Learn the Java skills you will need to start developing Android apps

Learn Java for Android Development

THIRD EDITION

Jeff Friesen

Apress®

Contents at a Glance

Introduction

Smartphones and tablets are all the rage these days. Their popularity is largely due to their ability to run apps. Although the iPhone and iPad, with their growing collection of Objective-C based apps, had a head start, Android-based smartphones and tablets, with their growing collection of Java-based apps, have proven to be a strong competitor.

Not only are many iPhone/iPad app developers making money by selling their apps, but many Android app developers are also making money by selling similar apps. According to tech web sites such as The Register (`www.theregister.co.uk`), some Android app developers are making lots of money (`www.theregister.co.uk/2010/03/02/android_app_profit`).

In today's challenging economic climate, you might like to try your hand at developing Android apps and make some money. If you have good ideas, perseverance, and some artistic talent (or perhaps know some talented individuals), you are already part of the way toward achieving this goal.

Tip A good reason to consider Android app development over iPhone/iPad app development is the lower startup costs that you'll incur with Android. For example, you don't need to purchase a Mac on which to develop Android apps (a Mac is required for developing iPhone/iPad apps); your existing Windows, Linux, or Unix machine will do nicely.

Most importantly, you'll need to possess a solid understanding of the Java language and foundational application programming interfaces (APIs) before jumping into Android. After all, Android apps are written in Java and interact with many of the standard Java APIs (such as threading and input/output APIs).

I wrote *Learn Java for Android Development* to give you a solid Java foundation that you can later extend with knowledge of Android architecture, API, and tool specifics. This book will give you a strong grasp of the Java language and the many important APIs that are fundamental to Android apps and other Java applications. It will also introduce you to key development tools.

Book Organization

The first edition of this book was organized into 10 chapters and 1 appendix. The second edition was organized into 14 chapters and 3 appendixes. This third edition is organized into 16 chapters and 2 appendixes with a bonus appendix on Android app development. Each chapter in each edition offers a set of exercises that you should complete to get the most benefit from its content. Their solutions are presented in Appendix A.

Chapter 1 introduces you to Java by first focusing on Java's dual nature (language and platform). It then briefly introduces you to Oracle's Java SE, Java EE, and Java ME editions of the Java platform. You next learn how to download and install the Java SE Development Kit (JDK), and you learn some Java basics by developing and playing with three simple Java applications. After receiving a brief introduction to the Eclipse IDE, you receive a brief introduction to Android.

Chapter 2 starts you on an in-depth journey of the Java language by focusing on language fundamentals. You first learn about simple application structure and then learn about comments, identifiers (and reserved words), types, variables, expressions (and literals), and statements.

Chapter 3 continues your journey by focusing on classes and objects. You learn how to declare a class and organize applications around multiple classes. You then learn how to construct objects from classes, declare fields in classes and access these fields, declare methods in classes and call them, initialize classes and objects, and remove objects when they're no longer needed. You also learn more about arrays, which were first introduced in Chapter 2.

Chapter 4 adds to Chapter 3's pool of object-based knowledge by introducing you to the language features that take you from object-based applications to object-oriented applications. Specifically, you learn about features related to inheritance, polymorphism, and interfaces. While exploring inheritance, you learn about Java's ultimate superclass. Also, while exploring interfaces, you discover why they were included in the Java language; interfaces are not merely a workaround for Java's lack of support for multiple implementation inheritance, but serve a higher purpose.

Chapter 5 introduces you to four categories of advanced language features: nested types, packages, static imports, and exceptions.

Chapter 6 introduces you to four additional advanced language feature categories: assertions, annotations, generics, and enums.

Chapter 7 begins a trend that focuses more on APIs than language features. This chapter first introduces you to Java's `Math` and `StrictMath` math-oriented types. It then explores `Number` and its various subtypes (such as `Integer`, `Double`, and `BigDecimal`). Next you explore the string-oriented types (`String`, `StringBuffer`, and `StringBuilder`) followed by the `System` type. Finally, you explore the `Thread` class and related types for creating multithreaded applications.

Chapter 8 continues to explore Java's basic APIs by focusing on the `Random` class for generating random numbers; the References API, Reflection, the `StringTokenizer` class for breaking a string into smaller components; and the `Timer` and `TimerTask` classes for occasionally or repeatedly executing tasks.

Chapter 9 focuses exclusively on Java's Collections Framework, which provides you with a solution for organizing objects in lists, sets, queues, and maps. You also learn about collection-oriented utility classes and review Java's legacy collection types.

Chapter 10 focuses exclusively on Java's Concurrency Utilities. After receiving an introduction to this framework, you explore executors, synchronizers (such as countdown latches), concurrent collections, the Locking Framework, and atomic variables (where you discover compare-and-swap).

Chapter 11 is all about classic input/output (I/O), largely from a file perspective. In this chapter, you explore classic I/O in terms of the `File` class, `RandomAccessFile` class, various stream classes, and various writer/reader classes. My discussion of stream I/O includes coverage of Java's object serialization and deserialization mechanisms.

Chapter 12 continues to explore classic I/O by focusing on networks. You learn about the `Socket`, `ServerSocket`, `DatagramSocket`, and `MulticastSocket` classes along with related types. You also learn about the `URL` class for achieving networked I/O at a higher level and learn about the related `URI` class. After learning about the low-level `NetworkInterface` and `InterfaceAddress` classes, you explore cookie management, in terms of the `CookieHandler` and `CookieManager` classes, and the `CookiePolicy` and `CookieStore` interfaces.

Chapter 13 introduces you to New I/O. You learn about buffers, channels, selectors, regular expressions, charsets, and the `Formatter` and `Scanner` types in this chapter.

Chapter 14 focuses on databases. You first learn about the Java DB and SQLite database products, and then explore JDBC for communicating with databases created via these products.

Chapter 15 emphasizes Java's support for XML. I first provide a tutorial on this topic where you learn about the XML declaration, elements and attributes, character references and CDATA sections, namespaces, comments and processing instructions, well-formed documents, and valid documents (in terms of Document Type Definition and XML Schema). I then show you how to parse XML documents via the SAX API, parse and create XML documents via the DOM API, parse XML documents via the XMLPULL V1 API (supported by Android as an alternative to Java's StAX API), use the XPath API to concisely select nodes via location path expressions, and transform XML documents via XSLT.

Chapter 16 completes the chapter portion of this book by covering odds and ends. You first learn about useful Java 7 language features that I've successfully used in Android apps. Next, you explore classloaders, the `Console` class, design patterns (with emphasis on the Strategy pattern), double brace initialization, fluent interfaces, immutability, internationalization (in terms of locales; resource bundles; break iterators; collators; dates, time zones, and calendars; and formatters), the Logging API, the Preferences API, the `Runtime` and `Process` classes, the Java Native Interface, and the ZIP and JAR APIs.

Appendix A presents solutions to all of the exercises in Chapters 1 through 16.

Appendix B introduces you to application development in the context of *Four of a Kind*, a console-based card game.

Appendix C provides an introduction to Android app development. It gives you a chance to see how various Java language features and APIs are used in an Android context.

Unlike the other elements, Appendix C is not included in this book—it's included with the book's source code. Appendix C doesn't officially belong in *Learn Java for Android Development* because this book's focus is to prepare you for getting into Android app development by teaching you the fundamentals of the Java language, and Appendix C goes beyond that focus by giving you a tutorial on Android app development. Besides, the presence of this appendix would cause the book to exceed the 1,200-page print-on-demand limit.

Note You can download this book's source code by pointing your web browser to `www.apress.com/9781430264545` and clicking the Source Code tab followed by the Download Now link.

What Comes Next?

After you complete this book, I recommend that you check out Apress's other Android-oriented books, such as Beginning Android 4 by Grant Allen (Apress, 2012), and learn more about developing Android apps. In that book, you learn Android basics and how to create "innovative and salable applications for Android 4 mobile devices."

Thanks for purchasing this third (and my final) edition of *Learn Java for Android Development*. I hope you find it a helpful preparation for, and I wish you lots of success in achieving, a satisfying and lucrative career as an Android app developer.

—Jeff Friesen, January 2014

Chapter 1

Getting Started with Java

Android apps are written in Java and use various Java application program interfaces (APIs). Because you'll want to write your own apps, but may be unfamiliar with the Java language and these APIs, this book teaches you about Java as a first step into Android app development. It provides you with Java language fundamentals and Java APIs that are useful when developing apps.

Note This book illustrates Java concepts via non-Android Java applications. It's easier for beginners to grasp these applications than corresponding Android apps. However, I also reveal a trivial Android app toward the end of this chapter for comparison purposes.

An *API* is an interface that application code uses to communicate with other code, which is typically stored in a software library. For more information on this term, check out Wikipedia's "Application programming interface" topic at `http://en.wikipedia.org/wiki/Application_programming_interface`.

This chapter sets the stage for teaching you the essential Java concepts that you need to understand before embarking on an Android app development career. I first answer the question: "What is Java?" Next, I show you how to install the Java SE Development Kit (JDK) and introduce you to JDK tools for compiling and running Java applications.

After presenting a few simple example applications, I show you how to install and use the open source Eclipse IDE (integrated development environment) so that you can more easily (and more quickly) develop Java applications and (eventually) Android apps. I then provide you with a brief introduction to Android and show you how Java fits into the Android development paradigm.

What Is Java?

Java is a language and a platform originated by Sun Microsystems. In this section, I briefly describe this language and reveal what it means for Java to be a platform. To meet various needs, Sun organized Java into three main editions: Java SE, Java EE, and Java ME. This section briefly explores each of these editions.

Note Java has an interesting history that dates back to December 1990. At that time, James Gosling, Patrick Naughton, and Mike Sheridan (all employees of Sun Microsystems) were given the task of figuring out the next major trend in computing. They concluded that one trend would involve the convergence of computing devices and intelligent consumer appliances. Thus was born the *Green Project*.

The fruits of Green were *Star7*, a handheld wireless device featuring a five-inch color LCD screen, a SPARC processor, a sophisticated graphics capability, a version of Unix, and *Oak*, a language developed by James Gosling for writing applications to run on Star7 that he named after an oak tree growing outside of his office window at Sun. To avoid a conflict with another language of the same name, Dr. Gosling changed this language's name to Java.

Sun Microsystems subsequently evolved the Java language and platform until Oracle acquired Sun in early 2010. Check out `http://oracle.com/technetwork/java/index.html` for the latest Java news from Oracle.

Java Is a Language

Java is a language in which developers express *source code* (program text). Java's *syntax* (rules for combining symbols into language features) is partly patterned after the C and C++ languages in order to shorten the learning curve for C/C++ developers.

The following list identifies a few similarities between Java and C/C++:

- Java and C/C++ share the same single-line and multi-line comment styles. Comments let you document source code.
- Many of Java's reserved words are identical to their C/C++ counterparts (`for`, `if`, `switch`, and `while` are examples) and C++ counterparts (`catch`, `class`, `public`, and `try` are examples).
- Java supports character, double precision floating-point, floating-point, integer, long integer, and short integer primitive types via the same `char`, `double`, `float`, `int`, `long`, and `short` reserved words.
- Java supports many of the same operators, including arithmetic (+, -, *, /, and %) and conditional (`?:`) operators.
- Java uses brace characters ({ and }) to delimit blocks of statements.