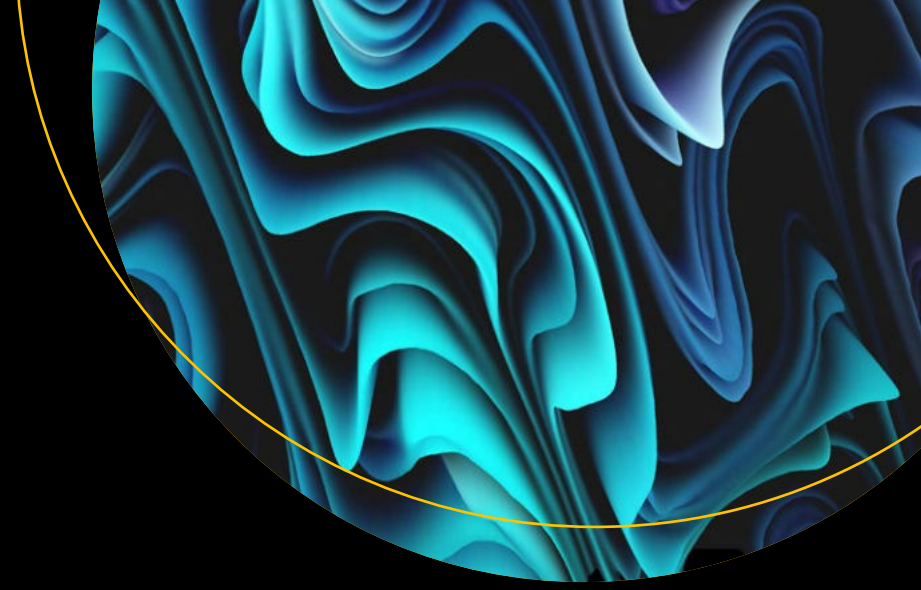

Frontend Development with JavaFX and Kotlin

Build State-of-the-Art Kotlin GUI Applications

Peter Späth

Apress®

Peter Späth

Frontend Development with JavaFX and Kotlin

Build State-of-the-Art Kotlin GUI Applications

Apress®

Peter Späth
Leipzig, Sachsen, Germany

ISBN-13 (pbk): 978-1-4842-9716-2 ISBN-13 (electronic): 978-1-4842-9717-9
https://doi.org/10.1007/978-1-4842-9717-9

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: Laura Berendson
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image by Jerryyaar Designer on Pixabay (www.pixabay.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC, and the sole member (owner) is Springer Science + Business Media Finance Inc. (SSBM Finance Inc). SSBM Finance Inc. is a **Delaware** corporation.
For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.
Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.
Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (https://github.com/Apress). For more detailed information, please visit https://www.apress.com/gp/services/source-code.

Paper in this product is recyclable

Contents

1 Getting Started 1
Introduction 1
Gradle for JavaFX and Kotlin 2
A HelloWorld Project 4
Setting Up for Eclipse 5
Setting Up for IntelliJ 10
Kotlin and Java Interoperability 13
A Note About Kotlin Utilities for JavaFX 14
A Note About FXML 16
A Note About Downloading JavaFX Releases 18
Build Setup for This Book 18

2 Properties 19
Why you Should use Properties 19
One-Way and Two-Way Bindings 22
Custom Bindings 26
About Observable Collections 26
Summary 30

3 Stages and Scenes 33
About Screens 33
Using Stages and the Application Class 35
Dialog-Like Stages 38
The JavaFX Application Thread 39
About Scenes 39
Position and Size 40
Camera 41
Cursor 41
Mnemonic and Accelerators 42
Focus 43
Node Lookup 43
Snapshots 44
Fill and Other Styles 45
Keyboard 46
Mouse Events 48
Mouse Event Handling 50

Mouse Drag Event Handling . . . 52
Gestures . . . 53
Summary . . . 57

4 Containers . . . 59
StackPane . . . 61
VBox and HBox . . . 63
FlowPane . . . 65
GridPane . . . 67
TilePane . . . 69
BorderPane . . . 71
AnchorPane . . . 72
Styling Panes . . . 73
Adding Stylesheets to the Whole Scene . . . 73
Adding Stylesheets to Individual Panes . . . 73
JavaFX CSS Selectors for Panes . . . 74
JavaFX CSS Properties for Panes . . . 75
Summary . . . 78

5 Visual Nodes . . . 79
Node Coordinate Systems . . . 79
Shapes . . . 81
Canvas . . . 83
Image Nodes . . . 84
Controls . . . 84
Text Fields and Text Areas . . . 84
Action Buttons . . . 86
Button Bars . . . 87
Menus . . . 88
Toolbars . . . 89
Checkboxes . . . 89
Radio Buttons . . . 90
Combo Boxes . . . 91
Sliders . . . 92
Miscellaneous Controls . . . 93
Control Panes . . . 93
Scroll Panes . . . 93
Accordions . . . 96
Tab Panes . . . 96
Split Panes . . . 97
Styling Visual Nodes . . . 97
Summary . . . 99

6 Lists and Tables . . . 101
Lists with ListView . . . 101
Tables with TableView . . . 104
Trees with TreeView . . . 109
Summary . . . 112

7 Events 113
What Events Are and Event Processing 113
Event Handlers and Filters 114
Drag and Drop Procedures 116
Summary 119

8 Effects and Animation 121
About Effects 121
Animating Your Scenes 122
Transitions 124
Timeline Animations 125
Summary 126

9 Concurrency 127
The JavaFX Concurrency Framework 127
About Kotlin Coroutines for JavaFX 131
Summary 134

Index 135

About the Author

Peter Späth graduated in 2002 as a physicist and soon afterward became an IT consultant, mainly for Java-related projects. In 2016, he decided to concentrate on writing books on various aspects, but with a main focus on software development. With two books about graphics and sound processing, three books on Android app development, and a couple of books about Java, Jakarta EE, and Kotlin, Peter continues his effort in writing software development-related literature.

About the Technical Reviewer

Massimo Nardone has more than 25 years of experience in security, web and mobile development, cloud, and IT architecture. His true IT passions are security and Android. He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years. He holds a Master of Science degree in Computing Science from the University of Salerno, Italy. He has worked as a CISO, CSO, security executive, IoT executive, project manager, software engineer, research engineer, chief security architect, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years. His technical skills include security, Android, cloud, Java, MySQL, Drupal, Cobol, Perl, web and mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, and more. He worked as visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (PKI, SIP, SAML, and Proxy areas). He is currently working for Cognizant as head of cyber security and CISO to help both internally and externally with clients in areas of information and cyber security, like strategy, planning, processes, policies, procedures, governance, awareness, and so forth. In June 2017 he became a permanent member of the ISACA Finland Chapter Board.

Massimo has reviewed more than 45 IT books for different publishing companies and is the coauthor of *Pro Spring Security: Securing Spring Framework 5 and Boot 2-based Java Applications* (Apress, 2019), *Beginning EJB in Java EE 8* (Apress, 2018), *Pro JPA 2 in Java EE 8* (Apress, 2018), and *Pro Android Games* (Apress, 2015).

Introduction

Building elegant and highly responsible, responsive, and stable Java client applications (fat clients) is a highly acceptable approach if security considerations or network availability speaks against web applications, or maintaining servers and server applications lies out of scope for your project. Additionally, using Kotlin as a programming language boosts code expressiveness and maintainability, allowing for a development yielding a clean code approach.

The book introduces JavaFX as a frontend technology and from the very beginning focuses on using Kotlin instead of Java for coding the program artifacts. Many listings and code snippets accompany the text, readily allowing for a hands-on learning style.

The Book's Targeted Audience

The book is for low- to mid-level Java or Kotlin developers with or without JavaFX experience, wishing to learn how to build JavaFX applications with Kotlin.

The readers will in the end be able to use Kotlin as a language for building basic to moderately advanced and elaborated apps targeting JavaFX.

Any experience in using JavaFX and frontend coding is not a requirement for reading the book. Being a Kotlin expert is not necessary either, but having read introductory-level books or studied online resources is surely helpful. The online documentation of Kotlin and JavaFX also provides valuable resources you can use as a reference while reading this book.

Source Code

All source code shown or referred to in this book can be found at github.com/apress/frontend-development-javafx-kotlin.

How to Read This Book

This book should be read sequentially to get the most benefit from it. Of course, you can skip one or the other chapter if you already gained knowledge elsewhere. Taking its introductory nature, the book is not meant to present a reference fully covering each and every aspect of Kotlin frontend programming or JavaFX, so also consulting the online documentation at

https://openjfx.io/
https://openjfx.io/javadoc/19/
https://kotlinlang.org/docs/home.html

while you are reading the book certainly is not a bad idea.

The book is split up into nine chapters. Chapter 1 gives a general introduction and presents hello world-style programs for Gradle, Eclipse, and IntelliJ.

Chapter 2 talks about using properties as data holders and addresses one- and two-way binding techniques for connecting controls and data in your program.

Chapter 3 introduces stages and scenes, which serve as primordial containers for visual artifacts.

Chapter 4 talks about containers and ways to lay out and style your scenes.

Chapter 5 handles nodes and controls including styling. These aspects usually constitute the biggest part of your project work speaking of time budget.

Chapter 6 presents lists and tables, which are particularly important for enterprise-level projects.

Chapter 7 is for summarizing and deepening our knowledge about event handling in JavaFX. This also includes drag and drop procedures.

Chapter 8 introduces effects and animation, improving user experience and giving your programs some eye candies.

As a prospect, Chapter 9 briefly introduces concurrency techniques, giving you a starting point for handling background processing needs.

Getting Started

1

In this chapter, we give a brief introduction to using JavaFX and Kotlin together, and we create "Hello World"–style projects for the command line, for Eclipse, and for IntelliJ IDEA.

Introduction

JavaFX is the dedicated fat client (desktop application) GUI toolkit for current Java releases. It is the replacement and successor of the venerable Java Swing technology. This switch happened around 2010, and since then JavaFX has been constantly improved and extended. With JREs up to version JDK 9, JavaFX was part of the Java distribution—with JDK 11 and later, it has to be installed separately.

The following features describe JavaFX:

- Built-in controls: Labels, editable text fields, buttons, combo boxes, checkboxes, radio buttons, menu bars, scrollbars, accordion, tabs, canvas (for drawing shapes and figures), color picker, pagination, 3D graphics (games, science, product presentation), WebView (presenting and interacting with web contents), dialogs, sliders, spinners, progress bars
- Lists, tables, trees
- Built-in layouts: AnchorPane (anchoring nodes to one of the edges or to the center point), BorderPane (placing nodes at bottom, top, right, left, center), FlowPane (placing nodes consecutively and wrapping at the boundaries), TilePane (same as FlowPane, but with all cells the same size), GridPane (placing nodes in a grid with cell sizes dynamically calculated and on demand spanning several rows and columns), VBox (placing nodes in columns), HBox (placing nodes in rows), StackPane (placing nodes in an overlay fashion)
- Animation (fade, fill, stroke, translate, rotate, scale, . . .), effects (glow, blend, bloom, blur, reflection, sepia, shadow, lighting)
- Nodes stylable via CSS
- Some built-in chart widgets
- Flexible and concise data binding via observable properties
- Descriptive layouting via FXML
- Module support (for JDK 9+)

P. Späth, *Frontend Development with JavaFX and Kotlin*,
https://doi.org/10.1007/978-1-4842-9717-9_1

- Graphics transformations and coordinate systems
- Media APIs
- Java Swing interoperability
- Comes as a set of JAR modules and native libraries
- An external Scene Builder for graphically creating scenes
- Printing API

In this book, we describe a subset of these features, giving you a starting point for your own projects.

Using Kotlin as a programming language instead of Java gives a boost to your coding experience. Just to give you an example, consider a button with a click handler. In Java, you'd write

```
Button btn = new Button();
btn.setText("Say 'Hello World'");
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

(255 characters) The very same code written in Kotlin reads

```
val btn = Button().apply {
   text = "Say 'Hello World'"
   setOnAction { _ ->
       println("Hello World!")
   }
}
```

With 142 characters, this is more than 40% shorter than the Java variant! And besides being shorter, it is also more expressive and by that easier to understand and easier to maintain.

Using some sufficiently nonobtrusive utility functions, this can even be further reduced to 81 characters in size:

```
val btn = Button("Say 'Hello World'") {
   println("Hello World!")
}
```

This works by Kotlin's ability to dynamically add additional constructors to classes.

Gradle for JavaFX and Kotlin

As a build tool, we use Gradle from https://gradle.org/. It is highly flexible, works on any operating system that provides a Java installation, and by means of plugins or preinstalled components can be operated from many IDEs.

I first describe the CLI mode for running Gradle builds. This is how you would use it in a server environment, but it also serves as a good starting point if you want to learn how to use Gradle inside an IDE workflow.

If not already present, get and install a version 17 JDK. Throughout the book, we will be using OpenJDK 17, but if chances are good you can also take Oracle's supported JDK 17 or a higher version from either Oracle or https://openjdk.org/ without any problems possibly coming up.

Note Using Oracle's JDK 17 or higher requires buying a license if you plan to use it for a longer term; see www.oracle.com/java/.

As a next step, fetch Gradle from https://gradle.org. In this book, we use version 7.6 from https://gradle.org/next-steps/?version=7.6&format=bin. In order to announce Java to Gradle, either make sure `java` and `javac` (with .bat extension on Windows) are in your PATH, or you have the environment variable JAVA_HOME point to your JDK installation folder (recommended). To simplify using Gradle, you can also put GRADLE-INST-DIR/bin (with GRADLE-INST-DIR pointing to your Gradle folder), or GRADLE-INST-DIR\bin for Windows, on the path.

Note In Linux, environment variables like PATH or JAVA_HOME get set via `export PATH=/bin:/usr/bin:/path/to/my/gradle/bin`.
In Windows, you must use the system settings dialog.

In order to check your Gradle installation, in a terminal enter

```
gradle -version
```

or, if Gradle is not in the path:

```
/path/to/gradle -version       (Linux)
C:\path\to\gradle.bat -version (Windows)
```

The output of the command should be similar to

```
------------------------------------------------------------
Gradle 7.6
------------------------------------------------------------

Build time:   2022-11-25 13:35:10 UTC
Revision:     daece9dbc5b79370cc8e4fd6fe4b2cd400e150a8

Kotlin:       1.7.10
Groovy:       3.0.13
Ant:          Apache Ant(TM) version 1.10.11 compiled on
              July 10 2021
JVM:          17.0.1 (Oracle Corporation 17.0.1+12-39)
OS:           Linux 5.15.0-56-generic amd64
```

Important is the "JVM:" line. The Kotlin version shown does *not* mean you would not be able to build applications running under a different Kotlin version—it just tells it is using Kotlin 1.7.10 for its own purposes.

Next, create a project folder anywhere on your system. For our example project, we call it `HelloWorld`. Change into that folder:

```
cd /path/to/HelloWorld       (Linux)
chdir C:\path\to\HelloWorld  (Windows)
```

In order to initialize the Gradle project, enter (one line)

```
gradle init --dsl groovy --incubating
    --insecure-protocol ALLOW --package book.kotlinfx
    --project-name kotlinfx --test-framework kotlintest
    --type kotlin-application
```

You can also enter just `gradle init`, but then you will subsequently be asked for project coordinates inside the terminal.

The "init" task creates a simple scaffold project which consists of a main project described by file `settings.gradle` and a subproject called "app" in the accordingly named subfolder. The application can be run by just entering either of

```
gradle app:run
gradle run
```

The second variant is possible, because there is just one subproject. By the way, you can list all possible tasks via `gradle tasks` or `gradle tasks --all`, and entering `gradle help` shows more info.

Did you notice that two executable files `gradlew` and `gradlew.bat` and a folder `gradle` were created? This is the *Gradle Wrapper*, and it is a Gradle installation on its own, and you can henceforth use it to build the project. Just use `gradlew` from the wrapper instead of `gradle` from the Gradle distribution. You can even delete the main Gradle installation folder at this time, if you like.

It is now time to add JavaFX to the project. In Gradle, the `build.gradle` file is the main configuration file for the build process. You can find it inside the app subproject inside the `app` folder. Open the file inside a text editor, and inside the `plugins { . . . }` section, add

```
plugins {
  ...
  id 'org.openjfx.javafxplugin' version '0.0.13'
}
```

This plugin adds almost all that is necessary to add JavaFX to a Java or Kotlin project. Kotlin capabilities were already added during `gradle init`. We however still need to make sure that Kotlin compiles for JDK 17 and that JavaFX uses version 19 and allows for using the modules "javafx.controls" and "javafx.graphics". For that aim, add at the end of `build.gradle`

```
compileKotlin {
  kotlinOptions {
    suppressWarnings = true
    jvmTarget = "17"
  }
}
javafx {
  version = "19"
  modules("javafx.controls", "javafx.graphics")
}
```

Note JavaFX is separated into different modules. The modules "javafx.base", "javafx.controls", and "javafx.graphics" are essential to almost any JavaFX application. Because both the controls and the graphics module require the base module, the latter gets implicitly included in any build and can be omitted from the modules list. For more details, see https://openjfx.io/javadoc/19/

In the next section, we code our little "Hello World" JavaFX with Kotlin application.

A HelloWorld Project

The scaffold project built via `gradle init` just prints "Hello World!" on the console if run. As a starter JavaFX project, we instead want to show a little window with a button on it reacting to press events. To do so, replace the contents of

`app/src/main/kotlin/book/kotlinfx/App.kt`

by

```
package book.kotlinfx

import javafx.application.Application
import javafx.event.ActionEvent
import javafx.event.EventHandler
import javafx.scene.Scene
import javafx.scene.control.Button
import javafx.scene.layout.StackPane
import javafx.stage.Stage

fun main(args:Array<String>) {
  Application.launch(HelloWorld::class.java, *args)
}

class HelloWorld : Application() {
  override
  fun start(primaryStage:Stage) {
    primaryStage.title = "Hello World!"
    val btn = Button().apply {
        text = "Say 'Hello World'"
        setOnAction { evnt ->
            println("Hello World!")
        }
    }

    val root = StackPane().apply {
        children.add(btn)
    }

    with(primaryStage){
        scene = Scene(root, 300.0, 250.0)
        show()
    }
  }
}
```

Save the file. To now run the application, enter

```
./gradlew run     (Linux)
gradlew run       (Windows)
```

See Figure 1-1.

To first compile and build the project is not necessary—Gradle takes care of that if needed.

Setting Up for Eclipse

Note You can skip this section if you don't use Eclipse.

Download and install a recent Eclipse IDE from www.eclipse.org/downloads/. Start Eclipse and then, at Window → Preferences → Java → Installed JREs, register a JDK version 17 and make it the default. See Figure 1-2.

Then, at File → New → Project. . . → Gradle → Gradle Project, create a new Gradle project. Once asked, enter "kotlinfx" as the project's name; see Figure 1-3.

Keep everything else at its defaults. You end up with a main and a subproject; see Figure 1-4.

The name of the subproject reads "lib." We want to change it to a more meaningful variant.

Figure 1-1 JavaFX HelloWorld Running

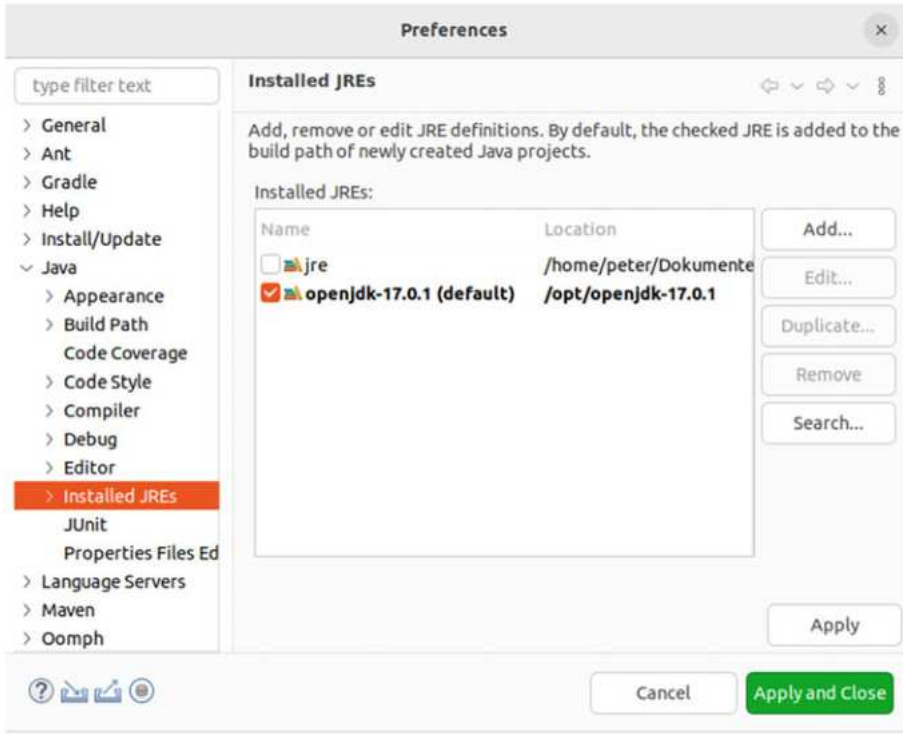

Figure 1-2 Eclipse JRE Setting

Caution Due to a design issue inside the Gradle-Plugin for Eclipse 2022-12, you cannot rename the subproject's name via Mouse-Right → Refactor → Rename... We must apply a workaround.

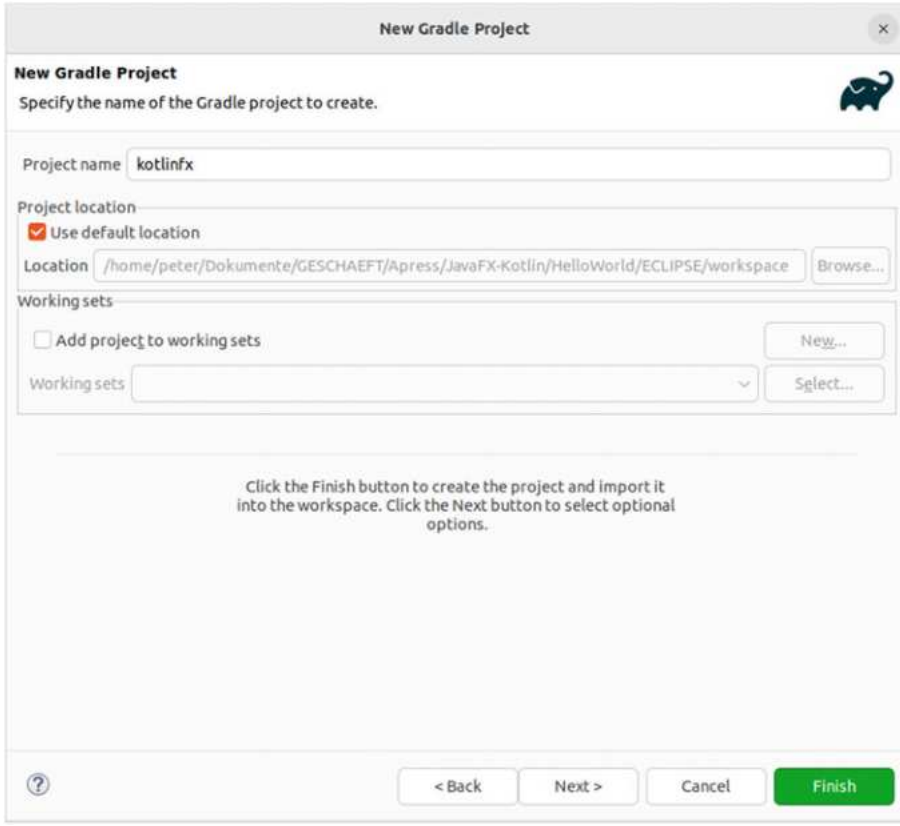

Figure 1-3 Eclipse Gradle Project Wizard

First, edit file `settings.gradle`. Change the line

```
include('lib')
->
include('HelloWorld')
```

Now delete the "lib" subproject from Eclipse. Make sure the "Also delete project contents" checkbox is *not* checked.

In your system's file explorer, rename folder `lib` inside `WORKSPACE/kotlinfx` to `HelloWorld`.

On the main project, invoke Mouse-Right → Configure → Configure and Detect Nested Projects... Press the "Finish" button. Ignore possibly shown errors.

Just to be on the safe side, restart Eclipse. The package view should now be as shown in Figure 1-5.

Back to the application, replace the contents of the `build.gradle` file by

```
plugins {
    id 'org.jetbrains.kotlin.jvm' version '1.7.10'
    id 'application'
    id 'org.openjfx.javafxplugin' version '0.0.13'
}

repositories {
    mavenCentral()
```

```
}

dependencies {
}

application {
    mainClass = 'book.kotlinfx.AppKt'
}

compileKotlin {
    kotlinOptions {
        suppressWarnings = true
        jvmTarget = "17"
    }
}

javafx {
    version = "19"
    modules("javafx.controls", "javafx.graphics")
}
```

After changes to file `build.gradle`, the project regularly needs to be updated: on "kotlinfx," press Mouse-Right → Gradle → Refresh Gradle Project. Also, remove the packages inside `src/test/java`; we don't need them for now.

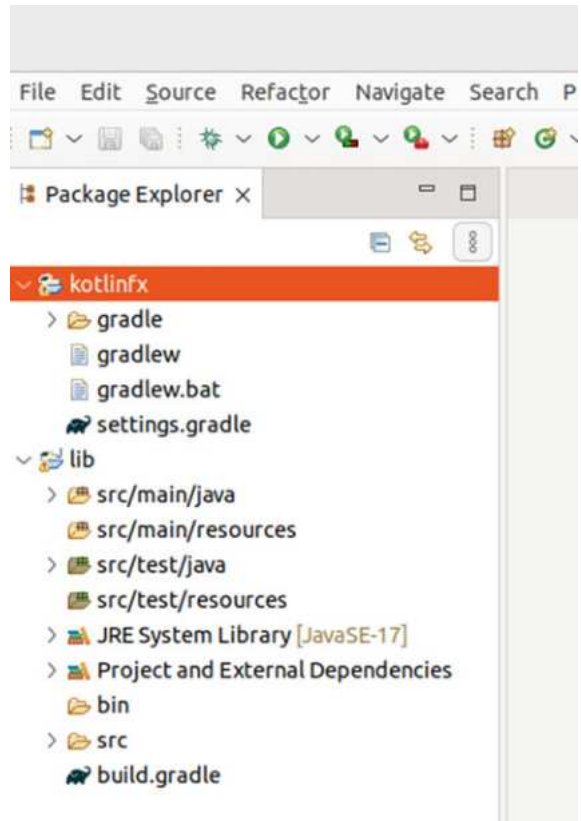

Figure 1-4 Eclipse Gradle Project