 291****javafx.line.separator property, 291****javafxmedia package, 628****javafxnodes package, 375****javafx.os.arch property, 291****javafx.os.name property, 291****javafx.os.version property, 291****javafxpackager command-line tool, 13-14, 46, 1025-1026****javafx.path.separator property, 291****javafx.reflect package, 309****javafxreflection package, 309****javafx.region property, 291**

javafx.runtime.version property, 291
 javafx.scene.Cursor class, 375
 javafx.scene.effects package, 651
 javafx.scene.effects.lighting package, 651
 javafx.scene.paint.LinearGradient class, 384
 javafx.scene.paint.RadialGradient class, 393
 javafx.scene.text.Font class, 7
 javafx.scene.text.Text class, 7
 javafx.scene.transform.Transform class, 505
 javafx/SequenceTriggers.fx file, 230
 javafxshapes package, 433
 javafxstyle package, 809
 javafxswing package, 829
 javafx.timezone property, 291
 javafxtriggers package, 221
 javafx.user.dir property, 291
 javafx.user.home property, 291
 javafx.util.Math class, 7
 javafx.version property, 291
 javafxw command, 1052
 java.lang.Math class, 7
 java.lang.reflect package, 309
 java.lang.System class, 6
 java.scene.input.MouseEvent argument, 402-403
 JavaScript Object Notation (JSON), 949
 java.util.Random class, 6-7
 Joins, 459-460
 JRE (Java Runtime Environment), 14-15
 JSON (JavaScript Object Notation), 949
 JSON weather service, 992-995
 JSON web service client, 988, 992
 jvisualvm profiler, 46

K

Keyboard events

- defined, 425-426
- input focus, 426-429
- modifier keys, 431
- noncharacter keys, 432
- receiving key events, 429-430
- repeated keys, 432
- types of key events, 430-431

KeyEvent class

- altDown variable, 431
- char variable, 429
- code variable, 429
- controlDown variable, 431
- metaDown variable, 431
- shiftDown variable, 431
- text variable, 429

KeyFrame class, 596-600

keyFrames variable (Timeline class), 594

Keywords

- and, 26
- as, 26
- at, 26
- before, 26
- for, 26
- from, 26
- in, 26
- into, 26
- on, 26
- with, 26
- abstract, 26
- after, 26
- assert, 26
- attribute, 26
- bind, 8, 26
- bound, 26
- break, 26
- catch, 26
- class, 26
- context, 26
- continue, 26
- def, 6, 26
- delete, 26
- else, 26
- exclusive, 26
- extends, 26
- false, 26
- finally, 26
- first, 26
- function, 26
- if, 26
- import, 26

indexof, 26
 init, 26
 insert, 26
 instanceof, 26
 invalidate, 26
 inverse, 26
 last, 26
 lazy, 26
 mixin, 26
 mod, 26
 nativearray, 26
 new, 7, 26
 not, 26
 null, 26
 or, 26
 override, 26
 package, 26
 postinit, 26
 private, 26
 protected, 26
 public, 26
 public-init, 26
 public-read, 26
 replace, 26
 reserved words, 26
 return, 26
 reverse, 26
 sizeof, 26
 static, 26
 step, 26
 super, 26
 then, 26
 this, 26
 throw, 26
 trigger, 26
 true, 26
 try, 26
 tween, 26
 typeof, 26
 types of, 26
 var, 6, 26
 where, 26
 while, 26

L

Label control

appearance, 739
 basic labels, 741-742
 CSS properties, 1067-1068
 features and functionality, 738-739
 icons, 834-835
 Label class, 741
 Labeled class, 739-741
 multiline text, 746-747
 positioning content, 836-838
 positioning text and graphics, 742-746
 text, 834-835
 text overruns and wrapping, 747-749
 variables, 833-834

labelFor variable (SwingLabel class), 833

last keyword, 26

LayoutInfo class, 542

Layouts of nodes, 501, 538-579

layoutX variable, 504

layoutY variable, 504

Lazy binding, 206-207

lazy keyword, 26, 206

length variable (Resource class), 302

less than operator (<), 122

less than or equal to operator (<=), 122

letterSpacing variable (Font class), 478

level variable

 Glow class, 666

 SepiaTone class, 673

Libraries, 1039-1042

Lifecycle of HTTP request, 952-955

ligatures variable (Font class), 478

light variable (Lighting class), 691

Lighting class, 651, 690-694

Line charts, 926-928

Line class, 434-436

line variable (PullParser class), 976

Linear gradients

 cycle methods, 388-390

 defined, 383

- direction of, 386
- example, 384-385
- `javafx.scene.paint.LinearGradient` class, 384
- multicolor linear gradients, 387
- nonproportional gradients, 390-391
- resizing areas filled linear gradients, 386
- Stop object, 385-388
- stops, 384
- strokes, 391-392
- support for, 383
- variables, 384
- Listing fonts and font families, 479**
- ListView control**
 - appearance, 739
 - cell rendering, 782-785
 - CSS properties, 1068
 - dynamic list content, 776
 - features and functionality, 773
 - `-fx-pannable` property, 1068
 - `-fx-vertical` property, 1068
 - ListView class, 773-776
 - selection of list items, 778-782
- ltx variable (PerspectiveTransform class), 674**
- lly variable (PerspectiveTransform class), 674**
- Loading**
 - applets, 15
 - graphics, 710-717
 - images, 486-487
- Local storage, 298-299**
- Local Variables view, 73-74**
- Localization, 108-116**
- location variable (FeedTask class), 996**
- Logging, 267-271**
- Logic of functions, 134**
- Logical fonts, 477**
- Long data type, 98**
- Loops and bound functions, 216-217**
- low identifier, 230**
- lrx variable (PerspectiveTransform class), 674**
- lry variable (PerspectiveTransform class), 674**

M

- main() method, 28**
- mapData variable (DisplacementMap class), 679**
- Mapping between source files and class files, 8-9, 1050**
- Mappings**
 - class mapping, 297-298
 - creating, 297-298
 - package mapping, 297
 - removing, 298
- maximum variable (SwingSlider component), 857**
- maxLength variable (Resource class), 302**
- Media class, 628-630**
- Media players**
 - animation, 633
 - control bar, 895-909
 - functionality, 627
 - monitoring playback, 634-635
 - synchronizing external events with playback, 637-638
 - triggers, 635-636
 - Volume control, 633
- media variable (MediaPlayer class), 631**
- MediaPlayer class, 628, 630**
- mediaPlayer variable (MediaView class), 639**
- MediaPlayerBehavior class, 908-909**
- MediaTimer class, 638-639**
- MediaView class, 10, 628, 639-640**
- Memory leaks, 46, 80-82**
- Memory profiling, 80-82**
- metadata variable (Media class), 628**
- metaDown variable**
 - KeyEvent class, 431
 - MouseEvent class, 414
- Methods**
 - `chooseBestCursor()`, 378
 - cycle methods, 388-390, 398-399
 - `findClass()`, 310
 - `getBestSize()`, 377
 - `getName()`, 310
 - `getSuperClasses()`, 310-313

- getValue(), 316
- instance, 24
- invokeLater(), 294
- invoking, 24
- isAssignableFrom(), 313–314
- isJfxType(), 313
- isMixin(), 313
- isSupported(), 307
- main(), 28
- references, 24
- static, 24
- Microsoft Windows Platform and cursors, 376**
- middleButtonDown variable (MouseEvent class), 410**
- MIDIlets, 4**
- MIDP (Mobile Information Device Profile)-based cell phones, 7**
- minimum variable (SwingSlider component), 857**
- mixin keyword, 26**
- Mixins**
 - characteristics, 272–273
 - defined, 266–267
 - inheritance, 273–280
 - initializing, 281–283
 - logging, 267–271
 - triggers, 280–281
 - uses for, 267
- Mobile applications**
 - deploying, 14, 1047–1048
 - packaging, 1045–1047
 - system properties, 292
- Mobile devices**
 - JavaFX runtime, 4
 - Mobile Information Device Profile (MIDP)-based cell phones, 7
 - Stage class, 350–351, 354
- Mobile emulator, 70–72**
- Mobile Information Device Profile (MIDP)-based cell phones, 7**
- mod keyword, 26**
- mod operator, 122, 124**
- mode variable (Blend class), 687**
- Modifying**
 - arrays, 176
 - sequences, 162
- Monitoring playback, 634–635**
- MotionBlur class, 659–660**
- Mouse cursors**
 - CROSSHAIR, 376
 - cursor variable, 375
 - customizing, 377
 - DEFAULT, 376
 - E_RESIZE, 376
 - HAND, 376
 - H_RESIZE, 376
 - javafx.scene.Cursor class, 375
 - Microsoft Windows Platform, 376
 - MOVE, 376
 - NE_RESIZE, 376
 - nodes, 375
 - N_RESIZE, 376
 - NW_RESIZE, 376
 - SE_RESIZE, 376
 - setting, 375–377
 - shape of, 375
 - S_RESIZE, 376
 - SW_RESIZE, 376
 - TEXT, 376
 - V_RESIZE, 376
 - WAIT, 376
 - W_RESIZE, 376
- Mouse events**
 - delivery of, 419–424
 - dragging, 415–418
 - hover variable, 424–425
 - modifier keys, 414–415
 - mouse buttons, 410–414
 - mouse wheel, 418–419
 - mouse-related node state, 424
 - pickOnBounds variable, 424
 - propagation, 406–410
 - reporting of, 402–403
- MouseEvent class**
 - altDown variable, 414
 - button variable, 410

- clickCount variable, 410
- controlDown variable, 414
- defined, 403
- dragAnchorX variable, 416
- dragAnchorY variable, 416
- dragX variable, 416
- dragY variable, 416
- metaDown variable, 414
- middleButtonDown variable, 410
- node variable, 403
- popupTrigger variable, 411
- primaryButtonDown variable, 411
- sceneX variable, 404
- sceneY variable, 404
- screenX variable, 404
- screenY variable, 404
- secondaryButtonDown variable, 411
- shiftDown variable, 414
- source variable, 403
- variables, 403-404
- wheelRotation variable, 419
- x variable, 404
- y variable, 404

MOVE cursor, 376

Moving nodes, 9

MP3 files, 629

Multicolor linear gradients, 387

Multiline comments, 18

mute variable (MediaPlayer class), 634

N

N/A variable (GET operation), 954

name variable

- Font class, 478
- Resource class, 302
- SwingComponent class, 830

Naming

- projects, in Eclipse, 40
- projects, in NetBeans, 34

NaN, 128-129

nativearray keyword, 26

NE_RESIZE cursor, 376

Nesting

- classes, 266
- groups, 364
- object literals, 7, 149
- sequences, 154

NetBeans

- development, 3, 33-39
- documentation, 43-44
- installing, 33
- Java Web Start, 66-67
- mobile emulator, 70-71
- profiler, 46-82
- support, 3, 13
- viewing documentation, 1055

Network access, 12-13

Networking APIs, 12-13

new keyword, 7, 26

new operator, 122, 131

newVals identifier, 230

unnnn escape sequence, 103

Node class

- disable variable, 426
- disabled variable, 426
- focused variable, 426
- focusTraversable variable, 426
- hover variable, 424-425

node variable

- MouseEvent class, 403
- Transition class, 614

Nodes

- bounds, 527-538
- centering, 577-579
- clipping, 523-526
- colors, 367, 378
- coordinate systems, 501
- coordinates, 362-364
- CSS properties, 1061-1062
- cursors, 375
- customizing, 482-485, 864
- defined, 9, 342-361, 375
- effects, 368-369, 654
- events, 367, 401-402

- example, 9-10
- groups, 9, 482-485
- identification, 361
- javafxnodes package, 375
- layouts, 153-501, 538-579
- MediaView node, 10, 628
- moving, 9
- organization, 361
- placement of, 9
- rotating, 9
- rotation, 508-511
- scaling, 9, 511-514
- shearing, 9, 515, 517
- Swing components, 830-832
- transforms, 501-504
- translation, 506-508
- types of, 341
- user interaction, 401
- user interfaces, 341
- visibility, 364-365
- z-axis, 372-374
- Z-order, 365-366

Nonproportional gradients, 390-391

not keyword, 26

not operator, 122

N_RESIZE cursor, 376

null keyword, 26

Null value for numeric data types, 102

Number data type, 91, 99

Number overflow, 126

Number underflow, 126-127

Numeric data types

- converting numeric values, 99-101
- Integer, 91, 97-99
- null value, 102
- Number, 91, 99
- value ranges, 98

Numeric operations, 123

NW_RESIZE cursor, 376

O

Object comparison, 133-134

Object literals

- binding, 196-199
- defined, 7, 121
- fonts, 480-481
- functions, 150-151
- nesting, 149
- syntax, 147
- uses, 148-149
- variables, 149-150

Objects

- comparing, 286-287
- creating, 95, 131
- initializing, 148-149
- instance variables, 95

oblique variable (Font class), 478

Observable class, 267

offsetX variable

- DisplacementMap class, 679, 683-684
- DropShadow class, 660
- InnerShadow class, 663

offsetY variable

- DisplacementMap class, 679
- DropShadow class, 660
- InnerShadow class, 663

oldVals identifier, 229

On keyword, 26

On2 Technologies utility, 630

onBuffering variable (MediaPlayer class), 635

onConnecting function, 953

onDone variable (Task class), 1009

onDoneConnect function, 953

onEndOfMedia variable (MediaPlayer class), 635

onError variable

- Media class, 628
- MediaPlayer class, 635
- MediaView class, 639

onEvent variable (PullParser class), 976

onException function, 953

onException variable (FeedTask class), 996

onForeignEvent variable (FeedTask class), 996
onMouseClicked variable, 402
onMouseDragged variable, 402
onMouseEntered variable, 402
onMouseExited variable, 402
onMouseMoved variable, 402
onMousePressed variable, 402
onMouseReleased variable, 402
onMouseWheelMoved variable, 402
onReadingHeaders function, 954
onRepeat variable (MediaPlayer class), 635
onResponseCode function, 954
onStalled variable (MediaPlayer class), 635
onStarted function, 953
onStarted variable (Task class), 1009
opacity variable
 Blend class, 687
 Stage class, 354
Operators
 and, 122
 as, 122, 131-132
 += (add and assign), 122
 + (addition), 122
 = (assignment), 122
 /= (divide and assign), 122
 / (division), 122
 == (equality), 122, 133-134, 157
 . (function invocation), 131
 != (inequality), 122, 133-134
 * (multiplication), 122
 *= (multiply and assign), 122
 { object literal }, 122
 ++ (postfix), 122
 — (postfix), 122
 ++ (prefix), 122
 — (prefix), 122
 -= (subtract and assign), 122
 - (subtraction), 122
 - (unary minus), 122
 == operator, 8
 arithmetic operations, 123-125
 Boolean operations, 129-130

comparison operators, 127-128
 division of a nonzero value by zero, 127
 division of zero by zero, 127
 evaluation order, 121-123
 function(), 122
 greater than (>), 122
 greater than or equal to (>=), 122
 indexOf, 122
 infinity, 128-129
 instanceof, 122, 132-133
 integer overflow, 125-126
 less than (<), 122
 less than or equal to (<=), 122
 mod, 122, 124
 NaN, 128-129
 new, 122, 131
 not, 122
 number overflow, 126
 number underflow, 126-127
 numeric operations, 123
 object comparison, 133-134
 or, 122
 () parentheses, 122-123
 priority values, 121
 range errors, 125
 reverse, 122, 154-155
 sizeof, 122
 supported, 121

Or keyword, 26

or operator, 122

Outline dashing, 460-462

OutputStream, 952

override keyword, 26

Overriding variables, 259-261

P

Pack200 compression, 1043

package keyword, 26

Package mapping, 297

package modifier, 120

package statement in source files, 15, 20

package/file.class file, 1050

Packages

- creating, in Eclipse, 41
 - creating, in NetBeans, 38
 - javafxanimation, 591
 - javafx.async, 949
 - javafx.binding, 194
 - javafx.classes, 240
 - javafx.controls, 735
 - javafx.data, 949
 - javafx.data.pull, 950
 - javafx.effects, 651
 - javafx.import, 703
 - javafx.io.http, 950
 - javafx.media, 628
 - javafx.nodes, 375
 - javafx.reflect, 309
 - javafx.reflection, 309
 - javafx.scene.effects, 651
 - javafx.scene.effects.lighting, 651
 - javafx.shapes, 433
 - javafx.style, 809
 - javafx.swing, 829
 - java.lang.reflect, 309
 - platformapi, 285
- Packaging**
- desktop applications, 1026–1029
 - javafxpackager command-line tool, 1025
 - mobile applications, 1045–1047
 - options, 14
 - SnowStorm application, 46
- pad variable (InvertMask class), 671**
- paint variable (Flood class), 669**
- Panel class, 884–887**
- Panel container, 572–574**
- ParallelCamera, 373–374**
- ParallelTransition class, 622–625**
- Parameters, setting, 1037–1039**
- () parentheses in operators, 122–123**
- Parsing XML**
- DOM-style parsing, 975
 - pull parsing, 975–981
 - stream parsing, 975–981

Passing a function to another function, 142–144

PasswordBox class, 739, 771–773

PasswordBox controls

- appearance, 739
- CSS properties, 1070
- features and functionality, 771
- fx-columns property, 1070
- fx-echo-char property, 1070
- fx-editable property, 1070
- fx-font property, 1070
- fx-select-on-focus property, 1070
- operation of, 772–773
- PasswordBox class, 739, 771–773

Path class, 451

PathElement class, 434–453

PathTransition class, 619–622

paused variable

- MediaPlayer class, 635
- Timeline class, 594

PauseTransition class, 622

Pausing animation, 608

Pausing audio/video, 632

percentDone variable (Task class), 1010

PerspectiveCamera, 372–374

PerspectiveTransform class, 673–678

Photoshop (Adobe), 14, 704

Physical fonts, 476–477

pickOnBouonds variable, 424

Pie charts, 914, 919

placeholder variable (Image class), 488

Platform API

- platformapi package, 285
- security, 285

Platform dependencies, 14–15

platformapi package, 285

Playing audio/video, 10, 627–628

PointLight class, 697–698

Polygon class, 440–442

Polyline class, 440–442

popupTrigger variable (MouseEvent class), 411

Position

- fonts, 475-476
- label content, 836-838
- text, 466-468

position variable (Font class), 478

POST requests, 971, 973

postinit block, 265

postinit keyword, 26

Postponing actions, 294-295

Predefined variables, 31

preserveRatio variable

- Image class, 488
- ImageView class, 498
- MediaView class, 640

pressed state, 816

pressedIcon variable (SwingButton component), 844

Previewing graphics, 708-710

primaryButtonDown variable (MouseEvent class), 411

print() function, 286

println() function, 9, 285-286

Priority values of operators, 121

Prism toolkit, 372-374

private keyword, 26

Production Suite

- downloading, 14
- features, 14

-profile name compiler option, 1050

__PROFILE__ variable, 31

Profilers

- jvisualvm, 46
- NetBeans, 46-82

Profiles and APIs, 4-5

Profiling

- CPU profiling, 78-80
- memory profiling, 80-82
- SnowStorm application, 46-82

Progress monitoring, 1010-1014

progress variable

- Image class, 488
- Task class, 1009

ProgressBar controls, 739, 804-807

ProgressIndicator controls, 739, 804-807

progressMax variable (Task class), 1009

Projects

- creating, in Eclipse, 40
- creating, in NetBeans, 34
- naming, in Eclipse, 40
- naming, in NetBeans, 34
- source files, in Eclipse, 41
- source files, in NetBeans, 34-36

proportional variable

- linear gradients, 384
- radial gradients, 393

protected keyword, 26

protected modifier, 120

public keyword, 26

public modifier, 120

public-init keyword, 26

public-read keyword, 26

public-read modifier, 120

public-read package modifier, 120

public-read protected modifier, 120

PullParser class, 949, 975

PUT requests, 965, 971

Q

QuadCurve class, 444-448

QuadCurveTo class, 453

Querying sequences, 158-161

Question alert, 371

R

Radial gradients

- changing the bounds of a filled shape, 395-397
- cycle methods, 398-399
- defined, 392-393
- example, 393-395
- focus, 399-401
- javafx.scene.paint.RadialGradient class, 393
- moving the center, 397-398
- variables, 393

RadioButton controls, 756-759**radius variable**

- DropShadow class, 660
- GaussianBlur class, 656
- InnerShadow class, 663
- MotionBlur class, 659
- RadialGradient class, 393
- Shadow class, 661

random() function, 7**Range errors, 125****Range notation, 156-157****rate variable**

- MediaPlayer class, 631
- Timeline class, 594

read variable (GET operation), 955**readable variable (Resource class), 302****Reading data, 299, 301****reading variable (GET operation), 954****readingHeaders variable (GET operation), 954****Rectangle class, 433, 436, 438**

- fx-arc-height property, 1064
- fx-arc-width property, 1064
- height and width of rounded corners, 1064

References

- to classes, 23
- to methods, 24

Reflection

- access to classes, 310-314
- Application Programming Interfaces (APIs), 309
- FXContext class, 309
- javafx.reflect package, 309
- javafxreflection package, 309
- java.lang.reflect package, 309
- reflection context, 309-310

Reflection class, 671, 673**Rejecting an invalid value, 225-227****Removal operation with sequence variables, 230-232****Removing**

- breakpoints, 76-77
- data, 303-305

- elements from sequences, 165-166
- mappings, 298

repeatCount variable

- MediaPlayer class, 631
- Timeline class, 594

Repeating

- animation, 605-607
- audio/video playback, 632-633

replace clause, 229-230**replace keyword, 26****Replacement operations with sequence variables, 230-231****Replacing**

- elements of sequences, 162-163
- a range of elements in a sequence, 166-167

Reserved words, 26**resizable variable (Stage class), 348****Resource class, 302-303****Resource names, 305-306****responseCode variable (GET operation), 954****responseMessage variable (GET operation), 954****Restarting animation, 609****RESTful web services, 13, 974-975****Restricting audio/video playback, 632-633****return keyword, 26****return statement, 26****reverse keyword, 26****reverse operator, 122, 154-155****Reverting to the previous value, 227-229****rotatable variable (MediaView class), 648****rotate variable, 504****RotateTransition class, 616-617****Rotating nodes, 9****Rotation of nodes, 508-511****rotationAxis variable, 504****RSS feeds, 949-950, 1004****run target, 1055****run() function, 28-29, 287****Running**

- applications, with command-line tools, 1049-1052
- applications, with Eclipse, 42

- applications, with NetBeans, 39
- SnowStorm application, as an applet, 67-70
- SnowStorm application, from JavaFX TV emulator, 72
- SnowStorm application, from Java Web Start, 46-67
- SnowStorm application, on a mobile emulator, 70-72
- running variable (Timeline class), 595**
- Runtime libraries**
 - animation, 10-12
 - audio, 10
 - features, 3, 9
 - network access, 12-13
 - user interface classes, 9-10
 - video, 10

S

- saturation variable (ColorAdjust class), 670
- Scalable Vector Graphics (SVG), 704-705, 733-735**
- Scalable Vector Graphics (SVG) editor, 14**
- ScaleTransition class, 617-618**
- scaleX variable, 504, 679, 684**
- scaleY variable, 504, 679, 684**
- scaleZ variable, 504**
- Scaling nodes, 9, 511-514**
- Scatter charts, 932-934**
- Scene class, 342-360**
- sceneX variable (MouseEvent class), 404**
- sceneY variable (MouseEvent class), 404**
- Scope of variables, 93**
- Screen class, 585-590**
- screenX variable (MouseEvent class), 404**
- screenY variable (MouseEvent class), 404**
- Script files**
 - classes, 265-266
 - creating, 41
- Script functions, 137-138**
- Script variables, binding, 194**
- Scripting mode, 31**

Scripts

- Ant build script, 33, 1049, 1052-1055
- JavaFX Script language, 8-9
- run() function, 28-29

ScrollBar class, 788-789

ScrollBar controls

- appearance, 739
- CSS properties, 1068-1069
- features and functionality, 787-788
- fx-block-increment property, 1069
- fx-click-to-position property, 1069
- fx-unit-increment property, 1069
- fx-vertical property, 1069
- operation of, 790-794
- ScrollBar class, 788-789

Scroller class, 794

ScrollView controls

- appearance, 739
- CSS properties, 1069
- display and values, 797
- features and functionality, 794
- fx-fit-to-height property, 1069
- fx-fit-to-width property, 1069
- fx-hbar-policy property, 1069
- fx-pannable property, 1069
- fx-vbar-policy property, 1069
- scrollable node size, 797
- ScrollView class, 794-797

SDK (Software Development Kit)

- Ant task, 13
- API documentation, 5
- application launcher, 13
- command-line compiler, 13
- command-line tools, 1049
- contents of, 13
- extracting documentation, 1049
- installing, 1049
- javafxdoc command-line tool, 1055-1058
- javafxpackager command-line tool, 13

Searching

- fonts, 481-482
- sequences, 172-173

Searching sequences for elements, 168-169**secondaryButtonDown** variable (**MouseEvent** class), 411**Security, 3, 285****select()** function, 712**selectedIndex** variable (**SwingComboBox** component), 854**selectedItem** variable (**SwingComboBox** component), 854**Selecting**

fonts, 479-482

functions by classname, 258

selectOnFocus variable (**SwingTextField** class), 839**Self-signed certificate, 1044****Separator control**

appearance, 739

CSS properties, 1069

-fx-hpos property, 1069

-fx-vpos property, 1069

Separator controls

adding, 808

features and functionality, 807

Separator class, 807-808

SepiaTone class, 673**Sequences**

binding, 217

comparing, 167-168

copying, 158

creating, 153-155

defined, 153

equality, 157

finding largest and smallest elements, 169-171

immutable, 154

inserting elements, 163-165

iterating over, 184-190

modifying, 162

nesting, 154

obtaining an element of, 159

obtaining part of, 160

obtaining size of, 158

querying, 158-161

range notation, 156-157

removing elements, 165-166

replacing a range of elements, 166-167

replacing elements of, 162-163

searching, 172-173

searching for elements, 168-169

shuffling, 174

size of, 154

sorting, 171-172

string form, 155

syntax, 153

triggers, deletion operations, 230-232

triggers, example of, 234-236

triggers, insertion operations, 230, 233-234

triggers, replacement operations, 230-231

triggers, syntax of, 229-230

SequentialTransition class, 622-625**SE_RESIZE** cursor, 376**Setting**

breakpoints, 46-73

cursors, 375

parameters, 1037-1039

Shadow class, 664-665**Shape** class, 433**shape of cursors, 375****ShapeIntersect** class, 449-451**Shapes**

Arc class, 442-444

ArcTo class, 453, 455

Circle class, 433, 438-439

CSS properties, 1063-1064

CubicCurve class, 448-449

CubicCurveTo class, 453

defined, 433

DelegateShape class, 434

Ellipses class, 439-440

fills, 434, 463-466

groups, 482

Line class, 434-436

morphing, 603, 605

outline dashing, 460-462

- Path class, 451
- PathElement class, 452-453
- Polygon class, 440-442
- Polyline class, 440-442
- QuadCurve class, 444-448
- QuadCurveTo class, 453
- Rectangle class, 433, 436-438
- Shape class, 433
- ShapeIntersect class, 449-451
- ShapeSubtract class, 449-451
- strokes, 456, 458
- SVGPath class, 456
- ShapeSubtract class, 449-451**
- Shearing nodes, 9, 515, 517**
- shiftDown variable**
 - KeyEvent class, 431
 - MouseEvent class, 414
- Short data type, 91**
- Shuffling sequences, 174**
- Shutdown, 292-294**
- Signing applications and applets, 1043-1045**
- Simple variables and triggers, 221**
- Single-line comments, 18**
- size variable (Font class), 478**
- sizeof keyword, 26**
- sizeof operator, 122**
- Skin class, 891-894**
- Slicing strings, 108**
- Slider controls**
 - appearance, 739
 - CSS properties, 1069-1070
 - features and functionality, 797
 - fx-block-increment property, 1070
 - fx-click-to-position property, 1070
 - fx-major-tick-unit property, 1070
 - fx-minor-tick-unit property, 1070
 - fx-show-tick-labels property, 1070
 - fx-show-tick-marks property, 1070
 - fx-snap-to-ticks property, 1070
 - fx-vertical property, 1070
 - operation of, 798-800
 - Slider class, 797-798
 - tick labels, 802-804
 - tick marks, 800-802
- smooth variable**
 - Image class, 488
 - ImageView class, 498
 - MediaView class, 640
- SnowStorm application**
 - animation, 58
 - background image, 49-52, 581-582
 - building, 46
 - creating, 46-47
 - debugging, 46-76
 - desktop application, 70
 - dynamic part, 47
 - functionality, 45
 - packaging, 46
 - performance problems, 46
 - previewing, 52-53
 - profiling, 46-82
 - running, as an applet, 67-70
 - running, on a mobile emulator, 70-72
 - running, with JavaFX TV emulator, 72
 - running, with Java Web Start, 46-67
 - screen display, 579-581
 - snowflakes, adding, 58-62, 582-585
 - snowflakes, animating, 62-64, 582-585
 - snowflakes, counting, 64-65
 - source code, 46-85
 - stage and scene, 48-49
 - static part, 47
 - text, 53-58
 - user interface, 47
- SOAP-based web services, 950**
- Software Development Kit (SDK)**
 - Ant task, 13
 - API documentation, 5
 - application launcher, 13
 - command-line compiler, 13
 - command-line tools, 1049
 - contents of, 13
 - extracting documentation, 1049
 - installing, 1049

javafxdoc command-line tool,
1055-1058
javafxpackager command-line tool, 13

Sorting sequences, 171-172**Source code**

documentation, 43
example source code, 33, 1049
SnowStorm application, 46-85

Source files

class files, 1050
comments, 18
documentation, 1057
.fx naming suffix, 15
import statement, 15, 20
package statement, 15, 20
structure, 15

source variable

Identity class, 668
Media class, 628
MouseEvent class, 403

__SOURCE_FILE__ variable, 31**-sourcepath path compiler option, 1051****specularConstant variable (Lighting class), 691****specularExponent variable (Lighting class), 691****Speeding up audio/video playback, 632****Spotlight class, 698, 700****spread variable (DropShadow class), 660****S_RESIZE cursor, 376****Stack class, 555-559****Stack container**

CSS properties, 1065-1066
-fx-node-hpos property, 1065
-fx-node-vpos property, 1065
-fx-padding property, 1066
-fx-snap-to-pixel property, 1064
Stack class, 555-559

Stage class

appearance, 342-343
bounds, 348-350
closing the stage, 347-348
extensions, 357-358
features, 342
focus, 344-345

icons, 343-344
mobile devices, 350-351, 354
opacity, 354-357, 367
stage position and size, 348
states, 342-343
style, 354, 357
variables, 342
visibility, 344
Z-order control, 345-347

stage variable (Scene class), 358**started variable (GET operation), 953****started variable (Task class), 1009****start() function, 951, 955****startTime variable (MediaPlayer class), 631****startX variable, 384****startY variable, 384****States**

Button controls, 751-752
of classes, 239
disabled state, 816
focused state, 816
hover state, 816
pressed state, 816
Stage class, 342-343
state-dependent CSS styling, 816-817

static keyword, 26**Static methods, 24****status variable (MediaPlayer class), 634-635****step keyword, 26****Stepping through code, 75-76****stop() function, 951****stopped variable (Task class), 1009****Stopping animation, 608-609****Stopping audio/video playback, 632****stops variable**

linear gradients, 384
radial gradients, 393

stopTime variable (MediaPlayer class), 631**Storage class**

available and used space, 302
listing resources, 301-302

Streams in HttpRequest class, 951**Strikethrough text, 471**

- String data type, 92, 103**
- String form of sequences, 155**
- String literals**
 - concatenation, 104-105
 - defined, 103
 - escape sequences, 103
 - internationalization, 109-110
 - localization, 108-109
 - slicing, 108
- Strokes**
 - linear gradients, 391-392
 - shapes, 456, 458
 - text, 470-471
- Structuring source files, 15**
- Style sheets. See CSS (Cascading Style Sheets)**
- style variable**
 - Font class, 478
 - Stage class, 354
- stylesheets variable (Scene class), 358**
- Subclassing, 249**
- substring() function, 108**
- succeeded variable (Task class), 1009**
- super keyword, 26**
- supportsMultiViews variable (MediaPlayer class), 640**
- surfaceScale variable (Lighting class), 691-692**
- SVG (Scalable Vector Graphics), 704-705, 733-735**
- SVG (Scalable Vector Graphics) editor, 14**
- SVGPath class, 456**
- SVG-to-JavaFX Graphics Converter, 734**
- Swing controls**
 - accessing the wrapped Swing control, 832
 - CSS (Cascading Style Sheets), 809
 - javafxswing package, 829
 - nodes, 830-832
 - support for, 829
 - wrappers, 829-830, 860-864
- Swing JColorChooser component, 382-383**
- SwingButton component**
 - defined, 829, 843
 - variables, 843-846
- SwingCheckBox component, 829, 848-849**
- SwingComboBox component, 830, 854, 857**
- SwingComponent class, 830**
- SwingLabel component, 829, 833-834**
- SwingList component, 830, 849-850, 852**
- SwingRadioButton component, 829, 848-849**
- SwingScrollPane component, 830, 852-853**
- SwingSlider component, 830, 857, 859**
- SwingTextField component**
 - configuring, 840-842
 - defined, 829
 - handling input, 842-843
 - variables, 838-839
- SwingToggleButton component, 829, 846-848**
- SW_RESIZE cursor, 376**
- Synchronizing external events with playback, 637-638**
- Syntax**
 - class declarations, 240
 - def statement, 92
 - if statement, 179
 - JavaFX Script, 3
 - object literals, 147
 - sequences, 153
 - triggers, 221, 229
 - var statement, 89
 - while statement, 181-182
- System properties**
 - example.class, 1055
 - getting, 290
 - javafx.applet.codebase, 291
 - javafx.application.codebase, 291
 - javafx.encoding, 291
 - javafx.file.separator, 291
 - javafx.java.ext.dirs, 291
 - javafx.java.io.tmpdir, 291
 - javafx.java.vendor, 291
 - javafx.java.vendor.url, 291
 - javafx.java.version, 291
 - javafx.language, 291
 - javafx.line.separator, 291
 - javafx.os.arch, 291
 - javafx.os.name, 291

- javafx.os.version, 291
- javafx.path.separator, 291
- javafx.region, 291
- javafx.runtime.version, 291
- javafx.timezone, 291
- javafx.user.dir, 291
- javafx.user.home, 291
- javafx.version, 291
- mobile, 292

T

Targets

- clean, 1055
- run, 1055

Task class,**Tasks, 13, 1009-1010****Text**

- alignment, 468-469
- decoration, 471-472
- displaying, 467
- fills, 470-471
- fonts, 472
- fx-font property, 1062
- fx-strikethrough property, 1062
- fx-text-alignment property, 1062
- fx-text-origin property, 1062
- fx-underline property, 1062
- multiline, 468-469
- overline, 471
- positioning, 466-468
- strikethrough, 471
- strokes, 470-471
- underline, 471
- wrapping, 468-469

Text class, 433, 466, 472**TEXT cursor, 376****Text node, 424****text variable**

- KeyEvent class, 429
- SwingButton component, 844
- SwingComboBox component, 854

- SwingLabel class, 833-835

- SwingTextField class, 839

TextBox control

- appearance, 739
- content, setting and getting, 766-769
- copy and paste, 771
- CSS properties, 1070
- editability, 764
- fx-columns property, 1070
- fx-editable property, 1070
- fx-font property, 1070
- fx-lines property, 1070
- fx-select-on-focus property, 1070
- height, 764
- selection, 769-771
- TextBox class, 762-764
- TextInputControl class, 761-762
- width, 764

TextOrigin class, 467-468**then keyword, 26****this keyword, 26****threshold variable (Bloom class), 665****throw keyword, 26****throw statement, 26****tick labels (Slider controls), 802-804****tick marks (Slider controls), 800-802****Tile class, 563-572****Tile container**

- CSS properties, 1066
- fx-snap-to-pixel property, 1064
- Tile class, 563-572

Time literals, 595-596**time variable**

- MediaTimer class, 638
- Timeline class, 594

Timeline class, 12, 594, 605**Timelines**

- Duration class, 596
- example, 592-594
- functionality, 591
- KeyFrame class, 600
- keyframes, 596

- time literals, 595-596
- timer, 611-613
- variables, 594-595
- TimeSlider class, 904-908**
- title variable (Stage class), 342**
- ToggleButton controls, 739, 756-758**
- Tooltip controls**
 - appearance, 739
 - CSS properties, 1067-1068
 - features and functionality, 808-809
- topInput variable (Blend class), 687**
- topOffset variable (Reflection class), 672**
- topOpacity variable (Reflection class), 672**
- toRead variable (GET operation), 954**
- toString() function, 155**
- totalDuration variable (Timeline class), 594**
- Tracking mouse motion, 404-406**
- tracks variable (Media class), 628**
- transformable variable (MediaView class), 648**
- Transformations, 654**
- Transforms**
 - combining, 517, 523
 - defined, 501
 - functionality, 501
 - javafx.scene.transform.Transform class, 505
 - layoutX variable, 504
 - layoutY variable, 504
 - order of multiple transfer effects, 517-520
 - rotate variable, 504
 - rotationAxis variable, 504
 - scaleX variable, 504
 - scaleY variable, 504
 - scaleZ variable, 504
 - transforms variable, 504
 - translateX variable, 504
 - translateY variable, 504
 - translateZ variable, 504
 - variables, 501-505
- transforms variable, 504**
- Transition class, 614**
- Transitions**
 - animation, 591, 613-614
 - video, 646-649
- TranslateTransition class, 614, 616**
- translateX variable, 504**
 - Timeline class, 12
- translateY variable, 504**
- translateZ variable, 371, 504**
- Translation of nodes, 506-508**
- trigger keyword, 26**
- Triggers**
 - attaching, 221, 229
 - binding, 223-224
 - constraints, 224-225
 - creating, 221
 - declaring, 221-222
 - defined, 26
 - deletion operations, 230-232
 - executing, 221-222
 - getting the previous value of variables, 223
 - insertation operations, 230, 233-234
 - javafx/SequenceTriggers.fx file, 230
 - javafxtriggers package, 221
 - media players, 635-636
 - mixins, 280-281
 - rejecting an invalid value, 225-227
 - replacement operations, 230-231
 - reverting to the previous value, 227-229
 - syntax, 221, 229
 - uses, 221
 - variables, instance, 236-237
 - variables, sequence, 229-230, 234-236
 - variables, simple, 221
- trim() function, 8**
- true keyword, 26**
- try keyword, 26**
- try/catch/finally statements, 26, 194**
- TV applications, deploying, 14**
- TV emulator, 72, 371**
- tween keyword, 26**
- Twitter search feed, 1004, 1008**

Twitter web service client, 983-985, 987

Type inference, 6, 117-119

typeof keyword, 26

Types

- FXClassType, 315-316
- FXFunctionType, 316-317
- FXJavaArrayType, 316
- FXPrimitiveType, 315
- FXSequenceType, 316
- FXType, 314-315
- representation of, 314

U

ulx variable (PerspectiveTransform class), 674

uly variable (PerspectiveTransform class), 674

Unbound functions, 207-209, 214

Underline text, 471

Unicode

- Character data type, 101
- escape sequences, 103-104
- String data type, 103

Unidirectional binding, 24-25, 206

updateViewport() function, 501

url variable (Image class), 488

urx variable (PerspectiveTransform class), 674

ury variable (PerspectiveTransform class), 674

user gestures, 789-790

User interaction

- charts, 914, 936-937
- nodes, 401

User interface classes, 9-10

User interfaces

- nodes, 341
- SnowStorm application, 47

V

value variable (SwingSlider component), 857

Values

- assigning, 23
- FXFunctionValue class, 320
- FXObjectValue class, 319

FXPrimitiveValue class, 318-319

FXSequenceValue class, 320

FXValue class, 318

getValue() method, 316

representation of, 316

var keyword, 6, 26

var statement, 89

Variables

- binding, 194-196
- changing values, 74
- data types, 6, 89
- declaring, 6, 23, 89-95
- __DIR__, 31
- __FILE__, 31
- functions, 137-142
- FXVarMember class, 322
- getting all variables of a class, 323
- getting previous value of variables, 223
- inspecting, 73
- instance variables, 95
- linear gradients, 384
- named variables, 322-323
- object literals, 149-150
- overriding, 259-261
- predefined, 31
- __PROFILE__, 31
- radial gradients, 393
- reserved words, 26
- scope, 93
- __SOURCE_FILE__, 31
- triggers, instance variables, 236-237
- triggers, sequence variables, 229-230, 234-236
- triggers, simple variables, 221
- Type inference, 6
- type inference, 6, 117-119
- visibility, 119-120

VBox class, 562-563

VBox container

- CSS properties, 1066
- fx-hpos property, 1066
- fx-node-hpos property, 1066
- fx-snap-to-pixel property, 1064

- fx-spacing property, 1066
 - fx-vpos property, 1066
 - VBox class, 562-563
 - verbose compiler option, 1051**
 - verify variable (SwingTextField class), 840**
 - Versions of JavaFX runtime, 4**
 - vertical variable (SwingSlider component), 857**
 - verticalAlignment variable**
 - SwingButton component, 844
 - SwingLabel class, 833-834, 837
 - verticalTextPosition variable**
 - SwingButton component, 844
 - SwingLabel class, 834
 - Video**
 - Application Programming Interfaces (APIs), 627
 - controlling media playback, 630-631
 - effects, 646
 - FLV, 630
 - FXM, 630
 - media players, 627
 - MP3 files, 629
 - pausing, 632
 - playing, 10, 627-628
 - repeating playback, 632-633
 - restricting playback, 632-633
 - size and position of video frame, 640-644
 - speeding up playback, 632
 - stopping playback, 632
 - support, 10
 - transitions, 646-649
 - viewport, 644-646
 - volume control, 633
 - Viewing documentation, 1055-1056**
 - viewport variable**
 - ImageView class, 498
 - MediaView class, 640
 - Visibility**
 - of classes, 242-243
 - of functions, 147, 255-256
 - of nodes, 364-365
 - Stage class, 344
 - of variables, 119-120
 - visible variable (Stage class), 342**
 - VM settings for applets, 15**
 - Volume control (media players), 633**
 - volume variable (MediaPlayer class), 634**
 - V_RESIZE cursor, 376**
-
- ## W
-
- WAIT cursor, 376**
 - warp effect, 684-686**
 - Web services**
 - RESTful, 13, 949-975
 - SOAP-based, 950
 - weight variable (Image class), 488**
 - wheelRotation variable (MouseEvent class), 419**
 - where keyword, 26**
 - while keyword, 26**
 - while statement, 26, 181-182**
 - width variable**
 - BoxBlur class, 657
 - DropShadow class, 660
 - Flood class, 669
 - InnerShadow class, 663
 - Media class, 628
 - Scene class, 358
 - Shadow class, 661
 - Stage class, 348
 - with keyword, 28**
 - wrap variable (DisplacementMap class), 679, 682-683**
 - Wrapping text, 468-469**
 - W_RESIZE cursor, 376**
 - writable variable (Resource class), 302**
 - Writing**
 - applets, 4
 - data, 298-301
 - MIDlets, 4
 - to the system output stream, 286

X

x-axis, 372

x variable

- Flood class, 669
- Identity class, 668
- ImageView class, 498
- MediaView class, 639
- MouseEvent class, 404
- Scene class, 358
- Stage class, 348

-XDdumpjava compiler option, 1055

XML (Extensible Markup Language)

- HttpRequest class, 949
- PullParser class, 949

XML parsing

- DOM-style parsing, 975
- pull parsing, 975, 981
- stream parsing, 975-976, 981

Y

y-axis, 372

y variable

- Flood class, 669
- Identity class, 668
- ImageView class, 498
- MediaView class, 639
- MouseEvent class, 404
- Scene class, 358
- Stage class, 348

Yahoo! RSS Weather Feed, 998, 1004

Z

z-axis, 372-374

Zero

- division of a nonzero value by zero, 127
- division of zero by zero, 127

Z-order

- nodes, 365-366
- Stage class, 345-347