Ralph Steyer

Einführung in JavaFX/ OpenJFX

Moderne GUIs für RIAs und Java-Applikationen

2. Auflage

Springer Vieweg

Einführung in JavaFX/OpenJFX

Ralph Steyer

Einführung in JavaFX/OpenJFX

Moderne GUIs für RIAs und Java-Applikationen

2. Aufl. 2021

Springer Vieweg

Ralph Steyer
RJS EDV-KnowHow
Bodenheim, Deutschland

ISBN 978-3-658-35538-8 ISBN 978-3-658-35539-5 (eBook)
https://doi.org/10.1007/978-3-658-35539-5

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbiblio-grafie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Ursprünglich erschienen unter dem Titel: Einführung in JavaFX

Planung/Lektorat: David Imgrund
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Vorwort

Willkommen in der Welt von JavaFX und Java, in der auf erstaunlich einfache und konsistente Weise unterschiedliche Plattformen für lokale Computer und mobile Endgeräte in Netzwerken bis hin zum Internet verschmelzen. JavaFX bezeichnet dabei ein Framework zur Erstellung von modernen, plattformübergreifenden Java-Applikationen, das ursprünglich von Oracle bereitgestellt wurde und mittlerweile als Open-Source-Projekt unter dem Bezeichner OpenJFX geführt wird. Mit JavaFX wird der bereits seit Jahren etablierte Zugang zur Java-Welt sowohl fortgeführt als auch vereinfacht. Denn die JavaFX-Technologie tritt nicht zuletzt mit dem Ziel an, das professionelle Erstellen und Verteilen von interaktiven, multimedialen Inhalten und grafischen Oberflächen über sämtliche Java-Plattformen hinweg zu erleichtern. JavaFX stellt dazu eine Reihe an neuen Schnittstellen bereit, über die erfahrene Java-Entwickler erst einmal wie gewohnt die bereits vorhandenen Java-Bibliotheken auf einfache Weise verwenden können. Desgleichen gibt es auch explizite Erweiterungen, die etwa Java Swing bei der Erstellung von Oberflächen ablösen sollen und die Nutzung von Multimediatechniken erleichtern bzw. erweitern. JavaFX bietet aber auch die Möglichkeit, eine grafische Oberfläche ohne Java-Kenntnisse zu erstellen. Das soll aber in nicht bedeuten, dass Java-Kenntnisse nicht von Vorteil wären, wenn man JavaFX effektiv einsetzen und voll funktionale Anwendungen entwickeln möchte.

JavaFX gibt es schon seit einigen Jahren, aber ab der Version 2 hat JavaFX fast nur noch den Namen mit den älteren Varianten gemeinsam. Die folgenden Versionen sind weitgehend konsistent bzw. abwärtskompatibel und sind sowohl die Gegenwart als auch Zukunft der Entwicklung von grafischen Java-Oberflächen.

An wen wendet sich nun das Buch beziehungsweise wer sollte sich für diese Technologie interessieren? Zum einen ist die Zielgruppe neutral als IT-Fachleute mit Neugier auf diese GUI-Technologie im Java-Umfeld zu benennen. Insbesondere sind dies die Webseitenersteller, Webdesigner beziehungsweise Designer und Webprogrammierer – Sun bzw. Oracle sprach hier am Anfang schon von den *creative minds* –, die mithilfe einer Java-Technik multimedial aktive Web-Applikationen erstellen wollen oder an einem anderen Java-Projekt beteiligt sind, ohne direkt mit Java arbeiten zu wollen oder zu können. Diese werden auf Basis klassischer Web-Technologien wie HTML, XML, JavaScript und CSS

mit JavaFX schnell und einfach produktiv. Zum anderen fallen (zukünftige) Programmierer für Desktop-Applikationen und mobile Apps in die Zielgruppe für dieses Buch, wenn sie einen Weg suchen, auf einfache und einheitliche Weise mächtige Java-Techniken zur Erstellung von grafischen Oberflächen zu nutzen, ohne Java direkt anwenden zu müssen und/oder zu tief in Java einsteigen zu wollen. Natürlich dürften auch Java-Programmierer an JavaFX interessiert sein, nicht zuletzt weil JavaFX konzeptionell Swing ablösen sollte bzw. Swing selbst um diverse neue Komponenten erweitert.

Die Zielgruppe wurde zwar mehrfach mit „Programmierer“ bezeichnet, dennoch handelt es sich ausdrücklich um ein Einsteigerbuch – zumindest in Hinsicht auf die Programmierung. Sie müssen für den Umgang mit diesem Buch kein Programmierprofi sein. Kenntnisse in Java werden ebenfalls nicht vorausgesetzt, obgleich sie sehr hilfreich sein können – insbesondere bei den komplexeren Themen im hinteren Teil des Buches. Auf die notwendigen Grundlagen wird im Rahmen des Buchs eingegangen, unter anderem in einem kleinen Java-Crashkurs im Anhang. Grundlegende Erfahrung in einer (beliebigen) Programmier- oder Skriptsprache ist dennoch Voraussetzung, genauso wie Grundlagen im Umgang mit dem Internet. Ebenso werde ich nicht weiter auf elementare Details zu HTML, CSS oder JavaScript eingehen.

Nun habe ich kurz beschrieben, wie ich mir die Leser dieses Buchs vorstelle. Aber was habe ich als Autor mit JavaFX zu tun? Ich stelle mich Ihnen am besten kurz vor. Ich bin Diplom-Mathematiker. Seit 1996 arbeite ich als Fachautor, Fachjournalist, Dozent und Programmierer rund um die Themen Computer, (Java-)Programmierung und Internet. Davor stehen einige Jahre als Programmierer bei einer großen Versicherung im Rhein-Main-Gebiet in meinem Lebenslauf. JavaFX verfolge ich bereits seit den ersten öffentlichen Vorversionen.

Alles in allem finde ich JavaFX spannend, denn die Technologie öffnet im Bereich der Oberflächenprogrammierung und in Hinsicht auf multimediale Effekte in der Tat neue Welten. Die Arbeit mit JavaFX macht auf jeden Fall viel Spaß. Und solchen Spaß als auch Erfolg wünsche ich Ihnen.

www.rjs.de

Januar 2022 Ralph Steyer

Inhaltsverzeichnis

Einleitung & Grundlagen

1

Ohne Grundlagen geht es nicht

1.1 Was behandeln wir in dem einleitenden Kapitel?

Im einleitenden Kapitel wollen wir die zentralen Grundlagen zum Umfeld behandeln, in dem man mit JavaFX programmiert. Das beginnt bei einem allgemeinen Blick in die Welt von Java und JavaFX mit einer kompakten Vorstellung von Java und insbesondere einer Übersicht zu JavaFX, seiner Historie, der aktuellen Architektur und den Zielen. Ebenso besprechen wir in diesem Paragrafen, was Sie als Voraussetzungen für den Umgang mit JavaFX benötigen. Das umfasst die Vorkenntnisse, die Hardware und das Betriebssystem sowie die Java- und JavaFX-Basisumgebung samt erweiterter Tools wie Ant, Maven, Gradle, NetBeans und dem Scene Builder. Desgleichen betrachten wir die Installationen der Tools, denn gerade mit NetBeans und dem Scene Builder sowie selbstverständlich der Java-Basis wollen wir im Buch intensiv arbeiten.

Ebenso erstellen wir in dem Kapitel bereits die ersten JavaFX-Applikationen, und zwar auf mehrere Weisen: sowohl mit NetBeans auf Basis von Java und reinem JavaFX, aber auch mit dem Scene Builder sowie FXML. In dem Buch möchte ich – wie schon erwähnt – überwiegend mit NetBeans als Referenz-IDE (IDE – Integrated Development Environment) arbeiten. Man kann aber natürlich auch andere Entwicklungsumgebungen und Build-Tools verwenden. So wird in diesem Kapitel ebenso erklärt, wie man etwa mit Eclipse eine JavaFX-Applikation erstellt oder Build-Tools wie Ant, Maven und Gradle einsetzt. Die Konfiguration eines JavaFX-Projekts spielt dabei eine zentrale Rolle.

Und es gibt zu Beginn noch mehr zu tun. Zwar möchte ich Ihren Einstieg in JavaFX nicht mit zu viel Theorie belasten, aber einige grundlegende Details zu JavaFX-Applikationen benötigen wir unbedingt schon am Anfang. Sonst wird der eigentliche Umgang mit JavaFX in der Folge nicht auf soliden Füßen stehen. Solche ersten grundlegenden Informationen und Hintergründe zu JavaFX-Applikationen erhalten Sie

R. Steyer, *Einführung in JavaFX/OpenJFX,*
https://doi.org/10.1007/978-3-658-35539-5_1

ebenfalls in diesem Kapitel, und zwar gekoppelt mit den ersten Beispielen, damit Sie die Informationen direkt im praktischen Zusammenhang sehen.

Das Deployment beschließt das Kapitel. Das bedeutet, dass das grundsätzliche Verteilen beziehungsweise Aufliefern von JavaFX-Applikationen behandelt wird. Das umfasst im Wesentlichen das Erzeugen von ausführbaren **JAR-Dateien** (JAR – Java Archive). Die lange favorisierte Technik des **Java Web Starts** wird nicht mehr detailliert angesprochen, denn diese wird in der Praxis so gut wie gar nicht mehr unterstützt und muss als gescheitert angesehen werden.

Im Buch wird an einigen Stellen davon ausgegangen, dass Sie über (geringe) Grundlagen in Java und objektorientierter Programmierung (OOP) verfügen. Diese Grundkenntnisse sind immer dann notwendig, wenn wir mit Java-Quellcode in Berührung kommen. Im letzten Kapitel finden Sie einen Java-Crashkurs, der zwar kein vollständiges Java-Buch beziehungsweise eine allumfassende Einführung in Java sein kann und soll, aber dennoch die wichtigsten Grundlagen zu Java vermittelt.

1.2 Die Welt von Java und JavaFX

Machen wir uns nun auf den Weg in die Welt von JavaFX. JavaFX ist offensichtlich ein Teil der Java-Plattform, die bereits seit vielen Jahren etabliert ist. Wer JavaFX verstehen will oder sich allgemein damit beschäftigt, sollte zumindest Java selbst in Grundzügen kennen. Deshalb werfen wir zunächst einen kleinen Blick auf dessen Struktur und auf die Geschichte von Java im Gesamten.

1.2.1 Was ist Java?

Java ist einerseits eine **Programmiersprache,** bezeichnet andererseits aber auch eine ganze **Plattform** zur Ausführung von stabilen, sicheren und leistungsfähigen Programmen unabhängig vom zugrunde liegenden Betriebssystem. Verallgemeinernd spricht man von der **Java Technology**. Darunter versteht man eine ganze Sammlung von Spezifikationen, die einerseits eine Programmiersprache und andererseits verschiedene Laufzeitumgebungen samt umfangreichen Bibliotheken für Computerprogramme definieren. Die gesamte Spezifikation der Java-Technologie umfasst folgende Bestandteile:

- Die eigentliche Programmiersprache Java,
- die Java-Plattform – eine standardisierte Software-Plattform,
- das grundlegende Entwicklungswerkzeug (Java Development Kit – JDK) samt darauf aufbauender Erweiterungen und
- die Java-Laufzeitumgebung JRE (Java Runtime Environment), um Java-Programme ausführen zu können.

Der Java-Erfinder Sun hat schon zu Beginn Java als Sprache so charakterisiert:

Java
eine einfache, objektorientierte, dezentrale, interpretierte, stabil laufende, sichere, architekturneutrale, portierbare und dynamische Sprache, die Hochgeschwindigkeits-Anwendungen und Multithreading unterstützt.

In dieser (technischen) Marketingaussage stecken bereits die wichtigsten Informationen, was genau Java als Sprache, aber auch als Plattform auszeichnet. Etwa die Tatsache, dass Java eine objektorientierte Sprache ist, mit der Anwendungen erstellt werden können, die auf verschiedenen Systemen mit unterschiedlichen Prozessoren und Betriebssystemarchitekturen lauffähig sind. Die fertigen Programme können auf jedem System ausgeführt werden, das die JRE samt der so genannten virtuellen Maschine (Java Virtual Machine beziehungsweise JVM oder kurz VM – ein virtueller Prozessor) von Java implementiert.

Doch wie ist das möglich? Java gilt als interpretiert und kompiliert zur gleichen Zeit, was im Grunde einen Widerspruch darstellt. Beide Vorgänge (Interpretation und Kompilierung) beschreiben den Vorgang der Übersetzung von einem Quelltext in einen lauffähigen Binärcode, der von einem Computer ausgeführt werden kann. Dies kann man auf zwei Arten machen. Entweder, der Quelltext wird auf einen Schlag mit einem geeigneten Programm übersetzt und dann dieser daraus resultierende Binärcode auf den Computer zum Laufen gebracht. Das bedeutet dann, dass der Quelltext kompiliert wurde. Man kann aber auch mit einem anderen Programmtyp den Quelltext laden und Zeile für Zeile lesen und direkt zur Laufzeit des Programms übersetzen lassen. Das ist dann der Vorgang der Interpretation. Java macht beides.

Der eigentliche Quellcode wird in einen binären Zwischencode (sogenannten **Bytecode**) kompiliert, der ein architekturneutrales und noch nicht vollständiges Object-Code-Format ist. Er ist noch nicht lauffähig (also noch nicht zu Ende übersetzt) und muss von einer **Laufzeitumgebung** interpretiert[1] werden. Dies ist die oben bereits genannte JRE, deren wesentlicher Bestandteil die JVM ist. Da jede Java-Laufzeitumgebung plattformspezifisch ist, arbeitet das endgültige ausgeführte Programm auf dieser spezifischen Plattform. Dort werden alle Elemente hinzugebunden, die für eine spezielle Plattform notwendig sind. Die Tatsache, dass der letzte Teil der Übersetzung des Bytecodes von einem plattformspezifischen Programm auf der Plattform des Endanwenders ausgeführt wird, nimmt dem Entwickler die Verantwortung, verschiedene Programme für verschiedene Plattformen erstellen zu müssen. Die Interpretation erlaubt zudem, Daten zur Laufzeit zu laden, was eine wichtige Grundlage für das dynamische Verhalten von Java ist.

[1] Dann erst entsteht der plattformspezifische Maschinencode.

1.2.2 Etwas zur Historie von Java

Hinter Java stand ursprünglich die amerikanische Firma Sun Microsystems, wobei diese 2010 von Oracle (https://www.oracle.com/) übernommen wurde. Die Geschichte von Java selbst geht bis ins Jahr 1990 zurück. Zu diesem Zeitpunkt versuchte Sun, im Rahmen eines Projekts mit dem Namen **Green** den zukünftigen Bedarf an EDV zu analysieren, um einen zukunftsträchtigen Markt zu lokalisieren. Der Konsumentenbereich der allgemeinen Elektronik (Telefone, Videorekorder, Waschmaschinen, Kaffeemaschinen und eigentlich alle elektrischen Maschinen, die Daten benötigen) wurde als der (!) Zukunftsmarkt der EDV prognostiziert – ein extrem heterogenes Umfeld mit den unterschiedlichsten Prozessoren beziehungsweise grundverschiedenen Kombinationen von Hard- und Software-Komponenten. Dafür eine gemeinsame Plattform zu schaffen und damit frühzeitig einen Standard festzulegen, wurde von Sun als die Zukunftschance der EDV schlechthin vorausgesagt.

Wichtigste Forderungen an eine solche auf allen denkbaren Systemen lauffähige Plattform waren eine erhebliche Fehlertoleranz, eine leichte Bedienbarkeit und eine bedeutend bessere Stabilität als diejenige, die bei allen bis dahin vorhandenen Plattformen existierte. Die Plattform musste deshalb ein neues Betriebssystem, oder zumindest eine neue Betriebssystemergänzung, für alle populären Betriebssysteme bereitstellen. Ebenso sollte möglichst eine neue Programmiersprache entwickelt werden, denn alle bis dahin vorhandenen Programmiersprachen wiesen zu große Schwächen hinsichtlich der Stabilität auf. Gerade bei Bedienungsfehlern waren damalige Techniken und Programme einfach zu intolerant.

Ab dem Frühjahr 1991 gingen die Planungen in die Generierung eines Prototyps für eine solche universale Plattform über, der **Oak** (Eiche) genannt wurde. Aber erst 1995 präsentierte Sun das unbenannte sowie für das Internet aufbereitete und optimierte Java auf Basis der Java-Applets. Dazu wurde das zugehörige kostenlose und frei zu verwendende Paket von Entwicklungs-Tools (**JDK** – Java Development Kit) in der Version 1.0 vorgestellt.

Natürlich wurde Java dann über die Jahre erheblich weiterentwickelt und erscheint derzeit in einem halbjährlichen Aktualisierungszyklus.

1.2.3 Wo findet Java heutzutage Anwendung?

Wie erwähnt, waren Applets am Anfang der (Erfolgs-)Geschichte von Java die ersten und auch sehr populären Java-Anwendungen. Aber heutzutage findet man im Web kaum noch Seiten, die Java-Applets einsetzen. Dafür gibt es diverse Gründe:

- Schlechte Performance und Ressourcenhunger der ersten Applets beziehungsweise von Java selbst.

- Relativ schwache Hardware bei Anwendern in den prägenden Jahren von Java im Web (90er-Jahre).
- Zu langsame Internet-Verbindungen bei Privatanwendern – ebenfalls in den entscheidenden Jahren.
- Permanente Konflikte zwischen Sun und Microsoft samt der mangelhaften Unterstützung von Java im Internet Explorer in den 90iger-Jahren.
- Allgemeiner Rückgang der clientseitigen Web-Programmierung um das Jahr 2000.
- Zwingend notwendige Installation einer passenden JVM bei einem Anwender.

Doch Java hat neue Einsatzgebiete gefunden und sich dort etabliert. Java kommt heutzutage zum Beispiel in folgenden Bereichen zum Einsatz:

- Erstellung von plattformunabhängigen Desktopanwendungen.
- Serverseitige Dienste im Internet beziehungsweise Intranet, etwa in Form eines Java Application Servers (z. B. Apache Tomcat, JBoss, GlassFish, usw.) oder rein von der Programmierseite mit Hilfe von Java-Servlets, Java Server Pages (JSP), Java Server Faces (JSF) und spezieller Web-Frameworks.
- Anwendungen auf Chipkarten – sogenannten Java Cards – und zahlreichen anderen eingebetteten Systemen, für die Java ja auch ganz am Anfang entwickelt wurde.
- Unter Android werden Apps für Smartphones und Tablets oft in Java geschrieben.
- Die Clientplattform im Web ist immer noch ein (theoretisches) Einsatzgebiet von Java. Auch wenn Applets wie erwähnt nahezu verschwunden sind. Gerade mit JavaFX sollte Java die Basis von RIAs (Rich Internet Applications) werden. Das kann man auch wirklich noch machen, hat sich jedoch nicht wirklich durchgesetzt.

1.2.4 Die Java-Plattform

1.2.4.1 Build-Tools & Co

Bis vor einigen Jahren hat man Java-Applikationen meist „von Hand“ konfiguriert und kompiliert, aber durch immer komplexere Abhängigkeiten und die Verwendung diverser Ressourcen ist der Aufwand immer größer geworden. Konventionen und sogenannte Build-Tools samt zentraler Quellen erleichtern die Arbeit erheblich. Dabei haben sich im Java-Umfeld mehrere Tools durchgesetzt, auf die wir im Buch auch teilweise zurückgreifen. Drei wichtige Vertreter, die in allen modernen IDEs für Java integriert sind, sollen hier kurz vorgestellt werden.

1.2.4.2 Ant

Apache **Ant** (englisch für Ameise, aber eigentlich ein Apronym für „Another Neat Tool“ – https://ant.apache.org/) ist Open Source, selbst in Java geschrieben und dient allgemein

zum automatisierten Erzeugen von ausführbaren Computerprogrammen aus Quelltexten und zusätzlichen Ressourcen. Ant orientiert sich an dem gerade in Linux sehr verbreiteten Programm **make**. Bei der automatisierten Erstellung werden Quelltexte, Bibliotheken und sonstigen Dateien verbunden.

Gesteuert wird Ant durch eine XML-Datei, die sogenannte **Build-Datei.** Sie heißt bei Ant standardmäßig *build.xml*. In der Build-Datei wird ein *project*–Element (deutsch „Projekt“) definiert, welches das Wurzelelement der XML-Datei bildet. Das Ant-Project selbst enthält Targets (deutsch „Ziele“), die gezielt über die Kommandozeile oder die Entwicklungsumgebung aufgerufen werden können. Ein Target selbst besteht wiederum aus Aufrufen von Tasks (deutsch „Aufgaben“). Wenn bei einem Target oder zwischen Targets Abhängigkeiten existierten, löst Ant diese Abhängigkeiten bei Bedarf automatisch auf und arbeitet die Targets entsprechend ab.

1.2.4.3 Maven

Apache **Maven** (https://maven.apache.org/) ist ein weiteres, auf Java basierendes Build-Management-Tool. Maven versucht konsequent „Konvention vor Konfiguration“ (englisch Convention over Configuration) für den gesamten Zyklus der Softwareerstellung abzubilden. Dabei sollen möglichst viele Schritte automatisiert werden. Wenn man die von Maven vorgegebenen Standards einhält, braucht man für die meisten Aufgaben des Build-Managements nur sehr wenige Konfigurationseinstellungen vorzunehmen.

So genannte Maven-**Archetypes**, die oft auch im Internet oder IDEs bereitgestellt werden, sind dabei eine Art Gerüst für unterschiedlichste Aufgaben bei Softwareprojekten samt einem ausgefeiltem Abhängigkeitsmanagement.

Wenn man Maven als Projektmanagement- Tool sieht, gibt es ein Projektobjektmodell (Project Object Model oder kurz POM), das über eine *pom.xml*-Datei abgebildet wird. Wie die Dateierweiterung deutlich macht, basiert diese auf XML. Damit können eine Reihe von Standards eingehalten werden und man kann Projektlebenszyklen verwalten, definieren, überwachen. Darüber hinaus hat man ein Managementsystem für Abhängigkeiten sowie eine Logik zur Ausführung von sogenannten Plug-in-Zielen (Goals) in den verschiedenen Lebenszyklusphasen wie etwa dem Testen oder dem Erstellen. Wenn man sich an die Regeln von Maven hält, wird man als Programmierer über best practices geführt und Projekte folgen einer konsistenten Struktur. Diese konsistente Struktur sorgt dafür, dass Projekte damit IDE-unabhängig werden, was die Portierbarkeit natürlich erleichtert. Vor allen Dingen müssen Sie eben keine IDE benutzen, um Projekte zu übersetzen.

Im Umfeld von Maven sind Repositories von zentraler Bedeutung. Ein Repository ist eigentlich nur ein verwaltetes Verzeichnis zur Speicherung und Beschreibung von digitalen Objekten. Also eine Art digitales Archiv mit der Möglichkeit zur Versionierung und ähnlichen Dingen. Das erlaubt es, gezielt bestimmte Ressourcen für Anwender zur Verfügung zu stellen.

Bei Maven gibt einmal das lokale Repository, aber auch ein zentrales, öffentliches Repository, in dem Open-Source-Komponenten zur Verfügung gestellt werden, die Sie in einem Maven-Projekt benutzen können.

Ein Großteil des Charmes von Maven basiert darauf, dass Sie eben diese Komponenten zur Verfügung haben, und diese werden auch erst auf Ihren lokalen Rechner übertragen, wenn Sie sie wirklich benötigen. Das heißt, sie werden irgendwo in einem Maven-Projekt angegeben und dann bei Bedarf automatisch in Ihr lokales Repository nachgeladen.

1.2.4.4 Gradle

Auch **Gradle** (https://gradle.org/) basiert auf Java und fungiert als Build-Management-Automatisierungs-Tool. Gradle nutzt eine auf Groovy basierende domänenspezifische Sprache (DSL) zur Beschreibung der zu erstellenden Projekte. Gradle-Skripte sind damit direkt ausführbarer Code und keine XML-Anweisungen, wie bei Ant oder Maven.

Gradles Build-Konzept verwendet Standardkonventionen („convention over configuration“) für das Verzeichnislayout der Projektquellen (ursprünglich von Maven eingeführt) sowie die üblichen Phasen für den Bau eines (Java-)Projekts (Validieren, Kompilieren, Testausführung, Archiv-Erstellung und Report-Generierung, Verteilung). Die Build-Datei kann daher sehr klein ausfallen, wenn man sich an die Defaulteinstellungen hält – vor allen Dingen bei einfachen Java-Projekten. Gradle übernimmt auch weitgehend das Maven-Konzept des Managements der Abhängigkeiten eines Projekts von anderen Projekten oder Fremdbibliotheken und kann deshalb die weitverbreiteten Maven-Repositories verwenden. Gradle besteht aus einem abstrakten Kern und einer Vielzahl von Plug-ins und ist durch diese Struktur vielfältig erweiterbar.

Bei der Buildverarbeitung unterscheidet Gradle zwei Hauptphasen, die immer durchlaufen werden: Konfiguration und Ausführung. Während der Konfiguration wird die gesamte Build-Definition durchlaufen, um Abhängigkeiten und Reihenfolge aller abzuarbeitenden Schritte festzulegen Im zweiten Teil werden diese Schritte dann durchlaufen.

Gradle nutzt für einen einfachen Build hauptsächlich drei benutzerdefinierte Dateien. Verbindlich ist die Datei *build.gradle* mit allen Tasks und Abhängigkeiten eines Projekts sowie die optionalen Dateien *settings.gradle* (bei einem Multiprojekt werden hier die Unterprojekte festgelegt) und *gradle.properties* (projektspezifische Gradle-Initialisierung eines Builds).

1.3 Was sind JavaFX sowie JavaFX Script und was hat es mit OpenJFX auf sich?

JavaFX ist eine Erweiterung von Java, die erstmals auf der JavaOne-Konferenz von Sun im Jahr 2007 vorgestellt wurde, und zwar als neue Scripting-Plattform für Web- und Desktop-Applikationen sowie mobile Anwendungen. Ende 2007 beziehungsweise Anfang 2008 erschienen erste offizielle Vorversionen und um den Jahreswechsel 2008 auf 2009 wurde

die erste Finalversion festgeschrieben. Eine erste große Referenzapplikation war eine RIA (Rich Internet Application – eine Webseite mit erweiterten interaktiven Möglichkeiten) zu den Olympischen Winterspielen in Vancouver, die es aber auch parallel auf Basis von reinen Webtechnologien gab, und die auch mittlerweile im Web nicht mehr gehostet wird.

Dabei stellte in den ersten Versionen eine integrierte Skriptsprache mit Namen **JavaFX Script** als zentraler Kern der gesamten Technologie die Mittel zur Verfügung, um allgemeine visuelle, hochleistungsfähige Anwendungen auf Basis von Java zu erzeugen. Obwohl auch ganz einfache Applikationen mit Konsolenausgabe erstellt werden können, wurde JavaFX Script von Anfang an speziell dafür designed, um den kreativen Prozess der Erstellung von Benutzeroberflächen zu optimieren. Wie Sun ausdrücklich betont hatte, wollte man mit JavaFX Script neben den klassischen Java-Programmierern vor allen Dingen die so genannten »creative minds« erreichen, die nicht unbedingt Java-Kenntnisse und möglicherweise sogar insgesamt wenig Programmiererfahrung haben. Dieser Ansatz mit JavaFX Script kann jedoch als gescheitert angesehen werden und die Skriptsprache wird in JavaFX ab der Version 2.0 nicht mehr zur Verfügung gestellt.

Quellcode, der mit JavaFX Script erzeugt wurde, kann in neuen Versionen von JavaFX nicht mehr verwendet werden! Insgesamt sind Quellcodes für JavaFX vor der Version 2.0 gar nicht oder nur mit sehr viel Mühe auf die neuen Versionen zu aktualisieren. Ab der Version 2.0 sind die Quellcodes hingegen auch in neueren JavaFX-Umgebungen (weitgehend) ohne große Anpassungen zu verwenden.

Insgesamt muss man zugeben, dass die ersten Versionen von JavaFX nicht den Erfolg hatten, den Sun und dann später Oracle wohl erwartet hatten. Nicht zuletzt deshalb wurde JavaFX für die Version 2.0 technisch vollkommen verändert. JavaFX bezeichnet ab der Version 2.0 ein Framework zur Erstellung von modernen, plattformübergreifenden Java-Applikationen beziehungsweise Rich Internet Applications, das eben mit den ersten Versionen fast nur noch den Namen gemeinsam hat. Mit der Version 2 hat JavaFX jedoch einen festen, modernen und ausgereiften Stand erreicht und ist eine Java-Kerntechnik bei der Entwicklung von grafischen Java-Oberflächen. JavaFX sollte von Anfang an Swing ablösen, denn es bietet mehr Möglichkeiten (neue Komponenten und Widgets) und ist gleichzeitig einfacher in der Anwendung.

Beachten Sie, dass die Version 2 von JavaFX direkt auf JavaFX 8 aktualisiert wurde und man deshalb nicht eine durchgängige Reihe an Versionsnummern vorliegen hat. JavaFX 8 brachte einige Neuerungen wie etwa die Unterstützung von Lambda-Ausdrücken und das JavaFX-UI-Toolkit. Allerdings gab es auch gewisse Einschränkungen, denn mit der Version 8u33 wurde der ARM-Support eingestellt.

Grundsätzlich muss man festhalten, dass die Zukunft von JavaFX nicht ganz klar ist. Das hängt einmal an den Interessen von Oracle, aber auch an den allgemeinen modernen Paradigmen „Mobile first“ und „Web first“, welche die aktuelle Programmierung von Oberflächen und Anwendungen allgemein bestimmen. Dennoch ist der Support für JavaFX wohl mindestens bis 2025 sicher. Zudem halten sich Techniken, deren offizielle Weiterentwicklung bzw. Support eingestellt wurden, erfahrungsgemäß noch viele Jahre.

1.3.1 Was ist OpenJFX?

Kommen wir nun zu dem Bezeichner „OpenJFX“, der im Grunde der neue Name von JavaFX ist.

JavaFX war bei seiner Entstehung eine Erweiterung von Java, die erst einmal nicht Bestandteil des Kern-APIs bzw. JDK war. Aber ab der Version 7 bzw. vor allen Dingen 8 wurde JavaFX dann als integraler Bestandteil des Kern-APIs von Java bzw. dem JDK gesehen (Abb. 1.5).

Mittlerweile wurde diese Integration jedoch wieder aufgehoben! Sie benötigen für JavaFX also zusätzliche Ressourcen – darauf wird im Buch natürlich noch eingegangen.

Man muss also festhalten, dass JavaFX immer wieder an anderer Stelle verschoben wurde und damit aber auch die Voraussetzungen für die Verwendung immer wieder anzupassen sind. Was auch Konsequenzen beim Erstellen von JavaFX-Applikationen hat. Vor allen Dingen hat Oracle mittlerweile die Verantwortung für die Entwicklung von JavaFX einer Open Source-Organisation übertragen, die unter dem Namen **OpenJFX** (Abb. 1.1) auftritt (https://openjfx.io/). Deshalb wird eben auch „OpenJFX“ als alternativer Bezeichner für JavaFX benutzt. Wobei auf den Webseiten von OpenJFX selbst konsequent weiter der Bezeichner „JavaFX“ verwendet wird. Dieser ist deutlich bekannter und vermeidet zudem die Verwechselung mit dem Bezeichner „OpenFX“, der nichts mit Java zu tun hat.

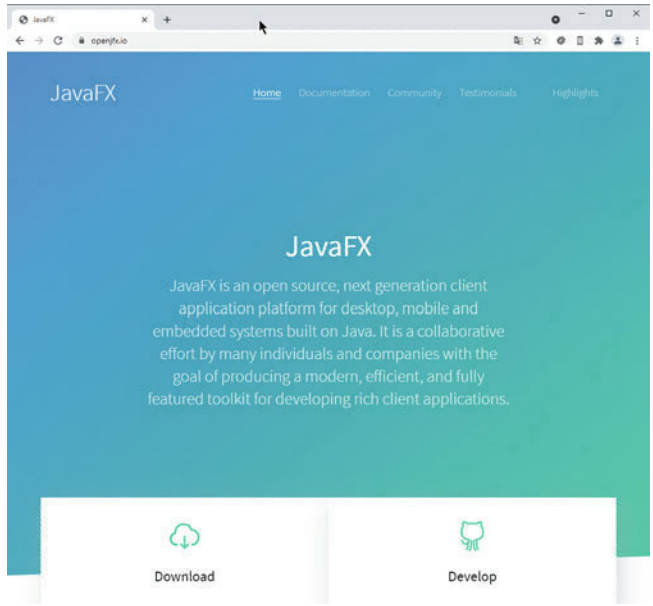

Abb. 1.1 Die zentrale Webseite zu OpenJFX bzw. JavaFX

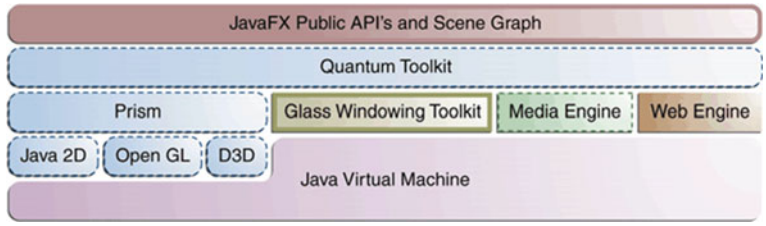

Abb. 1.2 Die Architektur, wie Sie Oracle offiziell für die Version 2 vorgestellt hatte und die immer noch Relevanz hat

Wir werden deshalb im Buch ebenfalls weitgehend bei dem Bezeichner „JavaFX“ bleiben, wenn nicht der Open Source-Aspekt oder eine andere Besonderheit im Fokus stehen soll.

Diese Entwicklung bei Java in Richtung Open Source findet man übrigens ebenso bei dem JDK selbst, das auch als freies OpenJDK (https://openjdk.java.net/) verfügbar ist. Mit OpenJFX und dem OpenJDK lässt sich ein komplett freies Java-System mit grafischer Oberfläche unter der GPL (https://www.gnu.org/licenses/#GPL) erstellen und im Wiki von OpenJDK gibt es eine eigene Kategorie für OpenJFX, was die Verzahnung deutlich macht (Abb. 1.3).

1.3.2 Die Architektur

Betrachten wir die Architektur von JavaFX etwas genauer.

- Bei der JavaFX Architektur (Abb. 1.2) befindet sich ganz unten – wie bei jeder Java-Applikation – die JVM. Darauf setzen diverse Erweiterungen auf, die man teils schon seit Jahren bei Java nutzen kann, und die auch JavaFX zur Verfügung stehen, wie etwa **Java 2D** zum Zeichnen von zweidimensionalen Formen.
- Mit dem **Glass Windowing Toolkit** bekommen Sie Zugriff auf native Betriebsleistungen wie die Fensterverwaltung, Timer oder Ereignisverwaltung. Dabei handelt es sich explizit um eine **plattformabhängige** Schicht.
- Bei **Prism** handelt es sich um eine Grafik-Pipeline[2], die auf Hardware- und Software-Renderern ausgeführt werden kann. Unter anderem werden damit auch Features von Java 2D und diverse grafische Effekte wie Schatten, Spiegelungen, Transformationen, Animationen, etc. auf einer hohen Ebene verfügbar gemacht.
- Das **Media**-Framework beziehungsweise die **Media Engine** basiert auf GStreamer und bietet – vereinfacht gesagt – sehr umfangreiche Unterstützung für Audio und Video.

[2] Gewisse andere Verwendungen in Hinsicht auf Datenspionage sind eine etwas unglückliche Mehrdeutigkeit.

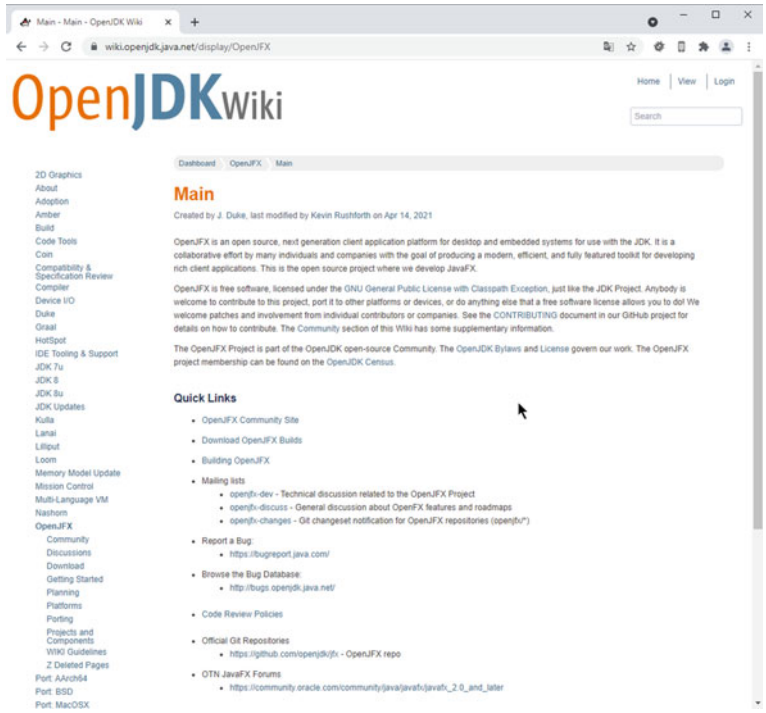

Abb. 1.3 Hinweise zum zur JavaFX-Entwicklung finden Sie auch im Wiki von OpenJDK

- Die **Web Engine** beziehungsweise **WebView**-Komponente erlaubt die Einbettung von Web-Inhalten in JavaFX-Applikationen. Das umfasst das Rendern von HTML auf Basis der Webkit-Engine[3] sowie das hardwareabhängige Rendern über Prism. Besonders interessant ist die Möglichkeit des DOM-Zugriffs (DOM – Document Object Model) und der Manipulation des DOM.
- Das **Quantum Toolkit** fügt nun Prism und das Glass Windowing Toolkit sowie die Engine für das Web und für Multimediaverarbeitung zusammen und macht diese einheitlich den JavaFX APIs zugänglich.

Insbesondere braucht man aber als Entwickler bei JavaFX im Grunde gar nicht wissen, wie die tiefliegenden Ebenen arbeiten. Das JavaFX API und das Quantum Toolkit kapseln diese Details und verbergen diese damit vor dem Anwender. Dabei gestatten die Java APIs für JavaFX eine End-to-end-Java-Entwicklung mit allen bekannten Java-Möglichkeiten bis hin zu Generics oder Multithreading. Neue UI-Komponenten lassen sich mit JavaFX schnell und einfach erstellen und alle UI-Komponenten lassen sich per

[3] Diese ist die Basis von Chrome oder Safari.

CSS gestalten. Ebenso gibt es diverse Enterprise-Möglichkeiten bei JavaFX wie die Bindung an Datenservices, Multitouch-Operationen, den Aufbau von Serverarchitekturen und die Kommunikation zwischen einem Webserver und dem Client (mit HTTP-GET, REST oder Webservices) oder der HTML5-Integration (inklusive einem Canvas-API).

1.3.3 JavaFX-Oberflächen ohne Java programmieren zu müssen

Obwohl JavaFX Script in neuen Versionen von JavaFX verschwunden ist, wurde die grundsätzliche Intention bei JavaFX, dass die Erstellung von grafischen Oberflächen für Java-Applikationen vereinfacht werden sollte[4] und insbesondere Java-Kenntnisse keine zwingende Voraussetzung mehr darstellen sollten, nicht aufgegeben. Ganz im Gegenteil: Gerade für Entwickler ohne vertiefende Java-Grundlagen stellt JavaFX nun statt JavaFX Script mit **FXML** eine einfach zu lernende, deklarative Sprache zur Verfügung, die eine alternative Definition von grafischen Oberflächen rein über XML erlaubt[5] und einen zentralen Teil des **JavaFX-UI-Toolkits** darstellt. Zudem können Sie dort auch Web-Technologien wie CSS (Cascading Style Sheets) oder JavaScript (aber auch andere Sprachen wie Groovy) einsetzen[6], die in den XML-Code[7] eingebettet oder damit verknüpft werden können. Das schafft ganz neuen Gruppen Zugang zur Java-Welt, die bisher ohne entsprechenden Java- und OO-Background ausgegrenzt waren.

Und dieser Ansatz geht mit einem GUI-Designer mit visuellen Controls zur Erstellung von Oberflächen einher – dem **Scene Builder** (https://gluonhq.com/products/scene-builder/). Dieser erlaubt sogar die Erstellung einer GUI per Drag & Drop beziehungsweise rein visuell.

Und wie bei den Architekturbetrachtungen oben gesehen, arbeitet JavaFX nahtlos mit reinem Java zusammen und fügt sich harmonisch in die gesamte Java-Architektur ein, was eine große Flexibilität zur Folge hat. Oder anders ausgedrückt: Egal welche Art von JavaFX-Applikationen Sie letztendlich erzeugen, JavaFX-Applikationen laufen wie alle Java-Applikationen auf sämtlichen Plattformen, die eine passende virtuelle Java-Maschine sowie eine passende JavaFX-Bibliothek bereitstellen. Auch für JavaFX-Applikationen gelten die Java-typischen Kriterien „write-once-run-anywhere“, das Sicherheitsmodel für Applikationen, die einheitliche Distribution und die Enterprise-Connectivity.

---

[4] Das offizielle Stichwort lautet: Fokus auf die Fähigkeiten anstelle der Technologie.

[5] So etwas kennt man beispielsweise aus der Programmierung unter dem.NET-Framework von Microsoft in Form von XAML (Extensible Application Markup Language) oder auch der Android-Programmierung.

[6] Was aber auch auf der Java-Ebene von JavaFX geht.

[7] Aber auch Java-Code.

1.4 Was benötigen Sie?

Schaffen wir uns jetzt einen Überblick über die Voraussetzungen, die bei Ihnen zur Arbeit mit JavaFX erfüllt sein müssen. Diese haben sich – wie angedeutet – über die letzten Versionen von Java bzw. JavaFX immer wieder geändert. Es werden vor allen Dingen in neuen Versionen des JDK bzw. JavaFX einige wichtiger Konfigurationsschritte notwendig, ohne die JavaFX-Applikationen nicht übersetzt bzw. ausgeführt werden können.

1.4.1 Die Hardware und das Betriebssystem

Sie benötigen selbstverständlich zu einer erfolgreichen Arbeit mit dem Buch und natürlich zur realen Programmierung erst einmal einen Computer (in der Regel wird das ein PC oder Apple sein), der eine ausreichende Leistungsstärke haben sollte. Moderne Rechner sollten allerdings die Voraussetzungen grundsätzlich erfüllen. Und Sie sollten auf jeden Fall einen (möglichst schnellen) Zugang zum Internet haben. Das betrachte ich als selbstverständlich.

Die Frage, welches Betriebssystem Sie haben sollten, ist nicht unwesentlich, aber sehr einfach zu beantworten. Sie benötigen ein Betriebssystem, für das es das JDK mit JavaFX-Unterstützung gibt. Unter dem Link https://wiki.openjdk.java.net/display/OpenJFX (Abb. 1.3) kommen Sie zu dem Wiki von OpenJDK und dem Bereich für OpenJFX mit allen relevanten Informationen samt Verlinkung zur eigentlichen Webseite von OpenJFX[8].

Wenn Sie auf dieser Seite über *Platforms* (oder direkt im Browser) den Link https://wiki.openjdk.java.net/display/OpenJFX/Platforms auswählen, erhalten Sie auf der folgenden Webseite eine Auflistung aller Betriebssysteme, mit denen Sie arbeiten können (Abb. 1.4). Beachten Sie jedoch, dass das Wiki nicht immer auf dem aktuellsten Stand ist.

Im Wesentlichen sind das verschiedene Versionen von Windows, Linux, MacOS X/macOS, iOS, Android und Embedded Linux, wobei die Unterstützung der letztgenannten Umgebung in neuen Versionen von JavaFX eingeschränkt ist. Wir konzentrieren uns im Buch auf Windows als Referenzsystem[9].

[8] Beachten Sie, dass sich der Aufbau der Webseite jederzeit ändern und diese Struktur nicht garantiert werden kann. Aber Oracle wird ziemlich sicher irgendwo auf der Webseite diese Information bereitstellen.

[9] Das spielt aber keine wichtige Rolle – das ist ja gerade das Wesen von Java.

Abb. 1.4 Die unterstützten Betriebssysteme

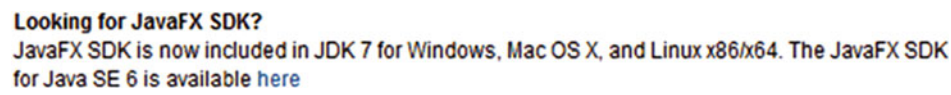

Abb. 1.5 JavaFX war integraler Bestand des JDK 7 und mehrerer weiterer Versionen

1.4.2 Die Java-Basisumgebung

Nun wird es spannend. Wir bewegen uns mit JavaFX natürlich im Java-Umfeld. Java-Applikationen benötigen eine spezielle Laufzeitumgebung, damit sie ausgeführt werden können. Und das bedeutet, dass Sie auf Ihrem Computer auf jeden Fall eine Java Laufzeitumgebung (JRE) benötigen. Genauso wie jeder Anwender, der eine Java- beziehungsweise JavaFX-Applikation ausführen möchte.

Eine solche Java-Laufzeitumgebung gibt es mittlerweile für fast jedes moderne PC-Betriebssystem und auch für die meisten mobilen Geräte wie Handys oder andere technische Geräte des täglichen Lebens. Bei vielen Betriebssystemen wird sie sogar bereits in der Grundausstattung zur Verfügung gestellt.

Möglicherweise haben Sie aber auf Ihrem Entwicklerrechner nun nicht die passende Version installiert. Und auch aus anderen Gründen müssen wir auf die JRE und die genaue Version noch etwas detaillierter eingehen.

Wir wollen ja nicht nur Java- beziehungsweise JavaFX-Applikationen ausführen, sondern selbstverständlich ebenso erstellen. Und dazu benötigen wir mehr als nur eine Laufzeitumgebung.

Wir brauchen entsprechende Entwicklungstools für Java respektive JavaFX. Im Kern bedeutet das, dass wir erst einmal minimal ein aktuelles Java Entwicklungspaket (das Java SE Development Kit oder nur Java Development Kit – abgekürzt JDK) benötigen. Und das müssen Sie sich erst einmal besorgen. Entweder direkt von Oracle über den URL https://www.oracle.com/java/technologies/java-se-glance.html bzw. http://jdk.java.net oder das OpenJDK-Projekt über http://openjdk.java.net/.

Beachten Sie, dass in der derzeit aktuellen Version des JDK JavaFX-Unterstützung **kein** integraler Bestandteil mehr ist. Das war – wie schon erwähnt – längere Zeit aber der Fall (Abb. 1.5).

Zu JavaFX kommen wir gleich also separat zurück. Wenn Sie noch kein JDK haben, wählen Sie das passende JDK für Ihr Betriebssystem aus (Abb. 1.6) und speichern es auf Ihren Rechner. Sie sollten minimal das JDK 11 zur Verfügung haben. Diese LTS-Version (Long Term Support) genügt, aber eine neuere Version ist kein Fehler. Selbst das JDK

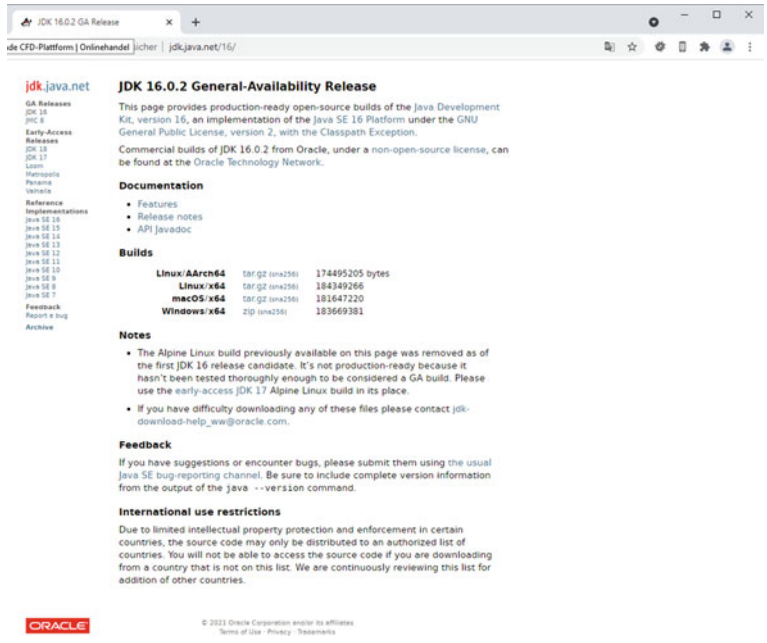

Abb. 1.6 Auswahl des passenden JDK

8 kann man noch verwenden. In einigen Situationen kann es sogar sehr sinnvoll sein, dass Sie die LTS-Version 11 des JDK als Basis verwenden, denn beispielsweise wurde die Nashorn Script Engine im JDK 15 entfernt und die benötigen Sie etwa, wenn Sie aus JavaFX heraus mit JavaScript arbeiten wollen – darauf kommen wir natürlich zurück. Es kann also durchaus sinnvoll sein, dass Sie parallel mehrere Versionen des JDK installiert haben. Ich verwende derzeit z. B. parallel die Versionen 8, 11 und die aktuell neuste Version 16 bzw. 17, die ich durch die jeweils aktuellste Version dann im Laufe der Zeit ersetze. Die beiden Versionen 8[10] und 11 halte ich aber als Fall-Back-Versionen immer bereit.

Die Installation der Java-Umgebung respektive des JDK ist absolut unproblematisch. Sie werden bei allen neueren Java-Versionen von einem typischen Installationsassistenten für Ihr Betriebssystem geführt. Dieser richtet alle notwendigen Einstellungen in Ihrem Betriebssystem ein. Anschließend sind auf Ihrem Rechner die passenden Verzeichnisse vorhanden und die Systemeinstellungen zum Ausführen von Java-Applikationen und den JDK-Tools angepasst. Mehr muss man zur Installation definitiv nicht mehr sagen. Wenn Sie das JDK installiert haben, haben Sie damit mehrere Fliegen mit einer Klappe geschlagen, denn das JDK enthält neben den eigentlichen Entwicklungstools eine vollständige Java-Laufzeitumgebung.

Das JDK selbst besteht aus einer ganzen Reihe von Programmen, die sich nach der Installation im Unterverzeichnis *bin* des JDK-Installationsverzeichnisses befinden. Dort können Sie sie über die Befehlszeilenebene mit eventuell benötigten Parametern aufrufen. Wenn Sie ein Programm des JDK ohne Parameter aufrufen, erhalten Sie in der Konsole Hinweise zu den optionalen und ebenso zu den zwingenden Parametern. Zu den Basisprogrammen des JDK im engeren Sinn zählen der Compiler (*javac*), der Interpreter (*java* beziehungsweise *javaw*), ein Debugger (*jdb*), das Dokumentations-Tool *javadoc* und das Java Archive Tool mit Namen *jar*.

1.4.3 Download und Bereitstellung von JavaFX

JavaFX wird mittlerweile als zusätzliche Ressource über das JavaFX SDK in Form eines zip-Archivs zur Verfügung gestellt und das können Sie von der OpenJFX-Webseite laden (https://openjfx.io/openjfx-docs/). Dabei kann man eine Version mit Long Term Support (LTS) oder die jeweils aktuellste Version bis hin zu Early-Access-Builds wählen (https://gluonhq.com/products/javafx/). Zum Zeitpunkt der Bucherstellung ist JavaFX 11 die LTS-Variante, JavaFX 16 ist die aktuelle Version und JavaFX 17 als Early-Access-Build verfügbar. Im Buch beziehen sich die Ausführungen auf JavaFX 16, wobei ich persönlich auch immer die letzte LTS-Version als Fall-Back installiert habe.

In jedem Fall baut JavaFX auf dem JDK auf und ist mittlerweile eine eigenständige Komponente, die Sie entweder direkt oder über Git laden können. Alternativ können Sie

[10] Diese Version wird etwa zudem immer noch von Cordova, aber auch teils bei JSF benötigt.

ein Build-System (z. B. Maven/Gradle) verwenden, um die benötigten Module aus einem Repository herunterzuladen.

Wenn Sie das JavaFX SDK direkt auf Ihren Rechner geladen haben, erhalten Sie eine ZIP-Datei, die Sie nur auf Ihren Rechner in ein passendes Verzeichnis extrahieren müssen. Sinnvoll ist etwa ein Parallelverzeichnis zum JDK selbst.

Angenommen, Sie haben das JDK 16 unter *C:\Program Files\Java\jdk-16* installiert, dann wäre das Verzeichnis *C:\Program Files\Java\javafx-sdk-16* ein empfehlenswerter Installationsort für das JavaFX SDK. Aber natürlich können Sie andere Verzeichnisstrukturen wählen.

Sehr sinnvoll ist es, die Installation von JavaFX in den Klassenpfad (Classpath) aufzunehmen – das wird noch genauer besprochen. Auch auf die Verwendung von Gradle und Maven gehen wir später noch ein.

1.4.4 Integrierte Entwicklungsumgebungen für JavaFX

Prinzipiell ist es möglich, dass Sie mithilfe eines reinen Texteditors und den Tools des JDK beziehungsweise JavaFX SDK JavaFX-Applikationen erstellen und ausführen[11]. Doch ist dieser Weg nicht sonderlich bequem und in der Praxis viel zu aufwendig, langsam und fehlerträchtig. Wenn man wirklich effektiv mit JavaFX in der Praxis programmieren möchte, führt kein Weg um eine vollständig integrierte Entwicklungsumgebung herum.

Selbstverständlich können Sie die gleichen JavaFX-Dateien mit verschiedenen Editoren beziehungsweise IDEs verarbeiten. Das werden wir in diesem Kapitel auch demonstrieren.

1.4.4.1 NetBeans

Eine IDE, mit der Sie JavaFX-Applikationen erstellen und ausführen können, ist **NetBeans.**

NetBeans wurde Ihnen lange Zeit von Oracle in verschiedenen Versionen und für verschiedene Betriebssysteme kostenlos zum Download zur Verfügung gestellt. Aber auch hier findet man die Umsetzung des Open Source-Gedankens, denn Oracle hat auch die Verantwortung für NetBeans abgegeben. Mittlerweile ist die Apache Foundation für NetBeans zuständig. Unter https://netbeans.apache.org/ finden Sie die Stelle, von wo Sie NetBeans laden können (Abb. 1.7).

Die Installation von NetBeans ist vollkommen unproblematisch – es führt ein klassischer Installationsassistent zum Erfolg.

[11] Das werden wir im Buch auch einmal durchspielen.