

C O R E
JAVA

Volume I—Fundamentals

ELEVENTH EDITION



CAY S. HORSTMANN

Core Java

Volume I—Fundamentals

Eleventh Edition

This page intentionally left blank

Core Java

Volume I—Fundamentals

Eleventh Edition

Cay S. Horstmann

◆ Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town

Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City

São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com

Library of Congress Preassigned Control Number: 2018942070

Copyright © 2019 Pearson Education Inc.

Portions copyright © 1996-2013 Oracle and/or its affiliates. All Rights Reserved.

Oracle America Inc. does not make any representations or warranties as to the accuracy, adequacy or completeness of any information contained in this work, and is not responsible for any errors or omissions.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided "as is" without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services. The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft® Windows®, and Microsoft Office® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions/.

ISBN-13: 978-0-13-516630-7

ISBN-10: 0-13-516630-6

ScoutAutomatedPrintCode

Contents



<i>Preface</i>	<i>xix</i>
<i>Acknowledgments</i>	<i>xxv</i>
Chapter 1: An Introduction to Java	1
1.1 Java as a Programming Platform	1
1.2 The Java “White Paper” Buzzwords	2
1.2.1 Simple	3
1.2.2 Object-Oriented	4
1.2.3 Distributed	4
1.2.4 Robust	4
1.2.5 Secure	5
1.2.6 Architecture-Neutral	6
1.2.7 Portable	6
1.2.8 Interpreted	7
1.2.9 High-Performance	7
1.2.10 Multithreaded	8
1.2.11 Dynamic	8
1.3 Java Applets and the Internet	9
1.4 A Short History of Java	10
1.5 Common Misconceptions about Java	13
Chapter 2: The Java Programming Environment	17
2.1 Installing the Java Development Kit	18
2.1.1 Downloading the JDK	18
2.1.2 Setting up the JDK	20
2.1.3 Installing Source Files and Documentation	22
2.2 Using the Command-Line Tools	23
2.3 Using an Integrated Development Environment	29
2.4 JShell	32

Chapter 3: Fundamental Programming Structures in Java	37
3.1 A Simple Java Program	38
3.2 Comments	41
3.3 Data Types	42
3.3.1 Integer Types	43
3.3.2 Floating-Point Types	44
3.3.3 The char Type	46
3.3.4 Unicode and the char Type	47
3.3.5 The boolean Type	48
3.4 Variables and Constants	48
3.4.1 Declaring Variables	48
3.4.2 Initializing Variables	50
3.4.3 Constants	51
3.4.4 Enumerated Types	52
3.5 Operators	52
3.5.1 Arithmetic Operators	52
3.5.2 Mathematical Functions and Constants	54
3.5.3 Conversions between Numeric Types	56
3.5.4 Casts	57
3.5.5 Combining Assignment with Operators	58
3.5.6 Increment and Decrement Operators	58
3.5.7 Relational and boolean Operators	59
3.5.8 Bitwise Operators	60
3.5.9 Parentheses and Operator Hierarchy	61
3.6 Strings	62
3.6.1 Substrings	62
3.6.2 Concatenation	63
3.6.3 Strings Are Immutable	63
3.6.4 Testing Strings for Equality	65
3.6.5 Empty and Null Strings	66
3.6.6 Code Points and Code Units	66
3.6.7 The String API	68
3.6.8 Reading the Online API Documentation	71
3.6.9 Building Strings	74
3.7 Input and Output	75

3.7.1	Reading Input	75
3.7.2	Formatting Output	78
3.7.3	File Input and Output	83
3.8	Control Flow	86
3.8.1	Block Scope	86
3.8.2	Conditional Statements	87
3.8.3	Loops	91
3.8.4	Determinate Loops	95
3.8.5	Multiple Selections The switch Statement	99
3.8.6	Statements That Break Control Flow	102
3.9	Big Numbers	105
3.10	Arrays	108
3.10.1	Declaring Arrays	108
3.10.2	Accessing Array Elements	109
3.10.3	The “for each” Loop	110
3.10.4	Array Copying	111
3.10.5	Command-Line Parameters	112
3.10.6	Array Sorting	113
3.10.7	Multidimensional Arrays	116
3.10.8	Ragged Arrays	120
Chapter 4: Objects and Classes	125	
4.1	Introduction to Object-Oriented Programming	126
4.1.1	Classes	127
4.1.2	Objects	128
4.1.3	Identifying Classes	129
4.1.4	Relationships between Classes	129
4.2	Using Predefined Classes	131
4.2.1	Objects and Object Variables	132
4.2.2	The LocalDate Class of the Java Library	135
4.2.3	Mutator and Accessor Methods	138
4.3	Defining Your Own Classes	141
4.3.1	An Employee Class	142
4.3.2	Use of Multiple Source Files	145
4.3.3	Dissecting the Employee Class	146
4.3.4	First Steps with Constructors	146

4.3.5	Declaring Local Variables with <code>var</code>	148
4.3.6	Working with <code>null</code> References	148
4.3.7	Implicit and Explicit Parameters	150
4.3.8	Benefits of Encapsulation	151
4.3.9	Class-Based Access Privileges	154
4.3.10	Private Methods	155
4.3.11	Final Instance Fields	155
4.4	Static Fields and Methods	156
4.4.1	Static Fields	156
4.4.2	Static Constants	157
4.4.3	Static Methods	158
4.4.4	Factory Methods	159
4.4.5	The <code>main</code> Method	160
4.5	Method Parameters	163
4.6	Object Construction	170
4.6.1	Overloading	170
4.6.2	Default Field Initialization	171
4.6.3	The Constructor with No Arguments	172
4.6.4	Explicit Field Initialization	173
4.6.5	Parameter Names	174
4.6.6	Calling Another Constructor	175
4.6.7	Initialization Blocks	175
4.6.8	Object Destruction and the <code>finalize</code> Method	180
4.7	Packages	180
4.7.1	Package Names	181
4.7.2	Class Importation	181
4.7.3	Static Imports	183
4.7.4	Addition of a Class into a Package	184
4.7.5	Package Access	187
4.7.6	The Class Path	189
4.7.7	Setting the Class Path	191
4.8	JAR Files	192
4.8.1	Creating JAR files	192
4.8.2	The Manifest	193
4.8.3	Executable JAR Files	194

4.8.4	Multi-Release JAR Files	195
4.8.5	A Note about Command-Line Options	197
4.9	Documentation Comments	198
4.9.1	Comment Insertion	199
4.9.2	Class Comments	199
4.9.3	Method Comments	200
4.9.4	Field Comments	201
4.9.5	General Comments	201
4.9.6	Package Comments	202
4.9.7	Comment Extraction	203
4.10	Class Design Hints	204
Chapter 5: Inheritance	207
5.1	Classes, Superclasses, and Subclasses	208
5.1.1	Defining Subclasses	208
5.1.2	Overriding Methods	210
5.1.3	Subclass Constructors	211
5.1.4	Inheritance Hierarchies	216
5.1.5	Polymorphism	217
5.1.6	Understanding Method Calls	218
5.1.7	Preventing Inheritance: Final Classes and Methods	221
5.1.8	Casting	223
5.1.9	Abstract Classes	225
5.1.10	Protected Access	231
5.2	Object: The Cosmic Superclass	232
5.2.1	Variables of Type Object	232
5.2.2	The equals Method	233
5.2.3	Equality Testing and Inheritance	234
5.2.4	The hashCode Method	238
5.2.5	The toString Method	241
5.3	Generic Array Lists	248
5.3.1	Declaring Array Lists	248
5.3.2	Accessing Array List Elements	251
5.3.3	Compatibility between Typed and Raw Array Lists	255
5.4	Object Wrappers and Autoboxing	256
5.5	Methods with a Variable Number of Parameters	260

5.6	Enumeration Classes	261
5.7	Reflection	264
5.7.1	The Class Class	264
5.7.2	A Primer on Declaring Exceptions	267
5.7.3	Resources	268
5.7.4	Using Reflection to Analyze the Capabilities of Classes	271
5.7.5	Using Reflection to Analyze Objects at Runtime	277
5.7.6	Using Reflection to Write Generic Array Code	283
5.7.7	Invoking Arbitrary Methods and Constructors	286
5.8	Design Hints for Inheritance	290
Chapter 6: Interfaces, Lambda Expressions, and Inner Classes		295
6.1	Interfaces	296
6.1.1	The Interface Concept	296
6.1.2	Properties of Interfaces	303
6.1.3	Interfaces and Abstract Classes	305
6.1.4	Static and Private Methods	306
6.1.5	Default Methods	307
6.1.6	Resolving Default Method Conflicts	308
6.1.7	Interfaces and Callbacks	310
6.1.8	The Comparator Interface	313
6.1.9	Object Cloning	314
6.2	Lambda Expressions	322
6.2.1	Why Lambdas?	322
6.2.2	The Syntax of Lambda Expressions	323
6.2.3	Functional Interfaces	326
6.2.4	Method References	328
6.2.5	Constructor References	332
6.2.6	Variable Scope	333
6.2.7	Processing Lambda Expressions	335
6.2.8	More about Comparators	339
6.3	Inner Classes	340
6.3.1	Use of an Inner Class to Access Object State	341
6.3.2	Special Syntax Rules for Inner Classes	345
6.3.3	Are Inner Classes Useful? Actually Necessary? Secure?	346
6.3.4	Local Inner Classes	349

6.3.5	Accessing Variables from Outer Methods	350
6.3.6	Anonymous Inner Classes	352
6.3.7	Static Inner Classes	356
6.4	Service Loaders	360
6.5	Proxies	362
6.5.1	When to Use Proxies	363
6.5.2	Creating Proxy Objects	363
6.5.3	Properties of Proxy Classes	368
Chapter 7: Exceptions, Assertions, and Logging	371	
7.1	Dealing with Errors	372
7.1.1	The Classification of Exceptions	373
7.1.2	Declaring Checked Exceptions	375
7.1.3	How to Throw an Exception	378
7.1.4	Creating Exception Classes	380
7.2	Catching Exceptions	381
7.2.1	Catching an Exception	381
7.2.2	Catching Multiple Exceptions	383
7.2.3	Rethrowing and Chaining Exceptions	384
7.2.4	The finally Clause	386
7.2.5	The try-with-Resources Statement	389
7.2.6	Analyzing Stack Trace Elements	391
7.3	Tips for Using Exceptions	396
7.4	Using Assertions	399
7.4.1	The Assertion Concept	399
7.4.2	Assertion Enabling and Disabling	400
7.4.3	Using Assertions for Parameter Checking	401
7.4.4	Using Assertions for Documenting Assumptions	402
7.5	Logging	403
7.5.1	Basic Logging	404
7.5.2	Advanced Logging	405
7.5.3	Changing the Log Manager Configuration	407
7.5.4	Localization	409
7.5.5	Handlers	410
7.5.6	Filters	414
7.5.7	Formatters	415

7.5.8	A Logging Recipe	415
7.6	Debugging Tips	425
Chapter 8:	Generic Programming	431
8.1	Why Generic Programming?	432
8.1.1	The Advantage of Type Parameters	432
8.1.2	Who Wants to Be a Generic Programmer?	433
8.2	Defining a Simple Generic Class	434
8.3	Generic Methods	437
8.4	Bounds for Type Variables	438
8.5	Generic Code and the Virtual Machine	441
8.5.1	Type Erasure	441
8.5.2	Translating Generic Expressions	442
8.5.3	Translating Generic Methods	443
8.5.4	Calling Legacy Code	445
8.6	Restrictions and Limitations	447
8.6.1	Type Parameters Cannot Be Instantiated with Primitive Types	447
8.6.2	Runtime Type Inquiry Only Works with Raw Types	447
8.6.3	You Cannot Create Arrays of Parameterized Types	448
8.6.4	Varargs Warnings	448
8.6.5	You Cannot Instantiate Type Variables	450
8.6.6	You Cannot Construct a Generic Array	451
8.6.7	Type Variables Are Not Valid in Static Contexts of Generic Classes	452
8.6.8	You Cannot Throw or Catch Instances of a Generic Class	453
8.6.9	You Can Defeat Checked Exception Checking	454
8.6.10	Beware of Clashes after Erasure	455
8.7	Inheritance Rules for Generic Types	457
8.8	Wildcard Types	459
8.8.1	The Wildcard Concept	459
8.8.2	Supertype Bounds for Wildcards	461
8.8.3	Unbounded Wildcards	464
8.8.4	Wildcard Capture	465
8.9	Reflection and Generics	467

8.9.1	The Generic Class Class	467
8.9.2	Using <code>Class<T></code> Parameters for Type Matching	469
8.9.3	Generic Type Information in the Virtual Machine	469
8.9.4	Type Literals	473
Chapter 9: Collections	481	
9.1	The Java Collections Framework	482
9.1.1	Separating Collection Interfaces and Implementation	482
9.1.2	The Collection Interface	485
9.1.3	Iterators	485
9.1.4	Generic Utility Methods	489
9.2	Interfaces in the Collections Framework	492
9.3	Concrete Collections	494
9.3.1	Linked Lists	496
9.3.2	Array Lists	507
9.3.3	Hash Sets	507
9.3.4	Tree Sets	511
9.3.5	Queues and Deques	516
9.3.6	Priority Queues	518
9.4	Maps	519
9.4.1	Basic Map Operations	519
9.4.2	Updating Map Entries	523
9.4.3	Map Views	525
9.4.4	Weak Hash Maps	526
9.4.5	Linked Hash Sets and Maps	527
9.4.6	Enumeration Sets and Maps	529
9.4.7	Identity Hash Maps	530
9.5	Views and Wrappers	532
9.5.1	Small Collections	532
9.5.2	Subranges	534
9.5.3	Unmodifiable Views	535
9.5.4	Synchronized Views	536
9.5.5	Checked Views	536
9.5.6	A Note on Optional Operations	537
9.6	Algorithms	541
9.6.1	Why Generic Algorithms?	541

9.6.2	Sorting and Shuffling	543
9.6.3	Binary Search	546
9.6.4	Simple Algorithms	547
9.6.5	Bulk Operations	549
9.6.6	Converting between Collections and Arrays	550
9.6.7	Writing Your Own Algorithms	551
9.7	Legacy Collections	552
9.7.1	The Hashtable Class	553
9.7.2	Enumerations	553
9.7.3	Property Maps	555
9.7.4	Stacks	558
9.7.5	Bit Sets	559
Chapter 10: Graphical User Interface Programming		565
10.1	A History of Java User Interface Toolkits	565
10.2	Displaying Frames	567
10.2.1	Creating a Frame	568
10.2.2	Frame Properties	570
10.3	Displaying Information in a Component	574
10.3.1	Working with 2D Shapes	579
10.3.2	Using Color	587
10.3.3	Using Fonts	589
10.3.4	Displaying Images	597
10.4	Event Handling	598
10.4.1	Basic Event Handling Concepts	598
10.4.2	Example: Handling a Button Click	600
10.4.3	Specifying Listeners Concisely	604
10.4.4	Adapter Classes	605
10.4.5	Actions	608
10.4.6	Mouse Events	614
10.4.7	The AWT Event Hierarchy	620
10.5	The Preferences API	624
Chapter 11: User Interface Components with Swing		631
11.1	Swing and the Model-View-Controller Design Pattern	632
11.2	Introduction to Layout Management	636

11.2.1	Layout Managers	637
11.2.2	Border Layout	639
11.2.3	Grid Layout	642
11.3	Text Input	643
11.3.1	Text Fields	643
11.3.2	Labels and Labeling Components	645
11.3.3	Password Fields	647
11.3.4	Text Areas	647
11.3.5	Scroll Panes	648
11.4	Choice Components	651
11.4.1	Checkboxes	651
11.4.2	Radio Buttons	654
11.4.3	Borders	658
11.4.4	Combo Boxes	661
11.4.5	Sliders	665
11.5	Menus	671
11.5.1	Menu Building	672
11.5.2	Icons in Menu Items	675
11.5.3	Checkbox and Radio Button Menu Items	676
11.5.4	Pop-Up Menus	677
11.5.5	Keyboard Mnemonics and Accelerators	679
11.5.6	Enabling and Disabling Menu Items	682
11.5.7	Toolbars	687
11.5.8	Tooltips	689
11.6	Sophisticated Layout Management	690
11.6.1	The Grid Bag Layout	691
11.6.1.1	The gridx, gridy, gridwidth, and gridheight Parameters	693
11.6.1.2	Weight Fields	694
11.6.1.3	The fill and anchor Parameters	694
11.6.1.4	Padding	694
11.6.1.5	Alternative Method to Specify the gridx, gridy, gridwidth, and gridheight Parameters	695
11.6.1.6	A Grid Bag Layout Recipe	695
11.6.1.7	A Helper Class to Tame the Grid Bag Constraints	696

11.6.2	Custom Layout Managers	702
11.7	Dialog Boxes	706
11.7.1	Option Dialogs	707
11.7.2	Creating Dialogs	712
11.7.3	Data Exchange	716
11.7.4	File Dialogs	723
Chapter 12:	Concurrency	733
12.1	What Are Threads?	734
12.2	Thread States	739
12.2.1	New Threads	740
12.2.2	Runnable Threads	740
12.2.3	Blocked and Waiting Threads	741
12.2.4	Terminated Threads	742
12.3	Thread Properties	743
12.3.1	Interrupting Threads	743
12.3.2	Daemon Threads	746
12.3.3	Thread Names	747
12.3.4	Handlers for Uncaught Exceptions	747
12.3.5	Thread Priorities	749
12.4	Synchronization	750
12.4.1	An Example of a Race Condition	750
12.4.2	The Race Condition Explained	752
12.4.3	Lock Objects	755
12.4.4	Condition Objects	758
12.4.5	The synchronized Keyword	764
12.4.6	Synchronized Blocks	768
12.4.7	The Monitor Concept	770
12.4.8	Volatile Fields	771
12.4.9	Final Variables	772
12.4.10	Atomics	773
12.4.11	Deadlocks	775
12.4.12	Thread-Local Variables	778
12.4.13	Why the stop and suspend Methods Are Deprecated	779
12.5	Thread-Safe Collections	781
12.5.1	Blocking Queues	781

12.5.2	Efficient Maps, Sets, and Queues	789
12.5.3	Atomic Update of Map Entries	790
12.5.4	Bulk Operations on Concurrent Hash Maps	794
12.5.5	Concurrent Set Views	796
12.5.6	Copy on Write Arrays	797
12.5.7	Parallel Array Algorithms	797
12.5.8	Older Thread-Safe Collections	799
12.6	Tasks and Thread Pools	800
12.6.1	Callables and Futures	800
12.6.2	Executors	802
12.6.3	Controlling Groups of Tasks	806
12.6.4	The Fork-Join Framework	811
12.7	Asynchronous Computations	814
12.7.1	Completable Futures	815
12.7.2	Composing Completable Futures	817
12.7.3	Long-Running Tasks in User Interface Callbacks	823
12.8	Processes	831
12.8.1	Building a Process	832
12.8.2	Running a Process	834
12.8.3	Process Handles	835
Appendix	839
<i>Index</i>	<i>843</i>

This page intentionally left blank

Preface



To the Reader

In late 1995, the Java programming language burst onto the Internet scene and gained instant celebrity status. The promise of Java technology was that it would become the *universal glue* that connects users with information wherever it comes from—web servers, databases, information providers, or any other imaginable source. Indeed, Java is in a unique position to fulfill this promise. It is an extremely solidly engineered language that has gained wide acceptance. Its built-in security and safety features are reassuring both to programmers and to the users of Java programs. Java has built-in support for advanced programming tasks, such as network programming, database connectivity, and concurrency.

Since 1995, eleven major revisions of the Java Development Kit have been released. Over the course of the last 20 years, the Application Programming Interface (API) has grown from about 200 to over 4,000 classes. The API now spans such diverse areas as user interface construction, database management, internationalization, security, and XML processing.

The book that you are reading right now is the first volume of the eleventh edition of *Core Java*. Each edition closely followed a release of the Java Development Kit, and each time, we rewrote the book to take advantage of the newest Java features. This edition has been updated to reflect the features of Java Standard Edition (SE) 9, 10, and 11.

As with the previous editions of this book, *we still target serious programmers who want to put Java to work on real projects*. We think of you, our reader, as a programmer with a solid background in a programming language other than Java, and we assume that you don't like books filled with toy examples (such as toasters, zoo animals, or "nervous text"). You won't find any of these in our book. Our goal is to enable you to fully understand the Java language and library, not to give you an illusion of understanding.

In this book you will find lots of sample code demonstrating almost every language and library feature that we discuss. We keep the sample programs purposefully simple to focus on the major points, but, for the most part, they

aren't fake and they don't cut corners. They should make good starting points for your own code.

We assume you are willing, even eager, to learn about all the advanced features that Java puts at your disposal. For example, we give you a detailed treatment of

- Object-oriented programming
- Reflection and proxies
- Interfaces and inner classes
- Exception handling
- Generic programming
- The collections framework
- The event listener model
- Graphical user interface design
- Concurrency

With the explosive growth of the Java class library, a one-volume treatment of all the features of Java that serious programmers need to know is no longer possible. Hence, we decided to break up the book into two volumes. This first volume concentrates on the fundamental concepts of the Java language, along with the basics of user-interface programming. The second volume, *Core Java, Volume II—Advanced Features*, goes further into the enterprise features and advanced user-interface programming. It includes detailed discussions of

- The Stream API
- File processing and regular expressions
- Databases
- XML processing
- Annotations
- Internationalization
- Network programming
- Advanced GUI components
- Advanced graphics
- Native methods

When writing a book, errors and inaccuracies are inevitable. We'd very much like to know about them. But, of course, we'd prefer to learn about each of them only once. We have put up a list of frequently asked questions, bug fixes, and workarounds on a web page at <http://horstmann.com/corejava>. Strategically placed at the end of the errata page (to encourage you to read through

it first) is a form you can use to report bugs and suggest improvements. Please don't be disappointed if we don't answer every query or don't get back to you immediately. We do read all e-mail and appreciate your input to make future editions of this book clearer and more informative.

A Tour of This Book

Chapter 1 gives an overview of the capabilities of Java that set it apart from other programming languages. We explain what the designers of the language set out to do and to what extent they succeeded. Then, we give a short history of how Java came into being and how it has evolved.

In **Chapter 2**, we tell you how to download and install the JDK and the program examples for this book. Then we guide you through compiling and running a console application and a graphical application. You will see how to use the plain JDK, a Java IDE, and the JShell tool.

Chapter 3 starts the discussion of the Java language. In this chapter, we cover the basics: variables, loops, and simple functions. If you are a C or C++ programmer, this is smooth sailing because the syntax for these language features is essentially the same as in C. If you come from a non-C background such as Visual Basic, you will want to read this chapter carefully.

Object-oriented programming (OOP) is now in the mainstream of programming practice, and Java is an object-oriented programming language. **Chapter 4** introduces encapsulation, the first of two fundamental building blocks of object orientation, and the Java language mechanism to implement it—that is, classes and methods. In addition to the rules of the Java language, we also give advice on sound OOP design. Finally, we cover the marvelous javadoc tool that formats your code comments as a set of hyperlinked web pages. If you are familiar with C++, you can browse through this chapter quickly. Programmers coming from a non-object-oriented background should expect to spend some time mastering the OOP concepts before going further with Java.

Classes and encapsulation are only one part of the OOP story, and **Chapter 5** introduces the other—namely, *inheritance*. Inheritance lets you take an existing class and modify it according to your needs. This is a fundamental technique for programming in Java. The inheritance mechanism in Java is quite similar to that in C++. Once again, C++ programmers can focus on the differences between the languages.

Chapter 6 shows you how to use Java's notion of an *interface*. Interfaces let you go beyond the simple inheritance model of Chapter 5. Mastering interfaces allows you to have full access to the power of Java's completely object-oriented approach to programming. After we cover interfaces, we move on to *lambda expressions*, a concise way for expressing a block of code that can be executed at a later point in time. We then cover a useful technical feature of Java called *inner classes*.

Chapter 7 discusses *exception handling*—Java's robust mechanism to deal with the fact that bad things can happen to good programs. Exceptions give you an efficient way of separating the normal processing code from the error handling. Of course, even after hardening your program by handling all exceptional conditions, it still might fail to work as expected. In the final part of this chapter, we give you a number of useful debugging tips.

Chapter 8 gives an overview of generic programming. Generic programming makes your programs easier to read and safer. We show you how to use strong typing and remove unsightly and unsafe casts, and how to deal with the complexities that arise from the need to stay compatible with older versions of Java.

The topic of **Chapter 9** is the collections framework of the Java platform. Whenever you want to collect multiple objects and retrieve them later, you should use a collection that is best suited for your circumstances, instead of just tossing the elements into an array. This chapter shows you how to take advantage of the standard collections that are prebuilt for your use.

Chapter 10 provides an introduction into GUI programming. We show how you can make windows, how to paint on them, how to draw with geometric shapes, how to format text in multiple fonts, and how to display images. Next, you'll see how to write code that responds to events, such as mouse clicks or key presses.

Chapter 11 discusses the Swing GUI toolkit in great detail. The Swing toolkit allows you to build cross-platform graphical user interfaces. You'll learn all about the various kinds of buttons, text components, borders, sliders, list boxes, menus, and dialog boxes. However, some of the more advanced components are discussed in Volume II.

Chapter 12 finishes the book with a discussion of concurrency, which enables you to program tasks to be done in parallel. This is an important and exciting application of Java technology in an era where most processors have multiple cores that you want to keep busy.

A **bonus JavaFX chapter** contains a rapid introduction into JavaFX, a modern GUI toolkit for desktop applications. If you read the print book, download the chapter from the book companion site at <http://horstmann.com/corejava>.

The **Appendix** lists the reserved words of the Java language.

Conventions

As is common in many computer books, we use monospace type to represent computer code.



NOTE: Notes are tagged with “note” icons that look like this.



TIP: Tips are tagged with “tip” icons that look like this.



CAUTION: When there is danger ahead, we warn you with a “caution” icon.



C++ NOTE: There are many C++ notes that explain the differences between Java and C++. You can skip over them if you don’t have a background in C++ or if you consider your experience with that language a bad dream of which you’d rather not be reminded.

Java comes with a large programming library, or Application Programming Interface (API). When using an API call for the first time, we add a short summary description at the end of the section. These descriptions are a bit more informal but, we hope, also a little more informative than those in the official online API documentation. The names of interfaces are in italics, just like in the official documentation. The number after a class, interface, or method name is the JDK version in which the feature was introduced, as shown in the following example:

Application Programming Interface 9

Programs whose source code is on the book's companion web site are presented as listings, for instance:

Listing 1.1 InputTest/InputTest.java

Sample Code

The web site for this book at <http://horstmann.com/corejava> contains all sample code from the book. See Chapter 2 for more information on installing the Java Development Kit and the sample code.

Register your copy of *Core Java, Volume I—Fundamentals, Eleventh Edition*, on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780135166307) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

Acknowledgments



Writing a book is always a monumental effort, and rewriting it doesn't seem to be much easier, especially with the continuous change in Java technology. Making a book a reality takes many dedicated people, and it is my great pleasure to acknowledge the contributions of the entire *Core Java* team.

A large number of individuals at Pearson provided valuable assistance but managed to stay behind the scenes. I'd like them all to know how much I appreciate their efforts. As always, my warm thanks go to my editor, Greg Doench, for steering the book through the writing and production process, and for allowing me to be blissfully unaware of the existence of all those folks behind the scenes. I am very grateful to Julie Nahil for production support, and to Dmitry Kirsanov and Alina Kirsanova for copyediting and typesetting the manuscript. My thanks also to my coauthor of earlier editions, Gary Cornell, who has since moved on to other ventures.

Thanks to the many readers of earlier editions who reported embarrassing errors and made lots of thoughtful suggestions for improvement. I am particularly grateful to the excellent reviewing team who went over the manuscript with an amazing eye for detail and saved me from many embarrassing errors.

Reviewers of this and earlier editions include Chuck Allison (Utah Valley University), Lance Andersen (Oracle), Paul Anderson (Anderson Software Group), Alec Beaton (IBM), Cliff Berg, Andrew Binstock (Oracle), Joshua Bloch, David Brown, Corky Cartwright, Frank Cohen (PushToTest), Chris Crane (devXsolution), Dr. Nicholas J. De Lillo (Manhattan College), Rakesh Dhoopar (Oracle), David Geary (Clarity Training), Jim Gish (Oracle), Brian Goetz (Oracle), Angela Gordon, Dan Gordon (Electric Cloud), Rob Gordon, John Gray (University of Hartford), Cameron Gregory (olabs.com), Marty Hall (coreservlets.com, Inc.), Vincent Hardy (Adobe Systems), Dan Harkey (San Jose State University), William Higgins (IBM), Vladimir Ivanovic (PointBase), Jerry Jackson (CA Technologies), Tim Kimmet (Walmart), Chris Laffra, Charlie Lai (Apple), Angelika Langer, Doug Langston, Hang Lau (McGill University), Mark Lawrence, Doug Lea (SUNY Oswego), Gregory Longshore, Bob Lynch (Lynch Associates), Philip Milne (consultant), Mark Morrissey (The Oregon Graduate Institute), Mahesh Neelakanta (Florida Atlantic University), Hao Pham, Paul Phillion, Blake Ragsdell, Stuart Reges (University of Arizona), Simon Ritter (Azul Systems), Rich Rosen (Interactive Data Corporation), Peter Sanders (ESSI University, Nice, France), Dr. Paul Sanghera (San

Jose State University and Brooks College), Paul Sevinc (Teamup AG), Devang Shah (Sun Microsystems), Yoshiki Shibata, Bradley A. Smith, Steven Stelling (Oracle), Christopher Taylor, Luke Taylor (Valtech), George Thiruvathukal, Kim Topley (StreamingEdge), Janet Traub, Paul Tyma (consultant), Peter van der Linden, Christian Ullenboom, Burt Walsh, Dan Xu (Oracle), and John Zavgren (Oracle).

Cay Horstmann
San Francisco, California
June 2018

An Introduction to Java

In this chapter

- 1.1 Java as a Programming Platform, page 1
- 1.2 The Java “White Paper” Buzzwords, page 2
- 1.3 Java Applets and the Internet, page 9
- 1.4 A Short History of Java, page 10
- 1.5 Common Misconceptions about Java, page 13

The first release of Java in 1996 generated an incredible amount of excitement, not just in the computer press, but in mainstream media such as the *New York Times*, the *Washington Post*, and *BusinessWeek*. Java has the distinction of being the first and only programming language that had a ten-minute story on National Public Radio. A \$100,000,000 venture capital fund was set up solely for products using a *specific* computer language. I hope you will enjoy a brief history of Java that you will find in this chapter.

1.1 Java as a Programming Platform

In the first edition of this book, my coauthor Gary Cornell and I had this to write about Java:

“As a computer language, Java’s hype is overdone: Java is certainly a *good* programming language. There is no doubt that it is one of the better languages available to serious programmers. We think it could *potentially* have been a

great programming language, but it is probably too late for that. Once a language is out in the field, the ugly reality of compatibility with existing code sets in.”

Our editor got a lot of flack for this paragraph from someone very high up at Sun Microsystems, the company that originally developed Java. The Java language has a lot of nice features that we will examine in detail later in this chapter. It has its share of warts, and some of the newer additions to the language are not as elegant as the original features because of compatibility requirements.

But, as we already said in the first edition, Java was never just a language. There are lots of programming languages out there, but few of them make much of a splash. Java is a whole *platform*, with a huge library, containing lots of reusable code, and an execution environment that provides services such as security, portability across operating systems, and automatic garbage collection.

As a programmer, you will want a language with a pleasant syntax and comprehensible semantics (i.e., not C++). Java fits the bill, as do dozens of other fine languages. Some languages give you portability, garbage collection, and the like, but they don’t have much of a library, forcing you to roll your own if you want fancy graphics or networking or database access. Well, Java has everything—a good language, a high-quality execution environment, and a vast library. That combination is what makes Java an irresistible proposition to so many programmers.

1.2 The Java “White Paper” Buzzwords

The authors of Java wrote an influential white paper that explains their design goals and accomplishments. They also published a shorter overview that is organized along the following 11 buzzwords:

1. Simple
2. Object-Oriented
3. Distributed
4. Robust
5. Secure
6. Architecture-Neutral
7. Portable
8. Interpreted

9. High-Performance
10. Multithreaded
11. Dynamic

In the following subsections, you will find a summary, with excerpts from the white paper, of what the Java designers say about each buzzword, together with a commentary based on my experiences with the current version of Java.



NOTE: The white paper can be found at www.oracle.com/technetwork/java/langenv-140151.html. You can retrieve the overview with the 11 buzzwords at <http://horstmann.com/corejava/java-an-overview/7Gosling.pdf>.

1.2.1 Simple

We wanted to build a system that could be programmed easily without a lot of esoteric training and which leveraged today's standard practice. So even though we found that C++ was unsuitable, we designed Java as closely to C++ as possible in order to make the system more comprehensible. Java omits many rarely used, poorly understood, confusing features of C++ that, in our experience, bring more grief than benefit.

The syntax for Java is, indeed, a cleaned-up version of C++ syntax. There is no need for header files, pointer arithmetic (or even a pointer syntax), structures, unions, operator overloading, virtual base classes, and so on. (See the C++ notes interspersed throughout the text for more on the differences between Java and C++.) The designers did not, however, attempt to fix all of the clumsy features of C++. For example, the syntax of the `switch` statement is unchanged in Java. If you know C++, you will find the transition to the Java syntax easy.

At the time Java was released, C++ was actually not the most commonly used programming language. Many developers used Visual Basic and its drag-and-drop programming environment. These developers did not find Java simple. It took several years for Java development environments to catch up. Nowadays, Java development environments are far ahead of those for most other programming languages.

Another aspect of being simple is being small. One of the goals of Java is to enable the construction of software that can run stand-alone on small machines. The size of the basic interpreter and class support is about 40K; the basic standard libraries and thread support (essentially a self-contained microkernel) add another 175K.

This was a great achievement at the time. Of course, the library has since grown to huge proportions. There is now a separate Java Micro Edition with a smaller library, suitable for embedded devices.

1.2.2 Object-Oriented

Simply stated, object-oriented design is a programming technique that focuses on the data—objects—and on the interfaces to those objects. To make an analogy with carpentry, an “object-oriented” carpenter would be mostly concerned with the chair he is building, and secondarily with the tools used to make it; a “non-object-oriented” carpenter would think primarily of his tools. The object-oriented facilities of Java are essentially those of C++.

Object orientation was pretty well established when Java was developed. The object-oriented features of Java are comparable to those of C++. The major difference between Java and C++ lies in multiple inheritance, which Java has replaced with a simpler concept of interfaces. Java has a richer capacity for runtime introspection (discussed in Chapter 5) than C++.

1.2.3 Distributed

Java has an extensive library of routines for coping with TCP/IP protocols like HTTP and FTP. Java applications can open and access objects across the Net via URLs with the same ease as when accessing a local file system.

Nowadays, one takes this for granted—but in 1995, connecting to a web server from a C++ or Visual Basic program was a major undertaking.

1.2.4 Robust

Java is intended for writing programs that must be reliable in a variety of ways. Java puts a lot of emphasis on early checking for possible problems, later dynamic (runtime) checking, and eliminating situations that are error-prone. . . . The single biggest difference between Java and C/C++ is that Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data.

The Java compiler detects many problems that in other languages would show up only at runtime. As for the second point, anyone who has spent hours chasing memory corruption caused by a pointer bug will be very happy with this aspect of Java.

1.2.5 Secure

Java is intended to be used in networked/distributed environments. Toward that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems.

From the beginning, Java was designed to make certain kinds of attacks impossible, among them:

- Overrunning the runtime stack—a common attack of worms and viruses
- Corrupting memory outside its own process space
- Reading or writing files without permission

Originally, the Java attitude towards downloaded code was “Bring it on!” Untrusted code was executed in a sandbox environment where it could not impact the host system. Users were assured that nothing bad could happen because Java code, no matter where it came from, could never escape from the sandbox.

However, the security model of Java is complex. Not long after the first version of the Java Development Kit was shipped, a group of security experts at Princeton University found subtle bugs that allowed untrusted code to attack the host system.

Initially, security bugs were fixed quickly. Unfortunately, over time, hackers got quite good at spotting subtle flaws in the implementation of the security architecture. Sun, and then Oracle, had a tough time keeping up with bug fixes.

After a number of high-profile attacks, browser vendors and Oracle became increasingly cautious. Java browser plug-ins no longer trust remote code unless it is digitally signed and users have agreed to its execution.



NOTE: Even though in hindsight, the Java security model was not as successful as originally envisioned, Java was well ahead of its time. A competing code delivery mechanism from Microsoft relied on digital signatures alone for security. Clearly this was not sufficient: As any user of Microsoft’s own products can confirm, programs from well-known vendors do crash and create damage.

1.2.6 Architecture-Neutral

The compiler generates an architecture-neutral object file format. The compiled code is executable on many processors, given the presence of the Java runtime system. The Java compiler does this by generating bytecode instructions which have nothing to do with a particular computer architecture. Rather, they are designed to be both easy to interpret on any machine and easy to translate into native machine code on the fly.

Generating code for a “virtual machine” was not a new idea at the time. Programming languages such as Lisp, Smalltalk, and Pascal had employed this technique for many years.

Of course, interpreting virtual machine instructions is slower than running machine instructions at full speed. However, virtual machines have the option of translating the most frequently executed bytecode sequences into machine code—a process called just-in-time compilation.

Java’s virtual machine has another advantage. It increases security because it can check the behavior of instruction sequences.

1.2.7 Portable

Unlike C and C++, there are no “implementation-dependent” aspects of the specification. The sizes of the primitive data types are specified, as is the behavior of arithmetic on them.

For example, an `int` in Java is always a 32-bit integer. In C/C++, `int` can mean a 16-bit integer, a 32-bit integer, or any other size that the compiler vendor likes. The only restriction is that the `int` type must have at least as many bytes as a `short int` and cannot have more bytes than a `long int`. Having a fixed size for number types eliminates a major porting headache. Binary data is stored and transmitted in a fixed format, eliminating confusion about byte ordering. Strings are saved in a standard Unicode format.

The libraries that are a part of the system define portable interfaces. For example, there is an abstract `Window` class and implementations of it for UNIX, Windows, and the Macintosh.

The example of a `Window` class was perhaps poorly chosen. As anyone who has ever tried knows, it is an effort of heroic proportions to implement a user interface that looks good on Windows, the Macintosh, and ten flavors of UNIX. Java 1.0 made the heroic effort, delivering a simple toolkit that provided common user interface elements on a number of platforms. Unfortunately, the result was a library that, with a lot of work, could give barely acceptable

results on different systems. That initial user interface toolkit has since been replaced, and replaced again, and portability across platforms remains an issue.

However, for everything that isn't related to user interfaces, the Java libraries do a great job of letting you work in a platform-independent manner. You can work with files, regular expressions, XML, dates and times, databases, network connections, threads, and so on, without worrying about the underlying operating system. Not only are your programs portable, but the Java APIs are often of higher quality than the native ones.

1.2.8 Interpreted

The Java interpreter can execute Java bytecodes directly on any machine to which the interpreter has been ported. Since linking is a more incremental and lightweight process, the development process can be much more rapid and exploratory.

This was a real stretch. Anyone who has used Lisp, Smalltalk, Visual Basic, Python, R, or Scala knows what a “rapid and exploratory” development process is. You try out something, and you instantly see the result. For the first 20 years of Java's existence, development environments were not focused on that experience. It wasn't until Java 9 that the `jshell` tool supported rapid and exploratory programming.

1.2.9 High-Performance

While the performance of interpreted bytecodes is usually more than adequate, there are situations where higher performance is required. The bytecodes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on.

In the early years of Java, many users disagreed with the statement that the performance was “more than adequate.” Today, however, the just-in-time compilers have become so good that they are competitive with traditional compilers and, in some cases, even outperform them because they have more information available. For example, a just-in-time compiler can monitor which code is executed frequently and optimize just that code for speed. A more sophisticated optimization is the elimination (or “inlining”) of function calls. The just-in-time compiler knows which classes have been loaded. It can use inlining when, based upon the currently loaded collection of classes, a particular function is never overridden, and it can undo that optimization later if necessary.

1.2.10 Multithreaded

[The] benefits of multithreading are better interactive responsiveness and real-time behavior.

Nowadays, we care about concurrency because Moore's law has come to an end. Instead of faster processors, we just get more of them, and we have to keep them busy. Yet when you look at most programming languages, they show a shocking disregard for this problem.

Java was well ahead of its time. It was the first mainstream language to support concurrent programming. As you can see from the white paper, its motivation was a little different. At the time, multicore processors were exotic, but web programming had just started, and processors spent a lot of time waiting for a response from the server. Concurrent programming was needed to make sure the user interface didn't freeze.

Concurrent programming is never easy, but Java has done a very good job making it manageable.

1.2.11 Dynamic

In a number of ways, Java is a more dynamic language than C or C++. It was designed to adapt to an evolving environment. Libraries can freely add new methods and instance variables without any effect on their clients. In Java, finding out runtime type information is straightforward.

This is an important feature in situations where code needs to be added to a running program. A prime example is code that is downloaded from the Internet to run in a browser. In C or C++, this is indeed a major challenge, but the Java designers were well aware of dynamic languages that made it easy to evolve a running program. Their achievement was to bring this feature to a mainstream programming language.



NOTE: Shortly after the initial success of Java, Microsoft released a product called J++ with a programming language and virtual machine that were almost identical to Java. This effort failed to gain traction, and Microsoft followed through with another language called C# that also has many similarities to Java but runs on a different virtual machine. This book does not cover J++ or C#.

1.3 Java Applets and the Internet

The idea here is simple: Users will download Java bytecodes from the Internet and run them on their own machines. Java programs that work on web pages are called *applets*. To use an applet, you only need a Java-enabled web browser, which will execute the bytecodes for you. You need not install any software. You get the latest version of the program whenever you visit the web page containing the applet. Most importantly, thanks to the security of the virtual machine, you never need to worry about attacks from hostile code.

Inserting an applet into a web page works much like embedding an image. The applet becomes a part of the page, and the text flows around the space used for the applet. The point is, this image is *alive*. It reacts to user commands, changes its appearance, and exchanges data between the computer presenting the applet and the computer serving it.

Figure 1.1 shows the Jmol applet that displays molecular structures. By using the mouse, you can rotate and zoom each molecule to better understand its structure. At the time that applets were invented, this kind of direct manipulation was not achievable with web pages—there was only rudimentary JavaScript and no HTML canvas.

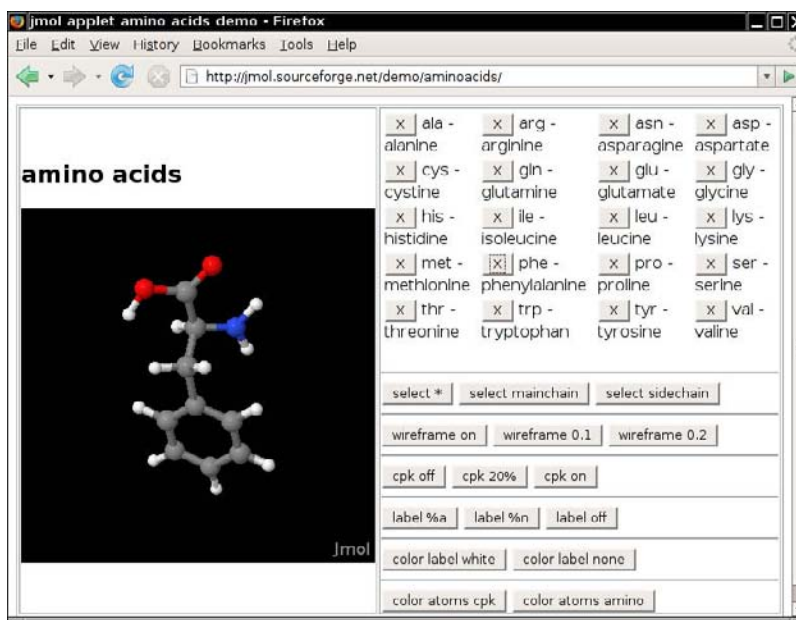


Figure 1.1 The Jmol applet