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

Learn Java for Android Development

SECOND EDITION

Jeff Friesen

Apress®

Learn Java for Android Development

Second Edition

Jeff Friesen

Apress®

Learn Java for Android Development

ISBN 978-1-4302-5722-6

ISBN 978-1-4302-5723-3 (eBook)

President and Publisher: Paul Manning
Lead Editor: Steve Anglin
Developmental Editors: Tom Welsh and Matthew Moodie
Technical Reviewers: Paul Connolly, Chad Darby and Onur Cinar
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Katie Sullivan
Copy Editor: Deanna K. Hegle
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. 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 rights@apress.com, or visit www.apress.com.

Apress and friends of ED books 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 Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

To Amaury.

Contents at a Glance

About the Author xvii

About the Technical Reviewers xix

Acknowledgments xxi

Introduction xxiii

Chapter 1: Getting Started With Java 1

Chapter 2: Learning Language Fundamentals 21

Chapter 3: Discovering Classes and Objects 63

Chapter 4: Discovering Inheritance, Polymorphism, and Interfaces 105

Chapter 5: Mastering Advanced Language Features Part 1 153

Chapter 6: Mastering Advanced Language Features Part 2 197

Chapter 7: Exploring the Basic APIs Part 1 247

Chapter 8: Exploring the Basic APIs Part 2 279

Chapter 9: Exploring the Collections Framework 327

Chapter 10: Exploring Additional Utility APIs 407

Chapter 11: Performing Classic I/O 449

Chapter 12: Accessing Networks 525

■ Chapter 13: Migrating to New I/O 561
■ Chapter 14: Accessing Databases 603
■ Appendix A: Solutions to Exercises 643
■ Appendix B: Four of a Kind 713

Index 735

Contents

About the Author xvii

About the Technical Reviewers xix

Acknowledgments xxi

Introduction xxiii

Chapter 1: Getting Started With Java 1

What Is Java? 2

Java Is a Language 2

Java Is a Platform 4

Java SE, Java EE, Java ME, and Android 5

Installing and Exploring the JDK 6

Installing and Exploring the Eclipse IDE 12

Overview of Java APIs 16

Language-Support and Other Language-Oriented APIs 17

Collections-Oriented APIs 17

Additional Utility APIs 17

Classic I/O APIs 17

Networking APIs 18

New I/O APIs 18
Database APIs 18
Summary 19
Chapter 2: Learning Language Fundamentals 21
Learning Comments 21
Single-Line Comments 22
Multiline Comments 22
Javadoc Comments 23
Learning Identifiers 25
Learning Types 26
Primitive Types 26
User-Defined Types 28
Array Types 28
Learning Variables 29
Learning Expressions 30
Simple Expressions 30
Compound Expressions 34
Learning Statements 46
Assignment Statements 46
Decision Statements 47
Loop Statements 51
Break and Labeled Break Statements 56
Continue and Labeled Continue Statements 58
Summary 60
Chapter 3: Discovering Classes and Objects 63
Declaring Classes and Instantiating Objects 63
Declaring Classes 64
Instantiating Objects with the New Operator and a Constructor 64
Specifying Constructor Parameters and Local Variables 65

Encapsulating State and Behaviors....69
Representing State via Fields....70
Representing Behaviors via Methods....75
Hiding Information....84
Initializing Classes and Objects....89
Class Initializers....89
Instance Initializers....91
Initialization Order....93
Collecting Garbage....96
Revisiting Arrays....99
Summary....104
Chapter 4: Discovering Inheritance, Polymorphism, and Interfaces....105
Building Class Hierarchies....105
Extending Classes....106
The Ultimate Superclass....112
Composition....122
The Trouble with Implementation Inheritance....122
Changing Form....126
Upcasting and Late Binding....127
Abstract Classes and Abstract Methods....131
Downcasting and Runtime Type Identification....133
Covariant Return Types....136
Formalizing Class Interfaces....139
Declaring Interfaces....139
Implementing Interfaces....140
Extending Interfaces....144
Why Use Interfaces?....146
Summary....152

Chapter 5: Mastering Advanced Language Features Part 1 153

Mastering Nested Types 153

Static Member Classes 153

Nonstatic Member Classes 157

Anonymous Classes 161

Local Classes 164

Interfaces within Classes 166

Mastering Packages 167

What Are Packages? 168

The Package Statement 169

The Import Statement 169

Searching for Packages and Types 170

Playing with Packages 172

Packages and JAR Files 176

Mastering Static Imports 177

Mastering Exceptions 179

What Are Exceptions? 179

Representing Exceptions in Source Code 179

Throwing Exceptions 184

Handling Exceptions 187

Performing Cleanup 190

Summary 195

Chapter 6: Mastering Advanced Language Features Part 2 197

Mastering Assertions 197

Declaring Assertions 198

Using Assertions 199

Avoiding Assertions 205

Enabling and Disabling Assertions 206

Mastering Annotations 207
Discovering Annotations 207
Declaring Annotation Types and Annotating Source Code 210
Processing Annotations 215
Mastering Generics 217
Collections and the Need for Type Safety 217
Generic Types 220
Generic Methods 229
Arrays and Generics 232
Mastering Enums 234
The Trouble with Traditional Enumerated Types 234
The Enum Alternative 235
The Enum Class 241
Summary 245
Chapter 7: Exploring the Basic APIs Part 1 247
Exploring the Math APIs 247
Math and StrictMath 247
BigDecimal 255
BigInteger 260
Exploring String Management 264
String 264
StringBuffer and StringBuilder 268
Obtaining Package Information 270
Summary 276
Chapter 8: Exploring the Basic APIs Part 2 279
Exploring the Primitive Type Wrapper Classes 279
Boolean 280
Character 281
Float and Double 282
Integer, Long, Short, and Byte 286
Number 288

Exploring Threads 288
Runnable and Thread 289
Thread Synchronization 298
Exploring System Capabilities 314
System 314
Runtime and Process 319
Summary 324
Chapter 9: Exploring the Collections Framework 327
Exploring Collections Framework Fundamentals 327
Comparable Versus Comparator 328
Iterable and Collection 330
Exploring Lists 337
ArrayList 341
LinkedList 342
Exploring Sets 344
TreeSet 344
HashSet 346
EnumSet 350
Exploring Sorted Sets 353
Exploring Navigable Sets 361
Exploring Queues 364
PriorityQueue 365
Exploring Deques 368
ArrayDeque 372
Exploring Maps 373
TreeMap 377
HashMap 378
IdentityHashMap 384
EnumMap 386

Exploring Sorted Maps 387
Exploring Navigable Maps 390
Exploring the Arrays and Collections Utility APIs 394
Exploring the Legacy Collection APIs 398
Summary 404

Chapter 10: Exploring Additional Utility APIs 407

Exploring the Concurrency Utilities 407
Executors 407
Synchronizers 417
Concurrent Collections 420
Locks 422
Atomic Variables 425
Exploring the Date Class 426
Exploring the Formatter Class 428
Exploring the Random Class 430
Exploring the Scanner Class 432
Exploring the ZIP and JAR APIs 434
Exploring the ZIP API 435
Exploring the JAR API 442
Summary 447

Chapter 11: Performing Classic I/O 449

Working with the File API 449
Working with the RandomAccessFile API 462
Working with Streams 473
Stream Classes Overview 474
OutputStream and InputStream 475
ByteArrayOutputStream and ByteArrayInputStream 478
FileOutputStream and FileInputStream 479
PipedOutputStream and PipedInputStream 482
FilterOutputStream and FilterInputStream 485

BufferedOutputStream and BufferedInputStream 493
DataOutputStream and DataInputStream 494
Object Serialization and Deserialization 496
PrintStream 510
Working with Writers and Readers 511
Writer and Reader Classes Overview 512
Writer and Reader 514
OutputStreamWriter and InputStreamReader 514
FileWriter and FileReader 516
Summary 524
Chapter 12: Accessing Networks 525
Accessing Networks via Sockets 526
Socket Addresses 528
Socket Options 529
Socket and ServerSocket 530
DatagramSocket and MulticastSocket 536
Accessing Networks via URLs 543
URL and URLConnection 543
URLEncoder and URLDecoder 547
Accessing Network Interfaces and Interface Addresses 549
Managing Cookies 555
Summary 559
Chapter 13: Migrating to New I/O 561
Working with Buffers 562
Buffer and Its Children 562
Buffers in Depth 566
Working with Channels 575
Channel and Its Children 576
Channels in Depth 580

Working With Regular Expressions....589
Pattern, PatternSyntaxException, and Matcher....589
Character Classes....593
Capturing Groups....594
Boundary Matchers and Zero-Length Matches....595
Quantifiers....596
Practical Regular Expressions....598
Summary....601
Chapter 14: Accessing Databases....603
Introducing Java DB....604
Java DB Installation and Configuration....605
Java DB Demos....607
Java DB Command-Line Tools....609
Introducing SQLite....611
Accessing Databases via JDBC....613
Data Sources, Drivers, and Connections....613
Exceptions....616
Statements....620
Metadata....633
Summary....639
Appendix A: Solutions to Exercises....643
Chapter 1: Getting Started with Java....643
Chapter 2: Learning Language Fundamentals....644
Chapter 3: Discovering Classes and Objects....647
Chapter 4: Discovering Inheritance, Polymorphism, and Interfaces....651
Chapter 5: Mastering Advanced Language Features Part 1....659
Chapter 6: Mastering Advanced Language Features Part 2....666
Chapter 7: Exploring the Basic APIs Part 1....671
Chapter 8: Exploring the Basic APIs Part 2....676
Chapter 9: Exploring the Collections Framework....679

Chapter 10: Exploring Additional Utility APIs 686
Chapter 11: Performing Classic I/O 689
Chapter 12: Accessing Networks 698
Chapter 13: Migrating to New I/O 704
Chapter 14: Accessing Databases 709

Appendix B: Four of a Kind 713

Understanding Four of a Kind 713
Modeling Four of a Kind in Pseudocode 714
Converting Pseudocode to Java Code 715
Compiling, Running, and Distributing FourOfAKind 731

Index 735

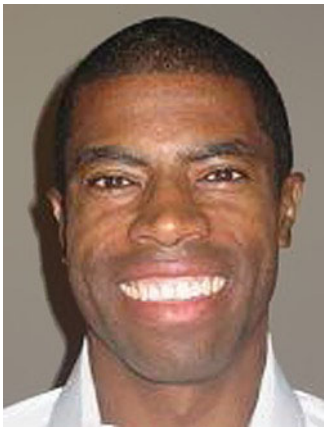
About the Author

Jeff Friesen is a freelance tutor and software developer with an emphasis on Java (and now Android). In addition to writing this book, Jeff has written numerous articles on Java and other technologies for JavaWorld (www.javaworld.com), informIT (www.informit.com), java.net, DevSource (www.devsource.com), SitePoint (www.sitepoint.com), BuildMobile (www.buildmobile.com), and JSPro (www.jspro.com). Jeff can be contacted via his web site at tutortutor.ca.

About the Technical Reviewers

Paul Connolly is the Director of Engineering for Atypon Systems' RightSuite product line. RightSuite is an enterprise access-control and commerce solution used by many of the world's largest publishing and media companies. Paul enjoys designing and implementing high-performance, enterprise-class software systems. He is also an active contributor in the open-source community.

Prior to joining Atypon Systems, Paul worked as a senior software engineer at Standard & Poor's where he designed and developed key communications systems. Paul is a Sun Certified Java Programmer, Sun Certified Business Component Developer, and a Sun Certified Web Component Developer. Paul lives in Rochester, NY, with his wife Marina and daughter Olivia.



Chád Darby is an author, instructor and speaker in the Java development world. As a recognized authority on Java applications and architectures, he has presented technical sessions at software development conferences worldwide. In his 15 years as a professional software architect, he's had the opportunity to work for Blue Cross/Blue Shield, Merck, Boeing, Northrop Grumman, and a handful of startup companies.

Chád is a contributing author to several Java books, including *Professional Java E-Commerce* (Wrox Press), *Beginning Java Networking* (Wrox Press), and *XML and Web Services Unleashed* (Sams Publishing). Chád has Java certifications from Sun Microsystems and IBM. He holds a B.S. in Computer Science from Carnegie Mellon University.

You can read Chád's blog at `www.luv2code.com` and follow him on Twitter `@darbyluvs2code`.

Onur Cinar is the author of *Android Apps with Eclipse*, and *Pro Android C++* with the NDK books from Apress. He has over 17 years of experience in design, development, and management of large scale complex software projects, primarily in mobile and telecommunication space. His expertise spans VoIP, video communication, mobile applications, grid computing, and networking technologies on diverse platforms. He has been actively working with the Android platform since its beginning. He has a B.S. degree in Computer Science from Drexel University in Philadelphia, PA. He is currently working at the Skype division of Microsoft as the Sr. Product Engineering Manager for the Skype client on Android platform.

Acknowledgments

I thank Steve Anglin for contacting me to write this book; Katie Sullivan for guiding me through the various aspects of this project; Tom Welsh and Matthew Moodie for helping me with the development of my chapters; and Paul Connolly, Chád Darby, and Onur Cinar for their diligence in catching various flaws that would otherwise have made it into this book.

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 are proving to be a strong competitor.

Not only are many iPhone/iPad app developers making money by selling their apps, 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, perhaps you would 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 will 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 important, you will 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 (e.g., 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 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 one appendix. The second edition is organized into 14 chapters and three appendixes. 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 an appendix.

In Chapter 1 I introduce you to Java by first focusing on Java's dual nature (language and platform). I then briefly introduce you to Oracle's Java SE, Java EE, and Java ME editions of the Java development software, as well as Google's Android edition. You next learn how to download and install the Java SE Development Kit (JDK) and learn some Java basics by developing and playing with a pair of simple Java applications. After a brief introduction to the Eclipse IDE, you receive an overview of the various APIs covered in this book.

In Chapter 2 I start you on an in-depth journey of the Java language by focusing on language fundamentals. You learn about comments, identifiers (and reserved words), types, variables, expressions (and literals), and statements.

In Chapter 3 I continue your journey by focusing on classes and objects. You learn how to declare a class and instantiate objects from the class, how to declare fields within the class and access these fields, how to declare methods within the class and call them, how to initialize classes and objects, and how to get rid of objects when they are no longer needed. You also learn more about arrays, which are first introduced in Chapter 2.

In Chapter 4 I add to Chapter 3's pool of object-based knowledge by introducing you to 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.

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

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

In Chapter 7 I begin a trend that focuses more on APIs than language features. In this chapter I first introduce you to many of Java's math-oriented types (e.g., `Math`, `StrictMath`, `BigDecimal`, and `BigInteger`) and then introduce you to its string-oriented types (e.g., `String`, `StringBuffer`, and `StringBuilder`). Finally, you explore the `Package` class for obtaining package information.

In Chapter 8 I continue to explore Java's basic APIs by focusing on primitive type wrapper classes, threading, and system-oriented APIs.

In Chapter 9 I focus 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 utility types.

In Chapter 10 I continue to explore Java's utility APIs by introducing you to Concurrency Utilities, the `Date` class (for representing time), the `Formatter` class (for formatting data items), the `Random` class (for generating random numbers), the `Timer` and `TimerTask` classes (for occasionally or repeatedly executing tasks), and the APIs for working with ZIP and JAR files.

Chapter 11 is all about classic input/output (I/O), seen 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.

In Chapter 12 I continue 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. 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.

In Chapter 13 I introduce you to New I/O. You learn about buffers, channels, and regular expressions in this chapter. I would have loved to cover selectors and charsets as well but could not do so for lack of space. To cover selectors, I would also have had to discuss socket channels, but I could only cover file channels. However, Chapter 11 does give you a small taste of charsets.

In Chapter 14 I wrap up the chapter portion of this book by focusing 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.

In Appendix A I present solutions to all exercises in Chapters 1 through 14.

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

In Appendix C, which is available as a separate PDF file that's bundled with this book's code, I introduce you to advanced APIs (e.g., Reflection and References) as well as APIs that might not be as useful in an Android app context (e.g., Preferences—Android offers its own solution.)

> **Note** You can download this book's source code by pointing your web browser to `www.apress.com/book/view/1430257226` and clicking the Source Code tab followed by the Download Now link.

First Edition vs. Second Edition

The first edition of this book debuted in September 2010. I'm generally pleased with the first edition, and I thank everyone who purchased it. However, as was pointed out to me on multiple occasions, the first edition is flawed. As well as small technical errors, there are certain organizational and other issues that got by me during that book's development.

For starters, I should not have introduced the *Four of a Kind* card game in Chapter 1. It was too complicated for many readers to encounter at this point. As a result, I've moved the game to Appendix B so as not to overwhelm Java beginners.

Also, I attempted to cover language fundamentals (e.g., statements and expressions) with the basics of classes and objects in the same chapter. Although some people appreciated this approach, it turned out to be too confusing for beginners; I apologize to readers who felt this way. In the second edition I separate these aspects of the Java language to (hopefully) sort out this problem. In Chapter 2 I focus on statements, expressions, and other non-class/non-object fundamentals; in Chapter 3 I focus on classes and objects.

Another issue was the inclusion of complex APIs that are either infrequently used when developing Android apps or are mostly irrelevant to Android developers. Examples include References, Reflection, Preferences, and Internationalization. I moved these APIs to Appendix C so that I could cover simpler (and possibly more useful) APIs such as ZIP and Timer. (I also included additional new content in Appendix C.)

While writing the first edition, I planned to go further by covering Java's support for networking and database access (via JDBC), security, XML, New I/O, and so on. I foolishly presented a plan to write six free chapters, but only managed to complete portions of three chapters.

Unfortunately, my original plan for six free chapters was flawed. For instance, I planned to write a free chapter on networking that would come after a free chapter on New I/O. That wasn't a good organization because New I/O includes socket channels, and so the networking chapter should have preceded a chapter on New I/O.

Also, I've learned (via various blogs about Android and security) that Java's security features aren't as necessary in an Android context. Because this book partly focuses on presenting the most useful Java APIs for subsequent use in an Android context, coverage of Java's security APIs is probably not as important (although I could be wrong).

> **Note** There are no free chapters to supplement the second edition. However, Appendix C is a freebie. Also, I might eventually offer some additional material (perhaps coverage of socket channels, selectors, and charsets) on my web site (see http://tutortutor.ca/cgi-bin/makepage.cgi?/books/ljfad).

What Comes Next?

After you complete this book, I recommend that you obtain a copy of *Beginning Android 4* by Grant Allen (Apress, 2012) and start learning how to develop Android apps. In that book, you learn Android basics and how to create "innovative and salable applications for Android 4 mobile devices." Rather than give a few superficial details of Android development, *Learn Java for Android Development Second Edition* concentrates on teaching you the Java language and APIs such as Collections that you will need to use in your apps. If you don't first understand Java, how can you proceed to understand Android?

> **Note** I also recommend that you check out the second edition of *Android Recipes* (see www.apress.com/9781430246145). Although the content of that book largely contains independent recipes for learning all kinds of things about Android, Chapter 1 contains a summarized and rapid introduction to Android app development. You will learn much about Android basics from reading that chapter.

Thanks for purchasing my book. 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.

Chapter 1

Getting Started With Java

Android is Google's software stack for mobile devices. This stack consists of applications (or apps as they are commonly called), a *virtual machine* (software-based processor and associated environment) in which apps run, *middleware* (software that sits on top of the operating system and provides various services to the virtual machine and its apps), and a Linux-based operating system.

Android apps are written in Java and use various Java Application Program Interfaces (APIs). Because you will 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 app development. It provides you with the fundamentals of the Java language 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.
>
> An API refers to an interface that an application's code uses to communicate with other code, which is typically stored in some kind of software library. For more information on this term, check out Wikipedia's "Application programming interface" topic (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 you embark on your Android app development career. I first answer the "What is Java?" question. I next show you how to install the Java SE Development Kit (JDK), and introduce you to JDK tools for compiling and running Java applications.

After showing 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 provide you with a high-level overview of various Java APIs that you can access from your Java applications and Android apps. In subsequent chapters, you'll explore these and other useful APIs in greater detail.

> **Note** Chapter 1 is short but intense, presenting many concepts that you'll encounter in more detail throughout this book. If you are new to Java, you might find yourself a little overwhelmed by these concepts. However, any fog should clear as you progress through remaining chapters. If you still feel somewhat confused, please contact me (`jeff@tutortutor.ca`) with your questions and I'll do my best to help you.

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 also briefly explores each of these editions, along with Android.

> **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, and a version of Unix; and *Oak*, a language developed by James Gosling for writing applications to run on Star7, which 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 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 multiline 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, and 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.

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

- Java supports an additional comment style known as Javadoc. (I briefly introduce Javadoc in Chapter 2.)
- Java provides reserved words not found in C/C++ (`extends`, `strictfp`, `synchronized`, and `transient` are examples).
- Java doesn't require machine-specific knowledge. It supports the byte integer type (see http://en.wikipedia.org/wiki/Integer_(computer_science)); doesn't provide a signed version of the character type; and doesn't provide unsigned versions of integer, long integer, and short integer. Furthermore, all of Java's primitive types have guaranteed implementation sizes, which is an important part of achieving portability (discussed later). The same cannot be said of equivalent primitive types in C and C++.
- Java provides operators not found in C/C++. These operators include `instanceof` and `>>>` (unsigned right shift).
- Java provides labeled break and continue statements that you will not find in C/C++.

You will learn about single-line and multiline comments in Chapter 2. Also, you will learn about reserved words, primitive types, operators, blocks, and statements (including labeled break and continue) in that chapter.

Java was designed to be a safer language than C/C++. It achieves safety in part by not letting you overload operators and by omitting C/C++ features such as *pointers* (variables containing addresses—see http://en.wikipedia.org/wiki/Pointer_(computer_programming)).

Java also achieves safety by modifying certain C/C++ features. For example, loops must be controlled by Boolean expressions instead of integer expressions where 0 is false and a nonzero value is true. (There is a discussion of loops and expressions in Chapter 2.)

Suppose you must code a C/C++ while loop that repeats no more than 10 times. Being tired, you specify `while (x) x++;` (assume that x is an integer-based variable initialized to 0—I discuss variables in Chapter 2) where x++ adds 1 to x's value. This loop doesn't stop when x reaches 10; you have introduced a bug.

This problem is less likely to occur in Java because it complains when it sees `while (x)`. This complaint requires you to recheck your expression, and you will then most likely specify `while (x != 10)`. Not only is safety improved (you cannot specify just x), meaning is also clarified: `while (x != 10)` is more meaningful than `while (x)`.

These and other fundamental language features support classes, objects, inheritance, polymorphism, and interfaces. Java also provides advanced features related to nested types,

packages, static imports, exceptions, assertions, annotations, generics, enums, and more. Subsequent chapters explore most of these language features.

Java Is a Platform

Java is a platform consisting of a virtual machine and an execution environment. The *virtual machine* is a software-based processor that presents an instruction set. The *execution environment* consists of libraries for running programs and interacting with the underlying operating system.

The execution environment includes a huge library of prebuilt classfiles that perform common tasks, such as math operations (e.g., trigonometry) and network communications. This library is commonly referred to as the *standard class library*.

A special Java program known as the *Java compiler* translates source code into instructions (and associated data) that are executed by the virtual machine. These instructions are known as *bytecode*.

The compiler stores a program's bytecode and data in files having the `.class` extension. These files are known as *classfiles* because they typically store the compiled equivalent of classes, a language feature discussed in Chapter 3.

A Java program executes via a tool (e.g., `java`) that loads and starts the virtual machine and passes the program's main classfile to the machine. The virtual machine uses a *classloader* (a virtual machine or execution environment component) to load the classfile.

After the classfile has been loaded, the virtual machine's *bytecode verifier* component makes sure that the classfile's bytecode is valid and doesn't compromise security. The verifier terminates the virtual machine when it finds a problem with the bytecode.

Assuming that all is well with the classfile's bytecode, the virtual machine's *interpreter* interprets the bytecode one instruction at a time. *Interpretation* consists of identifying bytecode instructions and executing equivalent native instructions.

> **Note** *Native instructions* (also known as *native code*) are the instructions understood by the underlying platform's physical processor.

When the interpreter learns that a sequence of bytecode instructions is executed repeatedly, it informs the virtual machine's *Just in Time (JIT) compiler* to compile these instructions into native code.

JIT compilation is performed only once for a given sequence of bytecode instructions. Because the native instructions execute instead of the associated bytecode instruction sequence, the program executes much faster.

During execution, the interpreter might encounter a request to execute another classfile's bytecode. When that happens, it asks the classloader to load the classfile and the bytecode verifier to verify the bytecode prior to executing that bytecode.

The platform side of Java promotes *portability* by providing an abstraction over the underlying platform. As a result, the same bytecode runs unchanged on Windows-based, Linux-based, Mac OS X-based, and other platforms.

> **Note** Java was introduced with the slogan "write once, run anywhere." Although Java goes to great lengths to enforce portability (e.g., an integer is always 32 binary digits [bits] and a long integer is always 64 bits—see http://en.wikipedia.org/wiki/Bit to learn about binary digits), it doesn't always succeed. For example, despite being mostly platform independent, certain parts of Java (e.g., the scheduling of threads, discussed in Chapter 8) vary from underlying platform to underlying platform.

The platform side of Java also promotes *security* by providing a secure environment (e.g., the bytecode verifier) in which code executes. The goal is to prevent malicious code from corrupting the underlying platform (and possibly stealing sensitive information).

Java SE, Java EE, Java ME, and Android

Developers use different editions of the Java platform to create Java programs that run on desktop computers, web browsers, web servers, mobile information devices (e.g., feature phones), and embedded devices (e.g., television set-top boxes):

- *Java Platform, Standard Edition (Java SE)*: The Java platform for developing *applications*, which are stand-alone programs that run on desktops. Java SE is also used to develop *applets*, which are programs that run in web browsers.
- *Java Platform, Enterprise Edition (Java EE)*: The Java platform for developing enterprise-oriented applications and *servlets*, which are server programs that conform to Java EE's Servlet API. Java EE is built on top of Java SE.
- *Java Platform, Micro Edition (Java ME)*: The Java platform for developing *MIDlets*, which are programs that run on mobile information devices, and *Xlets*, which are programs that run on embedded devices.

Developers also use a special Google-created edition of the Java platform (see http://developer.android.com/index.html) to create Android apps that run on Android-enabled devices. This edition is known as the *Android platform*.

Google's Android platform presents a *Dalvik virtual machine* that runs on top of a specially modified Linux kernel. An Android app's Java source code is compiled into Java classfiles, which are then translated into a special file for Dalvik to execute.

> **Note** Learn more about the Android OS via Wikipedia's "Android (operating system)" entry (http://en.wikipedia.org/wiki/Android_(operating_system)) and about the Dalvik virtual machine via Wikipedia's "Dalvik (software)" entry (http://en.wikipedia.org/wiki/Dalvik_(software)).

In this book, I cover the Java language (supported by Java SE and Android) and various Java SE APIs that are also supported by Android. I focus on language features through Java version 5 and on Java APIs through Java 5, with a small amount of Java 6.

> **Note** Google's Android platform is based on an open source release of Java 5. It doesn't officially recognize language features newer than Java 5, although it's possible to add this support (see www.informit.com/articles/article.aspx?p=1966024). Regarding APIs, this platform supports APIs from Java 6 and previous Java versions. Also, it provides its own unique APIs.

Installing and Exploring the JDK

The *Java Runtime Environment (JRE)* implements the Java SE platform and makes it possible to run Java programs. The public JRE can be downloaded from Oracle's Java SE Downloads page (http://oracle.com/technetwork/java/javase/downloads/index.html).

However, the public JRE doesn't make it possible to develop Java programs. For that task, you need to download and install the *Java SE Development Kit (JDK)*, which contains development tools (including the Java compiler) and a private JRE.

> **Note** JDK 1.0 was the first JDK to be released (in May 1995). Until JDK 6 arrived, JDK stood for Java Development Kit (SE was not part of the title). Over the years, numerous JDKs have been released, with JDK 7 being current at time of writing.
>
> Each JDK's version number identifies a version of Java. For example, JDK 1.0 identifies Java 1.0, and JDK 5 identifies Java 5.0. JDK 5 was the first JDK to also provide an internal version number: 1.5.0.
>
> Google doesn't provide a JDK. What it does provide is similar to a JRE, but with an Android focus.

The Java SE Downloads page also provides access to the current JDK, which is JDK 7 Update 9 at time of writing. Click the Download JDK link (on the page at http://oracle.com/technetwork/java/javase/downloads/index.html) to download the current JDK's installer program for your platform.

The JDK installer places the JDK in a home directory. (It can also install the public JRE in another directory.) On my Windows 7 platform, the home directory is `C:\Program Files\Java\jdk1.7.0_06`. (I currently use JDK 7 Update 6.)

> **Tip** After installing the JDK, you should add the `bin` subdirectory to your platform's `PATH` environment variable (see http://java.com/en/download/help/path.xml) so you can execute JDK tools from any directory. Also, you might want to create a `projects` subdirectory of the JDK's home directory to organize your Java projects and create a separate subdirectory within `projects` for each of these projects.